Alpha 21164 Microprocessor

Hardware Reference Manual

Order Number: EC–QAEQB–TE

Revision/Update Information: This preliminary document supersedes the *Alpha 21164 Microprocessor Hardware Reference Manual* EC–QAEQA–TE.

Digital Equipment Corporation
Maynard, Massachusetts

April 1995

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

Preface xxi

1 Introduction

1.1	The Architecture	1–1
1.1.1	Addressing	1–2
1.1.2	Integer Data Types	1–2
1.1.3	Floating-Point Data Types	1–3
1.2	Alpha 21164 Microprocessor Features	1–3

2 Internal Architecture

2.1	Alpha 21164 Microarchitecture	2–2
2.1.1	Instruction Fetch/Decode Unit and Branch Unit	2–4
2.1.1.1	Instruction Decode and Issue	2–4
2.1.1.2	Instruction Prefetch	2–5
2.1.1.3	Branch Execution	2–6
2.1.1.4	Instruction Translation Buffer	2–7
2.1.1.5	Interrupts	2–8
2.1.2	Integer Execution Unit	2–9
2.1.3	Floating-Point Execution Unit	2–10
2.1.4	Memory Address Translation Unit	2–10
2.1.4.1	Data Translation Buffer	2–11
2.1.4.2	Load Instruction and the Miss Address File	2–11
2.1.4.3	Dcache Control and Store Instructions	2–12
2.1.4.4	Write Buffer	2–12
2.1.5	Cache Control and Bus Interface Unit	2–12
2.1.6	Cache Organization	2–13
2.1.6.1	Data Cache	2–13
2.1.6.2	Instruction Cache	2–13
2.1.6.3	Second-Level Cache	2–13
2.1.6.4	External Cache	2–14
2.1.7	Serial Read-Only Memory Interface	2–14

2.2 Pipeline Organization 2–14
2.2.1 Pipeline Stages and Instruction Issue 2–18
2.2.2 Aborts and Exceptions 2–18
2.2.3 Nonissue Conditions 2–20
2.3 Scheduling and Issuing Rules 2–20
2.3.1 Instruction Class Definition and Instruction Slotting 2–20
2.3.2 Coding Guidelines 2–23
2.3.3 Instruction Latencies 2–24
2.3.3.1 Producer–Producer Latency 2–27
2.3.4 Issue Rules 2–28
2.4 Replay Traps 2–29
2.5 Miss Address File and Load-Merging Rules 2–30
2.5.1 Merging Rules 2–30
2.5.2 Read Requests to the Cbox 2–31
2.5.3 Load Instructions to Noncacheable Space 2–31
2.5.4 MAF Entries and MAF Full Conditions 2–32
2.5.5 Fill Operation 2–32
2.6 Mbox Store Instruction Execution 2–33
2.7 Write Buffer and the WMB Instruction 2–35
2.7.1 The Write Buffer 2–35
2.7.2 The Write Memory Barrier (WMB) Instruction 2–35
2.7.3 Entry-Pointer Queues 2–36
2.7.4 Write Buffer Entry Processing 2–36
2.7.5 Ordering of Noncacheable Space Write Instructions 2–37
2.8 Performance Measurement Support–Performance Counters 2–38
2.9 Floating-Point Control Register 2–38
2.10 Design Examples 2–40

3 Hardware Interface

3.1 Alpha 21164 Microprocessor Logic Symbol 3–1
3.2 Alpha 21164 Signal Names and Functions 3–3

4 Clocks, Cache, and External Interface Functional Description

4.1 Introduction to the External Interface 4–2
4.1.1 System Interface 4–2
4.1.1.1 Commands and Addresses 4–4
4.1.2 Bcache Interface 4–4
4.2 Clocks 4–5
4.2.1 CPU Clock 4–5
4.2.2 System Clock 4–6
4.2.3 Delayed System Clock 4–8

4.2.4 Reference Clock 4–8
4.2.4.1 Reference Clock Examples 4–9
4.2.4.1.1 Case 1: **ref_clk_in_h** Initially Sampled Low by DPLL 4–10
4.2.4.1.2 Case 2: **ref_clk_in_h** Initially Sampled High by DPLL 4–11
4.3 Physical Address Considerations 4–12
4.3.1 Physical Address Regions 4–12
4.3.2 Data Wrapping 4–13
4.3.3 Noncached Read Operations 4–14
4.3.4 Noncached Write Operations 4–14
4.4 Bcache Structure 4–15
4.4.1 Duplicate Tag Store 4–15
4.4.1.1 Full Duplicate Tag Store 4–16
4.4.1.2 Partial Scache Duplicate Tag Store 4–18
4.4.2 Bcache Victim Buffers 4–18
4.5 Systems Without a Bcache 4–19
4.6 Cache Coherency 4–19
4.6.1 Cache Coherency Basics 4–19
4.6.2 Write Invalidate Cache Coherency Protocol Systems 4–22
4.6.3 Write Invalidate Cache Coherency States 4–23
4.6.3.1 Write Invalidate Protocol State Machines 4–24
4.6.4 Flush Cache Coherency Protocol Systems 4–25
4.6.5 Flush-Based Protocol State Machines 4–27
4.6.6 Cache Coherency Transaction Conflicts 4–28
4.6.6.1 Case 1 4–28
4.6.6.2 Case 2 4–29
4.7 Lock Mechanisms 4–30
4.8 Alpha 21164-to-Bcache Transactions 4–31
4.8.1 Bcache Timing 4–31
4.8.2 Bcache Read Transaction (Private Read Operation) 4–32
4.8.3 Wave Pipeline 4–33
4.8.4 Bcache Write Transaction (Private Write Operation) 4–34
4.8.5 Selecting Bcache Options 4–35
4.9 Alpha 21164-Initiated System Transactions 4–36
4.9.1 READ MISS—No Bcache 4–40
4.9.2 READ MISS—Bcache 4–41
4.9.3 FILL 4–43
4.9.4 READ MISS with Victim 4–43
4.9.4.1 READ MISS with Victim (Victim Buffer) 4–44
4.9.4.2 READ MISS with Victim (Without Victim Buffer) 4–46
4.9.5 WRITE BLOCK and WRITE BLOCK LOCK 4–48
4.9.6 SET DIRTY and LOCK 4–50

4.9.7 Memory Barrier (MB) 4–52
4.9.7.1 When to Use a MEMORY BARRIER Command 4–52
4.9.8 FETCH 4–52
4.9.9 FETCH_M 4–52
4.10 System-Initiated Transactions 4–53
4.10.1 Sending Commands to the 21164 4–53
4.10.2 Write Invalidate Protocol Commands 4–55
4.10.2.1 Alpha 21164 Responses to Write Invalidate Protocol Commands 4–56
4.10.2.2 READ DIRTY and READ DIRTY/INVALIDATE 4–58
4.10.2.3 INVALIDATE 4–60
4.10.2.4 SET SHARED 4–62
4.10.3 Flush-Based Cache Coherency Protocol Commands 4–64
4.10.3.1 Alpha 21164 Responses to Flush-Based Protocol Commands 4–65
4.10.3.2 FLUSH 4–66
4.10.3.3 READ 4–68
4.11 Data Bus and Command/Address Bus Contention 4–70
4.11.1 Command/Address Bus 4–70
4.11.2 Read/Write Spacing—Data Bus Contention 4–71
4.11.3 Using **idle_bc_h** and **fill_h** 4–72
4.11.4 Using **data_bus_req_h** 4–74
4.11.5 Tristate Overlap 4–75
4.11.5.1 READ or WRITE to FILL 4–75
4.11.5.2 BCACHE VICTIM to FILL 4–75
4.11.5.3 System Bcache Command to FILL 4–78
4.11.5.4 FILL to Private Read or Write Operation 4–80
4.12 Alpha 21164 Interface Restrictions 4–81
4.12.1 FILL Operations after Other Transactions 4–81
4.12.2 Command Acknowledge for WRITE BLOCK Commands 4–81
4.12.3 Systems Without a Bcache 4–81
4.12.4 Fast Probes with No Bcache 4–81
4.12.5 WRITE BLOCK LOCK 4–82
4.13 Alpha 21164/System Race Conditions 4–83
4.13.1 Rules for 21164 and System Use of External Interface 4–83
4.13.2 READ MISS with Victim Example 4–84
4.13.3 **idle_bc_h** and **cack_h** Race Example 4–86
4.13.4 READ MISS with **idle_bc_h** Asserted Example 4–88
4.13.5 READ MISS with Victim Abort Example 4–89
4.13.6 Bcache Hit Under READ MISS Example 4–90
4.14 Data Integrity, Bcache Errors, and Command/Address Errors . . . 4–92
4.14.1 Data ECC and Parity 4–92
4.14.2 Force Correction 4–94

4.14.3 Bcache Tag Data Parity 4–94
4.14.4 Bcache Tag Control Parity 4–94
4.14.5 Address and Command Parity 4–95
4.14.6 Fill Error 4–95
4.14.7 Forcing 21164 Reset 4–95
4.15 Interrupts 4–96
4.15.1 Interrupt Signals During Initialization 4–96
4.15.2 Interrupt Signals During Normal Operation 4–96
4.15.3 Interrupt Priority Level 4–96

5 Internal Processor Registers

5.1 Instruction Fetch/Decode Unit and Branch Unit (Ibox) IPRs 5–5
5.1.1 Istream Translation Buffer Tag Register (ITB_TAG) 5–5
5.1.2 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register 5–6
5.1.3 Instruction Translation Buffer Address Space Number (ITB_ASN) Register 5–8
5.1.4 Instruction Translation Buffer Page Table Entry Temporary (ITB_PTE_TEMP) Register 5–9
5.1.5 Instruction Translation Buffer Invalidate All Process (ITB_IAP) Register 5–9
5.1.6 Instruction Translation Buffer Invalidate All (ITB_IA) Register 5–9
5.1.7 Instruction Translation Buffer IS (ITB_IS) Register 5–10
5.1.8 Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register 5–11
5.1.9 Virtual Page Table Base Register (IVPTBR) 5–12
5.1.10 Icache Parity Error Status (ICPERR_STAT) Register 5–13
5.1.11 Icache Flush Control (IC_FLUSH_CTL) Register 5–13
5.1.12 Exception Address (EXC_ADDR) Register 5–14
5.1.13 Exception Summary (EXC_SUM) Register 5–15
5.1.14 Exception Mask (EXC_MASK) Register 5–17
5.1.15 PAL Base Address (PAL_BASE) Register 5–18
5.1.16 Ibox Current Mode (ICM) Register 5–19
5.1.17 Ibox Control and Status Register (ICSR) 5–20
5.1.18 Interrupt Priority Level Register (IPLR) 5–23
5.1.19 Interrupt ID (INTID) Register 5–24
5.1.20 Asynchronous System Trap Request Register (ASTRR) 5–25
5.1.21 Asynchronous System Trap Enable Register (ASTER) 5–26
5.1.22 Software Interrupt Request Register (SIRR) 5–27
5.1.23 Hardware Interrupt Clear (HWINT_CLR) Register 5–28
5.1.24 Interrupt Summary Register (ISR) 5–29

5.1.25	Serial Line Transmit (SL_XMIT) Register	5–31
5.1.26	Serial Line Receive (SL_RCV) Register	5–32
5.1.27	Performance Counter (PMCTR) Register	5–33
5.2	Memory Address Translation Unit (Mbox) IPRs	5–38
5.2.1	Dstream Translation Buffer Address Space Number (DTB_ASN) Register	5–38
5.2.2	Dstream Translation Buffer Current Mode (DTB_CM) Register	5–39
5.2.3	Dstream Translation Buffer Tag (DTB_TAG) Register	5–40
5.2.4	Dstream Translation Buffer Page Table Entry (DTB_PTE) Register	5–41
5.2.5	Dstream Translation Buffer Page Table Entry Temporary (DTB_PTE_TEMP) Register	5–43
5.2.6	Dstream Memory Management Fault Status (MM_STAT) Register	5–44
5.2.7	Faulting Virtual Address (VA) Register	5–46
5.2.8	Formatted Virtual Address (VA_FORM) Register	5–47
5.2.9	Mbox Virtual Page Table Base Register (MVPTBR)	5–49
5.2.10	Dcache Parity Error Status (DC_PERR_STAT) Register	5–50
5.2.11	Dstream Translation Buffer Invalidate All Process (DTB_IAP) Register	5–52
5.2.12	Dstream Translation Buffer Invalidate All (DTB_IA) Register	5–52
5.2.13	Dstream Translation Buffer Invalidate Single (DTB_IS) Register	5–53
5.2.14	Mbox Control Register (MCSR)	5–54
5.2.15	Dcache Mode (DC_MODE) Register	5–56
5.2.16	Miss Address File Mode (MAF_MODE) Register	5–58
5.2.17	Dcache Flush (DC_FLUSH) Register	5–60
5.2.18	Alternate Mode (ALT_MODE) Register	5–60
5.2.19	Cycle Counter (CC) Register	5–61
5.2.20	Cycle Counter Control (CC_CTL) Register	5–62
5.2.21	Dcache Test Tag Control (DC_TEST_CTL) Register	5–63
5.2.22	Dcache Test Tag (DC_TEST_TAG) Register	5–64
5.2.23	Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register	5–66
5.3	External Interface Control (Cbox) IPRs	5–68
5.3.1	Scache Control (SC_CTL) Register (FF FFF0 00A8)	5–69
5.3.2	Scache Status (SC_STAT) Register (FF FFF0 00E8)	5–72
5.3.3	Scache Address (SC_ADDR) Register (FF FFF0 0188)	5–75
5.3.4	Bcache Control (BC_CONTROL) Register (FF FFF0 0128)	5–78

5.3.5 Bcache Configuration (BC_CONFIG) Register (FF FFF0 01C8) 5–84
5.3.6 Bcache Tag Address (BC_TAG_ADDR) Register (FF FFF0 0108) 5–89
5.3.7 External Interface Status (EI_STAT) Register (FF FFF0 0168) 5–91
5.3.8 External Interface Address (EI_ADDR) Register (FF FFF0 0148) 5–94
5.3.9 Fill Syndrome (FILL_SYN) Register (FF FFF0 0068) 5–95
5.4 PALcode Storage Registers 5–99
5.5 Restrictions 5–100
5.5.1 Cbox IPR PALcode Restrictions 5–100
5.5.2 PALcode Restrictions—Instruction Definitions 5–101

6 Privileged Architecture Library Code

6.1 PALcode Description 6–1
6.2 PALmode Environment 6–2
6.3 Invoking PALcode 6–3
6.4 PALcode Entry Points 6–5
6.4.1 CALL_PAL Entry 6–5
6.4.2 PALcode Trap Entry Points 6–6
6.5 Required PALcode Function Codes 6–7
6.6 Alpha 21164 Implementation of the Architecturally Reserved Opcodes 6–7
6.6.1 HW_LD Instruction 6–8
6.6.2 HW_ST Instruction 6–10
6.6.3 HW_REI Instruction 6–11
6.6.4 HW_MFPR and HW_MTPR Instructions 6–11

7 Initialization and Configuration

7.1 Input Signals **sys_reset_l** and **dc_ok_h** and Booting 7–1
7.1.1 Pin State with **dc_ok_h** Not Asserted 7–5
7.2 Sysclk Ratio and Delay 7–6
7.3 Built-In Self-Test (BiSt) 7–6
7.4 Serial Read-Only Memory Interface Port 7–6
7.4.1 Serial Instruction Cache Load Operation 7–7
7.5 Serial Terminal Port 7–8
7.6 Cache Initialization 7–8
7.6.1 Icache Initialization 7–8
7.6.2 Flushing Dirty Blocks 7–9
7.7 External Interface Initialization 7–10

7.8 Internal Processor Register Reset State 7–10
7.9 Timeout Reset 7–13
7.10 IEEE 1149.1 Test Port Reset 7–13

8 Error Detection and Error Handling

8.1 Error Flows 8–1
8.1.1 Icache Data or Tag Parity Error 8–1
8.1.2 Scache Data Parity Error—Istream 8–2
8.1.3 Scache Tag Parity Error—Istream 8–3
8.1.4 Scache Data Parity Error—Dstream Read/Write, READ_DIRTY 8–3
8.1.5 Scache Tag Parity Error—Dstream or System Commands . . . 8–4
8.1.6 Dcache Data Parity Error 8–4
8.1.7 Dcache Tag Parity Error 8–5
8.1.8 Istream Uncorrectable ECC or Data Parity Errors (Bcache or Memory) 8–5
8.1.9 Dstream Uncorrectable ECC or Data Parity Errors (Bcache or Memory) 8–6
8.1.10 Bcache Tag Parity Errors—Istream 8–7
8.1.11 Bcache Tag Parity Errors—Dstream 8–7
8.1.12 System Command/Address Parity Error 8–8
8.1.13 System Read Operations of the Bcache 8–8
8.1.14 Istream or Dstream Correctable ECC Error (Bcache or Memory) 8–9
8.1.15 Fill Timeout (FILL_ERROR_H) 8–9
8.1.16 System Machine Check 8–10
8.1.17 Ibox Timeout 8–10
8.1.18 cfail_h and Not cack_h 8–10
8.2 MCHK Flow 8–11
8.3 Processor-Correctable Error Interrupt Flow (IPL 31) 8–13
8.4 MCK_INTERRUPT Flow 8–14
8.5 System-Correctable Error Interrupt Flow (IPL 20) 8–14

9 Electrical Data

9.1 Electrical Characteristics 9–1
9.2 dc Characteristics 9–2
9.2.1 Power Supply 9–2
9.2.2 Input Signal Pins 9–2
9.2.3 Output Signal Pins 9–2
9.3 Clocking Scheme 9–4
9.3.1 Input Clocks 9–4

9.3.2 Clock Termination and Impedance Levels 9–6
9.3.3 ac Coupling 9–6
9.4 ac Characteristics 9–7
9.4.1 Test Configuration 9–7
9.4.2 Pin Timing 9–8
9.4.2.1 Backup Cache Loop Timing 9–8
9.4.2.2 sys_clk-Based Systems 9–10
9.4.2.3 Reference Clock-Based Systems 9–13
9.4.3 Digital Phase-Locked Loop 9–14
9.4.4 Timing—Additional Signals 9–16
9.4.5 Timing of Test Features 9–19
9.4.6 Icache BiSt Operation Timing 9–19
9.4.7 Automatic SROM Load Timing 9–21
9.4.8 Clock Test Modes 9–23
9.4.8.1 Normal Mode 9–23
9.4.8.2 Chip Test Mode 9–23
9.4.8.3 Module Test Mode 9–23
9.4.8.4 Clock Test Reset Mode 9–23
9.4.9 IEEE 1149.1 (JTAG) Performance 9–24
9.5 Power Supply Considerations 9–24
9.5.1 Decoupling 9–24
9.5.2 Power Supply Sequencing 9–25

10 Thermal Management

10.1 Operating Temperature 10–1
10.2 Heat Sink Specifications 10–3
10.3 Thermal Design Considerations 10–4

11 Mechanical Data and Packaging Information

11.1 Mechanical Specifications 11–1
11.2 Signal Descriptions and Pin Assignment 11–3
11.2.1 Signal Pin Lists 11–3
11.2.2 Pin Assignment 11–8

12 Testability and Diagnostics

12.1 Test Port Pins 12–1
12.2 Test Interface 12–2
12.2.1 IEEE 1149.1 Test Access Port 12–2
12.2.2 Test Status Pins 12–6
12.3 Boundary Scan Register 12–7

A Alpha Instruction Set

A.1 Alpha Instruction Summary A–1
A.1.1 Opcodes Reserved for Digital A–6
A.1.2 Opcodes Reserved for PALcode A–7
A.2 IEEE Floating-Point Instructions A–7
A.3 VAX Floating-Point Instructions A–9
A.4 Opcode Summary A–10
A.5 Required PALcode Function Codes A–12
A.6 Alpha 21164 Microprocessor IEEE Floating-Point Conformance A–12

B Alpha 21164 Microprocessor Specifications

C Serial Icache Load Predecode Values

D Errata Sheet

E Technical Support and Ordering Information

E.1 Technical Support E–1
E.2 Ordering Digital Semiconductor Products E–1
E.3 Ordering Digital Semiconductor Sample Kits E–1
E.4 Ordering Associated Literature E–2
E.5 Ordering Associated Third-Party Literature E–3

Glossary

Index

Figures

2–1	Alpha 21164 Microprocessor Block/Pipe Flow Diagram	2–3
2–2	Instruction Pipeline Stages	2–15
2–3	Floating-Point Control Register (FPCR) Format	2–39
2–4	Typical Uniprocessor Configuration	2–41
2–5	Typical Multiprocessor Configuration	2–42
2–6	Cacheless Multiprocessor Configuration	2–43
3–1	Alpha 21164 Microprocessor Logic Symbol	3–2
4–1	Alpha 21164 System/Bcache Interface	4–3
4–2	Clock Signals and Functions	4–6
4–3	Alpha 21164 Uniprocessor Clock	4–7
4–4	Alpha 21164 Reference Clock for Multiprocessor Systems	4–9
4–5	**ref_clk_in_h** Initially Sampled Low	4–10
4–6	**ref_clk_in_h** Initially Sampled High	4–11
4–7	Full Scache Duplicate Tag Store	4–16
4–8	Duplicate Tag Store Algorithm	4–17
4–9	Partial Scache Duplicate Tag Store	4–18
4–10	Cache Subset Hierarchy	4–20
4–11	Write Invalidate Protocol: 21164 State Transitions	4–24
4–12	Write Invalidate Protocol: System/Bus State Transitions	4–25
4–13	Flush-Based Protocol 21164 States	4–28
4–14	Flush-Based Protocol System/Bus States	4–28
4–15	Bcache Read Transaction	4–32
4–16	Wave Pipeline Timing Diagram	4–33
4–17	Bcache Write Transaction	4–34
4–18	READ MISS—No Bcache Timing Diagram	4–40
4–19	READ MISS MOD—Bcache Timing Diagram	4–42
4–20	READ MISS with Victim (Victim Buffer) Timing Diagram	4–45
4–21	READ MISS with Victim (without Victim Buffer) Timing Diagram	4–47
4–22	WRITE BLOCK Timing Diagram	4–49
4–23	SET DIRTY and LOCK Timing Diagram	4–51
4–24	Algorithm for System Sending Commands to the 21164	4–54
4–25	READ DIRTY Timing Diagram (Scache Hit)	4–59

4–26	INVALIDATE Timing Diagram—Bcache Hit	4–61
4–27	SET SHARED Timing Diagram	4–63
4–28	FLUSH Timing Diagram (Scache Hit)	4–67
4–29	READ Timing Diagram (Scache Hit)	4–69
4–30	Driving the Command/Address Bus	4–70
4–31	Example of Using **idle_bc_h** and **fill_h**	4–73
4–32	Using **data_bus_req_h**	4–74
4–33	READ MISS Completed First—Victim Buffer	4–76
4–34	READ MISS Second—No Victim Buffer	4–77
4–35	System Command to FILL Example 1	4–78
4–36	System Command to FILL Example 2	4–79
4–37	FILL to Private Read or Write Operation	4–80
4–38	READ MISS with Victim Example	4–85
4–39	idle_bc_h and cack_h Race Example	4–87
4–40	READ MISS with **idle_bc_h** Asserted Example	4–88
4–41	READ MISS with Victim Abort Example	4–90
4–42	Bcache Hit Under READ MISS Example	4–91
4–43	ECC Code	4–93
4–44	Alpha 21164 Interrupt Signals	4–96
5–1	Istream Translation Buffer Tag Register (ITB_TAG)	5–5
5–2	Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Write Format	5–6
5–3	Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Read Format	5–7
5–4	Instruction Translation Buffer Address Space Number (ITB_ASN) Register	5–8
5–5	Instruction Translation Buffer IS (ITB_IS) Register	5–10
5–6	Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register (NT_Mode=0)	5–11
5–7	Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register (NT_Mode=1)	5–11
5–8	Virtual Page Table Base Register (IVPTBR) (NT_Mode=0)	5–12
5–9	Virtual Page Table Base Register (IVPTBR) (NT_Mode=1)	5–12
5–10	Icache Parity Error Status (ICPERR_STAT) Register	5–13
5–11	Exception Address (EXC_ADDR) Register	5–14
5–12	Exception Summary (EXC_SUM) Register	5–15
5–13	Exception Mask (EXC_MASK) Register	5–17
5–14	PAL Base Address (PAL_BASE) Register	5–18

5–15	Ibox Current Mode (ICM) Register	5–19
5–16	Ibox Control and Status Register (ICSR)	5–20
5–17	Interrupt Priority Level Register (IPLR)	5–23
5–18	Interrupt ID (INTID) Register	5–24
5–19	Asynchronous System Trap Request Register (ASTRR)	5–25
5–20	Asynchronous System Trap Enable Register (ASTER)	5–26
5–21	Software Interrupt Request Register (SIRR)	5–27
5–22	Hardware Interrupt Clear (HWINT_CLR) Register	5–28
5–23	Interrupt Summary Register (ISR)	5–29
5–24	Serial Line Transmit (SL_XMIT) Register	5–31
5–25	Serial Line Receive (SL_RCV) Register	5–32
5–26	Performance Counter (PMCTR) Register	5–33
5–27	Dstream Translation Buffer Address Space Number (DTB_ASN) Register	5–38
5–28	Dstream Translation Buffer Current Mode (DTB_CM) Register	5–39
5–29	Dstream Translation Buffer Tag (DTB_TAG) Register	5–40
5–30	Dstream Translation Buffer Page Table Entry (DTB_PTE) Register—Write Format	5–42
5–31	Dstream Translation Buffer Page Table Entry Temporary (DTB_PTE_TEMP) Register	5–43
5–32	Dstream Memory Management Fault Status (MM_STAT) Register	5–44
5–33	Faulting Virtual Address (VA) Register	5–46
5–34	Formatted Virtual Address (VA_FORM) Register (NT_Mode=1)	5–47
5–35	Formatted Virtual Address (VA_FORM) Register (NT_Mode=0)	5–47
5–36	Mbox Virtual Page Table Base Register (MVPTBR)	5–49
5–37	Dcache Parity Error Status (DC_PERR_STAT) Register	5–50
5–38	Dstream Translation Buffer Invalidate Single (DTB_IS) Register	5–53
5–39	Mbox Control Register (MCSR)	5–54
5–40	Dcache Mode (DC_MODE) Register	5–56
5–41	Miss Address File Mode (MAF_MODE) Register	5–58
5–42	Alternate Mode (ALT_MODE) Register	5–60
5–43	Cycle Counter (CC) Register	5–61
5–44	Cycle Counter Control (CC_CTL) Register	5–62

5–45	Dcache Test Tag Control (DC_TEST_CTL) Register	5–63
5–46	Dcache Test Tag (DC_TEST_TAG) Register	5–64
5–47	Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register	5–66
5–48	Scache Control (SC_CTL) Register	5–69
5–49	Scache Status (SC_STAT) Register	5–72
5–50	Scache Address (SC_ADDR) Register	5–76
5–51	Bcache Control (BC_CONTROL) Register	5–78
5–52	Bcache Configuration (BC_CONFIG) Register	5–84
5–53	Bcache Tag Address (BC_TAG_ADDR) Register	5–89
5–54	External Interface Status (EI_STAT) Register	5–92
5–55	External Interface Address (EI_ADDR) Register	5–94
5–56	Fill Syndrome (FILL_SYN) Register	5–96
6–1	HW_LD Instruction Format	6–9
6–2	HW_ST Instruction Format	6–10
6–3	HW_REI Instruction Format	6–11
6–4	HW_MFPR and HW_MTPR Instruction Format	6–12
9–1	osc_clk_in_h,l Input Network and Terminations	9–5
9–2	Input/Output Pin Timing	9–7
9–3	Bcache Timing	9–10
9–4	sys_clk System Timing	9–12
9–5	ref_clk System Timing	9–15
9–6	BiSt Timing Event–Time Line	9–20
9–7	SROM Load Timing Event–Time Line	9–21
9–8	Serial ROM Load Timing	9–22
10–1	Type 1 Heat Sink	10–3
10–2	Type 2 Heat Sink	10–4
11–1	Package Dimensions	11–2
11–2	Alpha 21164 Top View (Pin Down)	11–8
11–3	Alpha 21164 Bottom View (Pin Up)	11–9
12–1	IEEE 1149.1 Test Access Port	12–4
12–2	TAP Controller State Machine	12–5

Tables

1	Register Field Type Notation	xxvi
2	Register Field Notation	xxvii
2–1	Effect of Branching Instructions on the Branch-Prediction Stack	2–7
2–2	Pipeline Examples—All Cases	2–16
2–3	Pipeline Examples—Integer Add	2–16
2–4	Pipeline Examples—Floating Add	2–16
2–5	Pipeline Examples—Load (Dcache Hit)	2–17
2–6	Pipeline Examples—Load (Dcache Miss)	2–17
2–7	Pipeline Examples—Store (Dcache Hit)	2–18
2–8	Instruction Classes and Slotting	2–20
2–9	Instruction Latencies	2–25
2–10	Floating-Point Control Register Bit Descriptions	2–39
3–1	Alpha 21164 Signal Descriptions	3–3
3–2	Alpha 21164 Signal Descriptions by Function	3–13
4–1	CPU Clock Generation Control	4–5
4–2	System Clock Divisor	4–7
4–3	System Clock Delay	4–8
4–4	Physical Memory Regions	4–13
4–5	Components for 21164 Write Invalidate Systems	4–22
4–6	Bcache States for Cache Coherency Protocols	4–23
4–7	Components for 21164 Flush Cache Protocol Systems	4–26
4–8	Bcache Options	4–35
4–9	Alpha 21164-Initiated Interface Commands	4–37
4–10	System-Initiated Interface Commands (Write Invalidate Protocol)	4–55
4–11	Alpha 21164 Responses on **addr_res_h<1:0>** to Write Invalidate Protocol Commands	4–57
4–12	Alpha 21164 Responses on **addr_res_h<2>** to 21164 Commands	4–57
4–13	Alpha 21164 Minimum Response Time to Write Invalidate Protocol Commands	4–58
4–14	System-Initiated Interface Commands (Flush Protocol)	4–64
4–15	Alpha 21164 Responses to Flush-Based Protocol Commands	4–65

4–16	Alpha 21164 Responses on **addr_res_h<2>** to 21164 Commands	4–65
4–17	Minimum 21164 Response Time to Flush Protocol Commands	4–66
4–18	Data Check Bit Correspondence to CB*n*	4–93
4–19	Interrupt Priority Level Effect	4–97
5–1	Ibox, Mbox, Dcache, and PALtemp IPR Encodings	5–2
5–2	Granularity Hint Bits in ITB_PTE_TEMP Read Format . . .	5–9
5–3	Icache Parity Error Status Register Fields	5–13
5–4	Exception Summary Register Fields	5–15
5–5	Ibox Control and Status Register Fields	5–21
5–6	Software Interrupt Request Register Fields	5–27
5–7	Hardware Interrupt Clear Register Fields	5–28
5–8	Interrupt Summary Register Fields	5–30
5–9	Serial Line Transmit Register Fields	5–31
5–10	Serial Line Receive Register Fields	5–32
5–11	Performance Counter Register Fields	5–34
5–12	PMCTR Counter Select Options	5–35
5–13	Measurement Mode Control	5–37
5–14	Dstream Memory Management Fault Status Register Fields	5–44
5–15	Formatted Virtual Address Register Fields	5–48
5–16	Dcache Parity Error Status Register Fields	5–51
5–17	Mbox Control Register Fields	5–55
5–18	Dcache Mode Register Fields	5–57
5–19	Miss Address File Mode Register Fields	5–59
5–20	Alternate Mode Register Settings	5–60
5–21	Cycle Counter Control Register Fields	5–62
5–22	Dcache Test Tag Control Register Fields	5–63
5–23	Dcache Test Tag Register Fields	5–65
5–24	Dcache Test Tag Temporary Register Fields	5–67
5–25	Cbox Internal Processor Register Descriptions	5–68
5–26	Scache Control Register Fields	5–70
5–27	Scache Status Register Fields	5–73
5–28	SC_CMD Field Descriptions	5–74
5–29	Scache Address Register Fields	5–77
5–30	Bcache Control Register Fields	5–79

5–31 PM_MUX_SEL Register Fields 5–83
5–32 Bcache Configuration Register Fields 5–85
5–33 Bcache Tag Address Register Fields 5–90
5–34 Loading and Locking Rules for External Interface Registers 5–92
5–35 EI_STAT Register Fields 5–93
5–36 Syndromes for Single-Bit Errors 5–96
5–37 Cbox IPR PALcode Restrictions 5–100
5–38 PALcode Restrictions Table 5–101
6–1 PALcode Trap Entry Points 6–6
6–2 Required PALcode Function Codes 6–7
6–3 Opcodes Reserved for PALcode 6–8
6–4 HW_LD Format Description 6–9
6–5 HW_ST Format Description 6–10
6–6 HW_REI Format Description 6–11
6–7 HW_MTPR and HW_MFPR Format Description 6–12
7–1 Alpha 21164 Signal Pin Reset State 7–3
7–2 Internal Processor Register Reset State 7–10
9–1 Alpha 21164 Absolute Maximum Ratings 9–1
9–2 CMOS dc Input/Output Characteristics 9–3
9–3 Input Clock Specification 9–6
9–4 Bcache Loop Timing 9–9
9–5 Output Driver Characteristics 9–9
9–6 Alpha 21164 System Clock Output Timing (sysclk=$T_{\o}$) 9–11
9–7 Alpha 21164 Reference Clock Input Timing 9–13
9–8 ref_clk System Timing Stages 9–15
9–9 Input Timing for sys_clk_out- or ref_clk_in-Based Systems 9–16
9–10 Output Timing for sys_clk_out- or ref_clk_in-Based Systems 9–17
9–11 Bcache Control Signal Timing 9–19
9–12 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (System Cycles) 9–20
9–13 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (CPU Cycles) 9–21
9–14 SROM Load Timing for Some System Clock Ratios (System Cycles) 9–22

9–15	SROM Load Timing for Some System Clock Ratios (CPU Cycles)	9–22
9–16	Test Modes	9–23
9–17	IEEE 1149.1 Circuit Performance Specifications	9–24
10–1	$\theta_{c}a$ at Various Airflows	10–2
10–2	Maximum T_{a} at Various Airflows	10–2
11–1	Alphabetic Signal Pin List	11–3
12–1	Alpha 21164 Test Port Pins	12–1
12–2	Compliance Enable Inputs	12–2
12–3	Instruction Register	12–6
12–4	Boundary Scan Register Organization	12–8
A–1	Instruction Format and Opcode Notation	A–1
A–2	Architecture Instructions	A–2
A–3	Opcodes Reserved for Digital	A–6
A–4	Opcodes Reserved for PALcode	A–7
A–5	IEEE Floating-Point Instruction Function Codes	A–7
A–6	VAX Floating-Point Instruction Function Codes	A–9
A–7	Opcode Summary	A–11
A–8	Required PALcode Function Codes	A–12
B–1	Alpha 21164 Microprocessor Specifications	B–2
D–1	Document Revision History	D–1

Preface

Audience

This reference manual is for system designers and programmers who use the Alpha 21164 microprocessor.

Content

This reference manual contains the following chapters and appendixes:

- Chapter 1 introduces the 21164 and provides an overview of the Alpha architecture.
- Chapter 2 describes the major hardware functions and the internal chip architecture. It describes performance measurement facilities, coding rules, and design examples.
- Chapter 3 lists and describes the external hardware interface signals.
- Chapter 4 describes the external bus functions and transactions, lists bus commands, and describes the clock functions.
- Chapter 5 lists and describes the 21164 internal processor register set.
- Chapter 6 describes the privileged architecture library code (PALcode).
- Chapter 7 describes the initialization and configuration sequence.
- Chapter 8 describes error detection and error handling.
- Chapter 9 provides electrical data and describes signal integrity issues.
- Chapter 10 provides information about thermal management.
- Chapter 11 provides mechanical data and packaging information, including signal pin lists.
- Chapter 12 describes chip and system testability features.
- Appendix A summarizes the Alpha instruction set.

- Appendix B summarizes the 21164 specifications.
- Appendix C provides a C code example that calculates the predecode values of a serial Icache load.
- Appendix D lists changes and revisions to this manual.
- Appendix E provides phone numbers for support and lists related Digital and third-party publications with order information.
- The Glossary lists and defines terms associated with the 21164.

The companion volume to this manual, the *Alpha Architecture Reference Manual*, contains the Alpha architecture information.

Terminology and Conventions

The following sections describe the terminology and conventions used in this manual.

Numbering

All numbers are decimal unless otherwise indicated. Where there is ambiguity, numbers other than decimal are indicated with the name of the base following the number in parentheses, for example FF (hex).

Security Holes

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, either overtly or covertly, without authorization from the operating system.

UNPREDICTABLE and UNDEFINED

Throughout this manual, the terms UNPREDICTABLE and UNDEFINED are used. Their meanings are quite different and must be carefully distinguished.

In particular, only privileged software (that is, software running in kernel mode) can trigger UNDEFINED operations. Unprivileged software cannot trigger UNDEFINED operations. However, either privileged or unprivileged software can trigger UNPREDICTABLE results or occurrences.

UNPREDICTABLE results or occurrences do not disrupt the basic operation of the processor. The processor continues to execute instructions in its normal manner. In contrast, UNDEFINED operations can halt the processor or cause it to lose information.

The terms UNPREDICTABLE and UNDEFINED can be further described as follows:

UNPREDICTABLE

- 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.
- An UNPREDICTABLE result 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.
- 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.

  Specifically, UNPREDICTABLE results must not depend upon, 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 to which the current process in the current access mode does not have access.
  - 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, on the contents of processor temporary registers left behind by some previously running process, or on a sequence of actions of different processes.

UNDEFINED

- 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 may halt the processor or cause it to lose information. However, UNDEFINED operations must not cause the processor to hang, that is, reach an unhalted state from which there is no transition to a normal state in which the machine executes instructions. Only privileged software (that is, software running in kernel mode) may trigger UNDEFINED operations.

Data Field Size

The term INT*nn*, where *nn* is one of 2, 4, 8, 16, 32, or 64, refers to a data field of *nn* contiguous NATURALLY ALIGNED bytes. For example, INT4 refers to a NATURALLY ALIGNED longword.

Ranges and Extents

Ranges are specified by a pair of numbers separated by three 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> specify an extent of bits including bits 7, 6, 5, 4, and 3.

ALIGNED and UNALIGNED

In this manual the terms ALIGNED and NATURALLY ALIGNED are used interchangeably to 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. Thus, an ALIGNED 64-byte stack frame has a memory address that is a multiple of 64.

If a datum of size 2**N is stored at a byte address that is not a multiple of 2**N, it is called UNALIGNED.

Register Format Notation

This manual contains illustrations that show the format of various registers. Some registers are followed by a description of each field. The fields on the register are labeled with either a name or a mnemonic. The description of each field includes the name or mnemonic, the bit extent, and the type.

The ‘‘Type’’ column in the field description includes both the actual type of the field, and an optional initialized value, separated from the type by a comma. The type denotes the functional operation of the field, and may be one of the values shown in Table 1. If present, the initialized value indicates that the field is initialized by hardware to the specified value at power-up. If the initialized value is not present, the field is not initialized at power-up.

Table 1 Register Field Type Notation

Notation	Description
RC	A read-to-clear field. The value is written by hardware and remains unchanged until read. The value may be read by software at which point, hardware may write a new value into the field.
RO	A read-only bit or field. The value may be read by software. It is written by hardware. Software write operations are ignored.
RW	A read/write bit or field. The value may be read and written by software.
W0C	A write-zero-to-clear bit. If read operations are allowed to the register, then the value may be read by software. If it is a write-only register, then a read operation by software returns an UNPREDICTABLE result. Software write operations of a 0 cause the bit to be cleared by hardware. Software write operations of a 1 do not modify the state of the bit.
W1C	A write-one-to-clear bit. If read operations are allowed to the register, then the value may be read by software. If it is a write-only register, then a read operation by software returns an UNPREDICTABLE result. Software write operations of a 1 cause the bit to be cleared by hardware. Software write operations of a 0 do not modify the state of the bit.
WA	A write-anything-to-the-register-to-clear bit. If read operations are allowed to the register, then the value may be read by software. If it is a write-only register, then a read operation by software returns an UNPREDICTABLE result. Software write operations of any value to the register cause the bit to be cleared by hardware.
WO	A write-only bit or field. The value may be written by software and is used by hardware. Read operations by software return an UNPREDICTABLE result.
WZ	A write bit or field. The value may be written by software and is used by hardware. Read operations by software return a 0.

In addition to named fields in registers, other bits of the register may be labeled with one of the five symbols listed in Table 2. These symbols denote the type of the unnamed fields in the register.

Table 2 Register Field Notation

Notation	Description
IGN	Register bits specified as ignore (IGN) are ignored when written and are UNPREDICTABLE when read if not otherwise specified.
MBZ	Register bits specified as MBZ (must be zero) must never be filled by software with a non-zero value. If the processor encounters a non-zero value in a field specified as MBZ, an UNDEFINED operation may result.
RAO	Register bits specified as RAO (read as one) return a one when read.
RAZ	Register bits specified as RAZ (read as zero) return a zero when read.
SBZ	Register bits specified as SBZ (should be zero) should be filled by software with a zero value. Non-zero values in SBZ fields produce UNDEFINED results and may produce extraneous instruction-issue delays.

1
Introduction

This chapter provides a brief introduction to the Alpha architecture, Digital's RISC (reduced instruction set computing) architecture designed for high performance. The chapter then summarizes the specific features of the Alpha 21164 (hereafter called the 21164), a microprocessor that implements the Alpha architecture. Appendix A provides a list of Alpha instructions.

For a complete definition of the Alpha architecture, refer to the companion volume, the *Alpha Architecture Reference Manual*.

1.1 The Architecture

The Alpha architecture is a 64-bit load and store RISC architecture designed with particular emphasis on speed, multiple instruction issue, multiple processors, and software migration from many operating systems.

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 load or store operations. All data manipulation is done between registers.

The Alpha architecture supports the following data types:

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

In the Alpha 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 instructions every CPU cycle.

1.1 The Architecture

The 21164 uses a set of subroutines, called privileged architecture library code (PALcode), that is specific to a particular Alpha 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 architecture performs byte shifting and masking with normal 64-bit, register-to-register instructions; it does not include single-byte load and store instructions.

1.1.1 Addressing

The basic addressable unit in the Alpha architecture is the 8-bit byte. The 21164 supports a 43-bit virtual address.

Virtual addresses as seen by the program are translated into physical memory addresses by the memory management mechanism. The 21164 supports a 40-bit physical address.

1.1.2 Integer Data Types

Alpha architecture supports four integer data types:

Data Type	Description
Byte	A byte is 8 contiguous bits that start at an addressable byte boundary. A byte is an 8-bit value. A byte is supported in Alpha architecture by the EXTRACT, MASK, INSERT, and ZAP instructions.
Word	A word is 2 contiguous bytes that start at an arbitrary byte boundary. A word is a 16-bit value. A word is supported in Alpha architecture by the EXTRACT, MASK, and INSERT instructions.
Longword	A longword is 4 contiguous bytes that start at an arbitrary byte boundary. A longword is a 32-bit value. A longword is supported in the Alpha architecture by sign-extended load and store instructions and by longword arithmetic instructions.
Quadword	A quadword is 8 contiguous bytes that start at an arbitrary byte boundary. A quadword is supported in Alpha architecture by load and store instructions and quadword integer operate instructions.

Note

Alpha implementations may impose a significant performance penalty when accessing operands that are not NATURALLY ALIGNED. Refer to the *Alpha Architecture Reference Manual* for details.

1.1.3 Floating-Point Data Types

The 21164 supports the following floating-point data types:

- Longword integer format in floating-point unit
- Quadword integer format in floating-point unit
- IEEE floating-point formats
  - S_floating
  - T_floating
- VAX floating-point formats
  - F_floating
  - G_floating
  - D_floating (limited support)

1.2 Alpha 21164 Microprocessor Features

The 21164 microprocessor is a superscalar pipelined processor manufactured using 0.5-micron CMOS technology. It is packaged in a 499-pin IPGA carrier and has removable application-specific heat sinks. A number of configuration options allow its use in a range of system designs ranging from extremely simple uniprocessor systems with minimum component count to high-performance multiprocessor systems with very high cache and memory bandwidth.

The 21164 can issue four Alpha instructions in a single cycle, thereby minimizing the average cycles per instruction (CPI). A number of low-latency and/or high-throughput features in the instruction issue unit and the onchip components of the memory subsystem further reduce the average CPI.

The 21164 and associated PALcode implements 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. Partial hardware implementation is provided for the architecturally optional FETCH and FETCH_M instructions.

Other 21164 features include:

- A peak instruction execution rate of four times the CPU clock frequency.
- The ability to issue up to four instructions during each clock cycle.
- An onchip, demand-paged memory-management unit with translation buffer, which, when used with PALcode, can implement a variety of page table structures and translation algorithms. The unit consists of a 64-entry data translation buffer (DTB) and a 48-entry instruction translation buffer (ITB), with each entry able to map a single 8K-byte page or a group of 8, 64, or 512 8K-byte pages. The size of each translation buffer entry's group is specified by hint bits stored in the entry. The DTB and ITB implement 7-bit address space numbers (ASN), (MAX_ASN=127).
- Two onchip, high-throughput pipelined floating-point units, capable of executing both Digital and IEEE floating-point data types.
- An onchip, 8K-byte virtual instruction cache with 7-bit ASNs (MAX_ ASN=127).
- An onchip, dual-read-ported, 8K-byte data cache.
- An onchip write buffer with six 32-byte entries.
- An onchip, 96K-byte, 3-way, set-associative, write-back, second-level mixed instruction and data cache.
- A 128-bit data bus with onchip parity and error correction code (ECC) support.
- Support for an optional external third-level cache. The size and access time of the external third-level cache is programmable.
- An internal clock generator providing a high-speed clock used by the 21164, and a pair of programmable system clocks for use by the CPU module.
- Onchip performance counters to measure and analyze CPU and system performance.
- Chip and module level test support, including an instruction cache test interface to support chip and module level testing.
- A 3.3-V power supply. (Direct connection to 5-V logic supported.)

Refer to Chapter 9 for 21164 dc and ac electrical characteristics. Refer to the *Alpha Architecture Reference Manual* for a description of address space numbers (ASNs).

2
Internal Architecture

This chapter provides both an overview of the 21164 microarchitecture and a system designer's view of the 21164 implementation of the Alpha architecture. The combination of the 21164 microarchitecture and privileged architecture library code (PALcode) defines the chip's implementation of the Alpha architecture. If a certain piece of hardware seems to be ''architecturally incomplete,'' the missing functionality is implemented in PALcode. Chapter 6 provides more information on PALcode.

This chapter describes the major functional hardware units and is not intended to be a detailed hardware description of the chip. It is organized as follows:

- 21164 microarchitecture
- Pipeline organization
- Scheduling and issuing rules
- Replay traps
- Miss address file (MAF) and load-merging rules
- Mbox store execution
- Write buffer and the WMB instruction
- Performance measurement support
- Floating-point control register
- Design examples

2.1 Alpha 21164 Microarchitecture

The 21164 microprocessor is a high-performance implementation of Digital's Alpha architecture. Figure 2–1 is a block diagram of the 21164 showing the major functional blocks relative to pipeline stage flow. Please see the end of this book for an enlarged foldout version of this figure. The following paragraphs provide an overview of the chip's architecture and major functional units.

The 21164 microprocessor consists of the following internal sections:

- Clock generation logic (Section 4.2)
- Instruction fetch/decode unit and branch unit (Ibox) (Section 2.1.1), which includes:

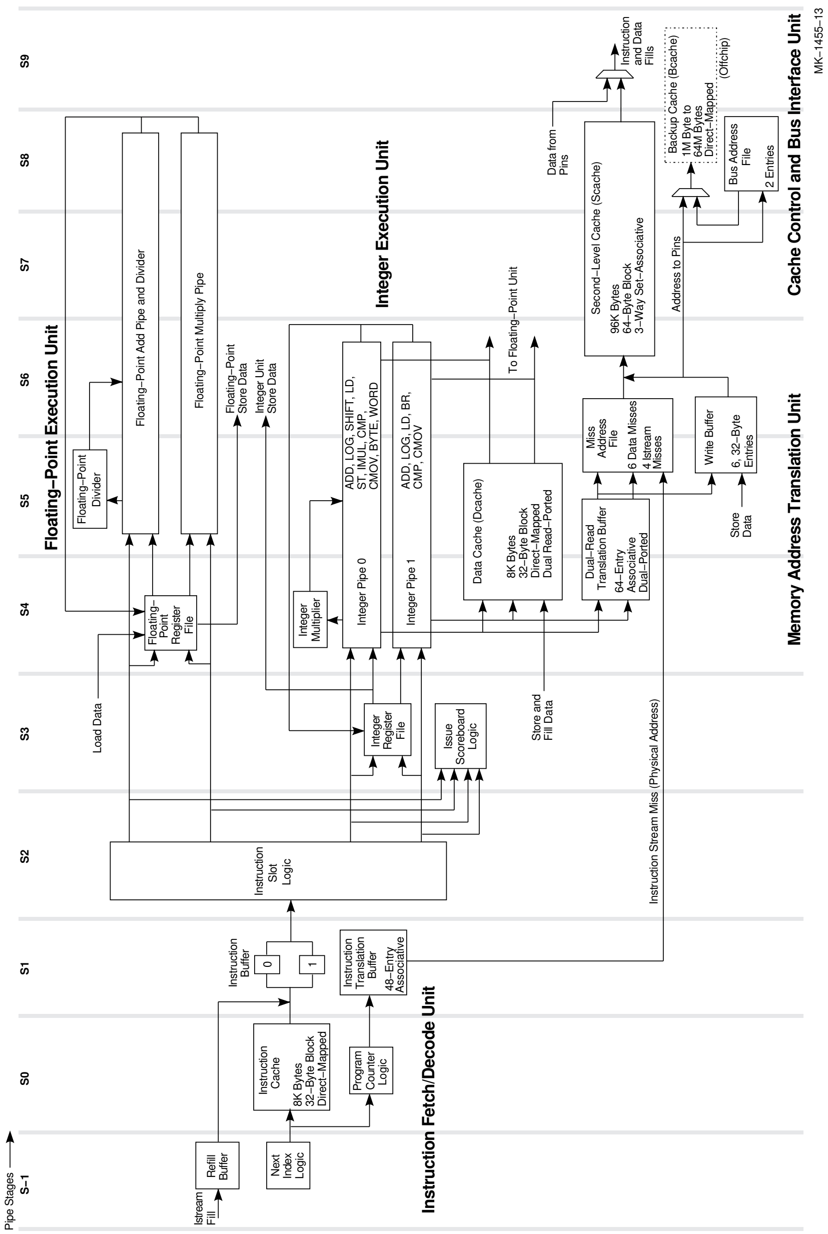
  Instruction prefetcher and instruction decoder
  Instruction translation buffer
  Branch prediction
  Instruction slotting/issue
  Interrupt support

- Integer execution unit (Ebox) (Section 2.1.2)
- Floating-point execution unit (Fbox) (Section 2.1.3)
- Memory address translation unit (Mbox) (Section 2.1.4), which includes:

  Data translation buffer (DTB)
  Miss address file (MAF)
  Write buffer
  Dcache control

- Cache control and bus interface unit (Cbox) with interface to external cache (Section 2.1.5)
- Data cache (Dcache) (Section 2.1.6.1)
- Instruction cache (Icache) (Section 2.1.6.2)
- Second-level cache (Scache) (Section 2.1.6.3)
- Serial read-only memory (SROM) interface (Section 2.1.7)

Figure 2–1 Alpha 21164 Microprocessor Block/Pipe Flow Diagram



2.1 Alpha 21164 Microarchitecture

2.1.1 Instruction Fetch/Decode Unit and Branch Unit

The primary function of the instruction fetch/decode unit and branch unit (Ibox) is to manage and issue instructions to the Ebox, Mbox, and Fbox. It also manages the instruction cache. The Ibox contains:

- Prefetcher and instruction buffer
- Instruction slot and issue logic
- Program counter (PC) and branch prediction logic
- 48-entry instruction translation buffers (ITBs)
- Abort logic
- Register conflict logic
- Interrupt and exception logic

2.1.1.1 Instruction Decode and Issue

The Ibox decodes up to four instructions in parallel and checks that the required resources are available for each instruction. The Ibox issues only the instructions for which all required resources are available. The Ibox does not issue instructions out of order, even if the resources are available for a later instruction and not for an earlier one.

In other words:

- If resources are available, and multiple issue is possible, then all four instructions are issued.
- If resources are available only for a later instruction and not for an earlier one, then only the instructions up to the latest one for which resources are available are issued.

The Ibox handles only NATURALLY ALIGNED groups of four instructions (INT16). The Ibox does not advance to a new group of four instructions until all instructions in a group are issued. If a branch to the middle of an INT16 group occurs, then the Ibox attempts to issue the instructions from the branch target to the end of the current INT16, then it proceeds to the next INT16 of instructions after all the instructions in the target INT16 are issued. Thus, achieving maximum issue rate and optimal performance requires that code be be scheduled properly and that floating or integer NOP instructions be used to fill empty slots in the scheduled instruction stream.

For more information on instruction scheduling and issuing, including detailed rules governing multiple instruction issue, refer to Section 2.3.

2.1.1.2 Instruction Prefetch

The Ibox contains an instruction prefetcher and a 4-entry, 32-byte-per-entry, prefetch buffer called the refill buffer. Each instruction cache (Icache) miss is checked in the refill buffer. If the refill buffer contains the instruction data, it fills the Icache and instruction buffer simultaneously. If the refill buffer does not contain the necessary data, a fetch and a number of prefetches are sent to the Mbox. One prefetch is sent per cycle until each of the four entries in the refill buffer is filled or has a pending fill. If these requests are all Scache hits, it is possible for instruction data to stream into the Ibox at the rate of one INT16 (four instructions) per cycle. The Ibox can sustain up to quad-instruction issue from this Scache fill stream, filling the Icache simultaneously. The refill buffer holds all returned fill data until the data is required by the Ibox pipeline.

When there is a hit in the refill buffer, the 21164 waits until there is a ‘‘true’’ miss. A ‘‘true’’ miss is one that misses in the Icache and then in the refill buffer. If an Icache miss results in a refill buffer hit, prefetching is not started until all the data has been moved from the refill buffer entry into the pipeline.

Each fill of the Icache by the refill buffer occurs when the instruction buffer stage in the Ibox pipeline requires a new INT16. The INT16 is written into the Icache and the instruction buffer simultaneously. This can occur at a maximum rate of one Icache fill per cycle. The actual rate depends on how frequently the instruction buffer stage requires a new INT16, and on availability of data in the refill buffer.

Once an Icache miss occurs, the Icache enters fill mode. When the Icache is in fill mode, the refill buffer is checked each cycle to see if it contains the next INT16 required by the instruction buffer.

When the required data is not available in the refill buffer (also a miss), the Icache is checked for a hit while it awaits the arrival of the data from the Scache or beyond. The Ibox sends a read request to the Cbox by means of the Mbox. The Cbox checks the Scache and Bcache, and if the request misses in all caches, the Cbox drives a main memory request.

If there is an Icache hit at this time, the Icache returns to access mode and the prefetcher stops sending fetches to the Mbox. When a new program counter (PC) is loaded (that is, taken branches), the Icache returns to access mode until the first miss. The refill buffer receives and holds instruction data from fetches initiated before the Icache returned to access mode.

The Icache has a 32-byte block size, whereas the refill buffer is able to load the Icache with only one INT16 (16 bytes) per cycle. Therefore, each Icache block has two valid bits, one for each 16-byte subblock.

2.1.1.3 Branch Execution

When a branch or jump instruction is fetched from the Icache by the prefetcher, the Ibox needs one cycle to calculate the target PC before it is ready to fetch the target instruction stream. In the second cycle after the fetch, the Icache is accessed at the target address. Branch and PC prediction are necessary to predict and begin fetching the target instruction stream before the branch or jump instruction is issued.

The Icache records the outcome of branch instructions in a 2-bit history state provided for each instruction location in the Icache. This information is used as the prediction for the next execution of the branch instruction. The 2-bit history state is a saturating counter that increments on taken branches and decrements on not-taken branches. The branch is predicted taken on the top two count values and is predicted not-taken on the bottom two count values. The history status is not initialized on Icache fill, therefore it may ''remember'' a branch that was evicted from the Icache and subsequently reloaded.

The 21164 does not limit the number of branch predictions outstanding to one. It predicts branches even while waiting to confirm the prediction of previously predicted branches. There can be one branch prediction pending for each of pipeline stages 3 and 4, plus up to four in pipeline stage 2. Refer to Section 2.2 for a description of pipeline stages.

When a predicted branch is issued, the Ebox or Fbox checks the prediction. The branch history table is updated accordingly. On branch mispredict, a mispredict trap occurs and the Ibox restarts execution from the correct PC.

The 21164 provides a 12-entry subroutine return stack that is controlled by decoding the opcode (BSR, HW_REI and JMP/JSR/RET/JSR_COROUTINE), and DISP<15:14> in JMP/JSR/RET/JSR_COROUTINE. The stack stores an Icache index in each entry. The stack is implemented as a circular queue that wraps around in the overflow and underflow cases. Table 2–1 lists the effect each of these instructions has on the state of the branch-prediction stack.

Table 2–1 Effect of Branching Instructions on the Branch-Prediction Stack

Intruction	Stack Used for Prediction?	Effect on Stack
BSR, JSR	No	Push PC+4
RET	Yes	Pop
JMP, BR, BR*xx*	No	No effect
JSR_COROUTINE	Yes	Pop, then push PC+4
PAL entry	No	Push PC+4
HW_REI	Yes	Pop

The 21164 uses the Icache index hint in the JMP and JSR instructions to predict the target PC. The Icache index hint in the instruction's displacement field is used to access the direct-mapped Icache. The upper bits of the PC are formed from the data in the Icache tag store at that index. Later in the pipeline, the PC prediction is checked against the actual PC generated by the Ebox. A mismatch causes a PC mispredict trap and restart from the correct PC. This is similar to branch prediction.

The RET, JSR_COROUTINE, and HW_REI instructions predict the next PC by using the index from the subroutine return stack. The upper bits of the PC are formed from the data in the Icache tag at that index. These predictions are checked against the actual PC in exactly the same way that JMP and JSR predictions are checked.

Changes from PALmode to native mode and vice versa are predicted on all PC predictions that use the subroutine return stack. In all cases, if the PC prediction is correct, the mode prediction will also be correct. Instruction stream (Istream) prefetching is disabled when a PC prediction is outstanding.

2.1.1.4 Instruction Translation Buffer

The Ibox includes a 48-entry, fully associative instruction translation buffer (ITB). The buffer stores recently used Istream address translations and protection information for pages ranging from 8K bytes to 4M bytes and uses a not-last-used replacement algorithm.

PALcode fills and maintains the ITB. Each entry supports all four granularity hint bit combinations, so that any single ITB entry can provide translation for up to 512 contiguously mapped 8K-byte pages. The operating system, using PALcode, must ensure that virtual addresses can only be mapped through a single ITB entry or superpage mapping at one time. Multiple simultaneous mapping can cause UNDEFINED results.

While not executing in PALmode, the 43-bit virtual PC is routed to the ITB each cycle. If the page table entry (PTE) associated with the PC is cached in the ITB, the protection bits for the page that contains the PC are used by the Ibox to do the necessary access checks. If there is an Icache miss and the PC is cached in the ITB, the page frame number (PFN) and protection bits for the page that contains the PC are used by the Ibox to do the address translation and access checks.

The 21164's ITB supports 128 address space numbers (ASNs) (MAX_ASN=127) by means of a 7-bit ASN field in each ITB entry. PALcode uses the hardware-specific HW_MTPR instruction to write to the architecturally defined ITB_IAP register. This has the effect of invalidating ITB entries that do not have their ASM bit set.

The 21164 provides two optional translation extensions called superpages. Access to superpages is enabled using ICSR<SPE> and is allowed only while executing in privileged mode.

- One superpage maps virtual address bits <39:13> to physical address bits <39:13>, on a one-to-one basis, when virtual address bits <42:41> equal 2. This maps the entire physical address space four times over to the quadrant of the virtual address space.
- The other superpage maps virtual address bits <29:13> to physical address bits <29:13>, on a one-to-one basis, and forces physical address bits <39:30> to 0 when virtual address bits <42:30> equal $1FFE_{16}$. This effectively maps a 30-bit region of physical address space to a single region of the virtual address space defined by virtual address bits <42:30> = $1FFE_{16}$.

Access to either superpage mapping is allowed only while executing in kernel mode. Superpage mapping allows the operating system to map all physical memory to a privileged virtual memory region.

2.1.1.5 Interrupts

The Ibox exception logic supports three sources of interrupts:

- Hardware interrupts

  There are seven level-sensitive hardware interrupt sources supplied by the following signals:

  irq_h<3:0>
  mch_hlt_irq_h
  pwr_fail_irq_h
  sys_mch_chk_irq_h

- Software interrupts

There are 15 prioritized software interrupts sourced by the software interrupt request register (SIRR) (see Section 5.1.22).

- Asynchronous system traps (ASTs)

  There are four ASTs sourced by the asynchronous system trap request (ASTRR) register.

The serial interrupt, the internally detected correctable error interrupt, the performance counter interrupts, and **irq_h<3:0>** are all maskable by bits in the ICSR (see Section 5.1.17). The four AST traps are maskable by bits in the ASTER (see Section 5.1.21). In addition, the AST traps are qualified by the current processor mode. All interrupts are disabled when the processor is executing PALcode.

Each interrupt source, or group of sources, is assigned an interrupt priority level (IPL), as shown in Table 4–19. The current IPL is set using the IPLR register (see Section 5.1.18). Any interrupts that have an equal or lower IPL are masked. When an interrupt occurs that has an IPL greater than the value in the IPLR register, program control passes to the INTERRUPT PALcode entry point. PALcode processes the interrupt by reading the ISR (see Section 5.1.24) and the INTID register (see Section 5.1.19).

2.1.2 Integer Execution Unit

The integer execution unit (Ebox) contains two 64-bit integer execution pipelines, E0 and E1, which include the following:

- Two adders
- Two logic boxes
- A barrel shifter
- Byte-manipulation logic
- An integer multiplier

The Ebox also includes the 40-entry, 64-bit integer register file (IRF) that contains the 32 integer registers defined by the Alpha architecture and 8 PAL shadow registers. The register file has four read ports and two write ports that provide operands to both integer execution pipelines and accept results from both pipes. The register file also accepts load instruction results (memory data) on the same two write ports.

2.1.3 Floating-Point Execution Unit

The onchip, pipelined floating-point unit (FPU) can execute both IEEE and VAX floating-point instructions. The 21164 supports IEEE S_floating and T_floating data types, and all rounding modes. It also supports VAX F_floating and G_floating data types, and provides limited support for the D_floating format. The FPU contains:

- A 32-entry, 64-bit floating-point register file.
- A user-accessible control register.
- A floating-point multiply pipeline.
- A floating-point add pipeline—The floating-point divide unit is associated with the floating-point add pipeline but is not pipelined.

The FPU can accept two instructions every cycle, with the exception of floating-point divide instructions. The result latency for nondivide, floating-point instructions is four cycles.

The floating-point register file (FRF) has five read ports and four write ports. Four of the read ports are used by the two pipelines to source operands. The remaining read port is used by floating-point stores. Two of the write ports are used to write results from the two pipelines. The other two write ports are used to write fills from floating-point loads.

2.1.4 Memory Address Translation Unit

The memory address translation unit (Mbox) contains three major sections:

- Data translation buffer (dual ported)
- Miss address file
- Write buffer address file

There are a pair of write ports on the floating-point register file devoted to loads and fills for previous loads that missed. The Mbox arbitrates between floating-point loads that hit in the Dcache and floating-point fills from the Cbox, making certain that only one register is written per fill port in each cycle. Floating-point loads that conflict with Cbox fills for use of these write ports are forced to miss in the Dcache so that the Cbox fill can execute.

The Mbox receives up to two virtual addresses every cycle from the Ebox. The translation buffer generates the corresponding physical addresses and access control information for each virtual address. The 21164 implements a 43-bit virtual address and a 40-bit physical address.

2.1.4.1 Data Translation Buffer

The 64-entry, fully associative, dual-read-ported data translation buffer (DTB) stores recently used data stream (Dstream) page table entries (PTEs). Each entry supports all four granularity hint-bit combinations, so that a single DTB entry can provide translation for up to 512 contiguously mapped, 8K-byte pages. The translation buffer uses a not-last-used replacement algorithm.

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

The DTB also supports the optional superpage extensions that are enabled using ICSR<SPE>. The DTB superpage maps provide virtual-to-physical address translation for two regions of the virtual address space, as described in Section 2.1.1.4.

PALcode fills and maintains the DTB. The operating system, using PALcode, must ensure that virtual addresses be mapped either through a single DTB entry or through superpage mapping. Multiple simultaneous mapping can cause UNDEFINED results. The only exception to this rule is that any given virtual page may be mapped twice with identical data in two different DTB entries. This occurs in operating systems, such as OpenVMS, which utilize virtually accessible page tables. If the level 1 page table is accessed virtually, PALcode loads the translation information twice; once in the double-miss handler, and once in the primary handler. The PTE mapping the level 1 page table must remain constant during accesses to this page to meet this requirement.

2.1.4.2 Load Instruction and the Miss Address File

The Mbox begins the execution of each load instruction by translating the virtual address and by accessing the data cache (Dcache). Translation and Dcache tag read operations occur in parallel. If the addressed location is found in the Dcache (a hit), then the data from the Dcache is formatted and written to either the integer register file (IRF) or floating-point register file (FRF). The formatting required depends on the particular load instruction executed. If the data is not found in the Dcache (a miss), then the address, target register number, and formatting information are entered in the miss address file (MAF).

The MAF performs a load-merging function. When a load miss occurs, each MAF entry is checked to see if it contains a load miss that addresses the same Dcache (32-byte) block. If it does, and certain merging rules are satisfied, then the new load miss is merged with an existing MAF entry. This allows the Mbox to service two or more load misses with one data fill from the Cbox.

There are six MAF entries for load misses and four more for Ibox instruction fetches and prefetches. Load misses are usually the highest Mbox priority.

Refer to Section 2.5 for information on load-merging rules.

2.1.4.3 Dcache Control and Store Instructions

The Dcache follows a write-through protocol. During the execution of a store instruction, the Mbox probes the Dcache to determine whether the location to be overwritten is currently cached. If so (a Dcache hit), the Dcache is updated. Regardless of the Dcache state, the Mbox forwards the data to the Cbox.

A load instruction that is issued one cycle after a store instruction in the pipeline creates a conflict if both the load and store operations access the same memory location. (The store instruction has not yet updated the location when the load instruction reads it.) This conflict is handled by forcing the load instruction to take a replay trap; that is, the Ibox flushes the pipeline and restarts execution from the load instruction. By the time the load instruction arrives at the Dcache the second time, the conflicting store instruction has written the Dcache and the load instruction is executed normally.

Replay traps can be avoided by scheduling the load instruction to issue three cycles after the store instruction. If the load instruction is scheduled to issue two cycles after the store instruction, then it will be issue-stalled for one cycle.

2.1.4.4 Write Buffer

The Mbox contains a write buffer that has six 32-byte entries, each of which holds the data from one or more store instructions that access the same 32-byte block in memory until the data is written into the Scache. The write buffer provides a finite, high-bandwidth resource for receiving store data to minimize the number of CPU stall cycles. The write buffer and associated WMB instruction are described in Section 2.7.

2.1.5 Cache Control and Bus Interface Unit

The cache control and bus interface unit (Cbox) processes all accesses sent by the Mbox and implements all memory-related external interface functions, particularly the coherence protocol functions for write-back caching. It controls the second-level cache (Scache) and the optional board-level backup cache (Bcache). The Cbox handles all instruction and primary Dcache read misses, performs the function of writing data from the write buffer into the shared coherent memory subsystem, and has a major role in executing the Alpha memory barrier (MB) instruction. The Cbox also controls the 128-bit bidirectional data bus, address bus, and I/O control. Chapter 4 describes the external interface.

2.1.6 Cache Organization

The 21164 has three onchip caches—a primary data cache (Dcache), a primary instruction cache (Icache), and a second-level data and instruction cache (Scache). All memory cells in the onchip caches are fully static, 6-transistor, CMOS structures.

The 21164 also provides control for an optional board-level, external cache (Bcache).

2.1.6.1 Data Cache

The data cache (Dcache) is a dual-read-ported, single-write-ported, 8K-byte cache. It is a write-through, read-allocate, direct-mapped, physical cache with 32-byte blocks.

2.1.6.2 Instruction Cache

The instruction cache (Icache) is an 8K-byte, virtual, direct-mapped cache with 32-byte blocks. Each block tag contains:

- A 7-bit address space number (ASN) field as defined by the Alpha architecture
- A 1-bit address space match (ASM) field as defined by the Alpha architecture
- A 1-bit PALcode (physically addressed) indicator

Software, rather than Icache hardware, maintains Icache coherence with memory.

2.1.6.3 Second-Level Cache

The second-level cache (Scache) is a 96K-byte, 3-way, set-associative, physical, write-back, write-allocate cache with 32- or 64-byte blocks. It is a mixed data and instruction cache. The Scache is fully pipelined; it processes read and write operations at the rate of one INT16 per CPU cycle and can alternate between read and write accesses without bubble cycles.

When operating in 32-byte block mode, the Scache has 64-byte blocks with 32-byte subblocks, one tag per block. If configured to 32 bytes, the Scache is organized as three sets of 512 blocks, with each block divided into two 32-byte subblocks. If configured to 64 bytes, the Scache is three sets of 512 64-byte blocks.

2.1.6.4 External Cache

The Cbox implements control for an optional, external, direct-mapped, physical, write-back, write-allocate cache with 32- or 64-byte blocks. The 21164 supports board-level cache sizes of 1, 2, 4, 8, 16, 32, and 64 megabytes.

2.1.7 Serial Read-Only Memory Interface

The serial read-only memory (SROM) interface provides the initialization data load path from a system SROM to the Icache. Chapter 7 provides information about the SROM interface.

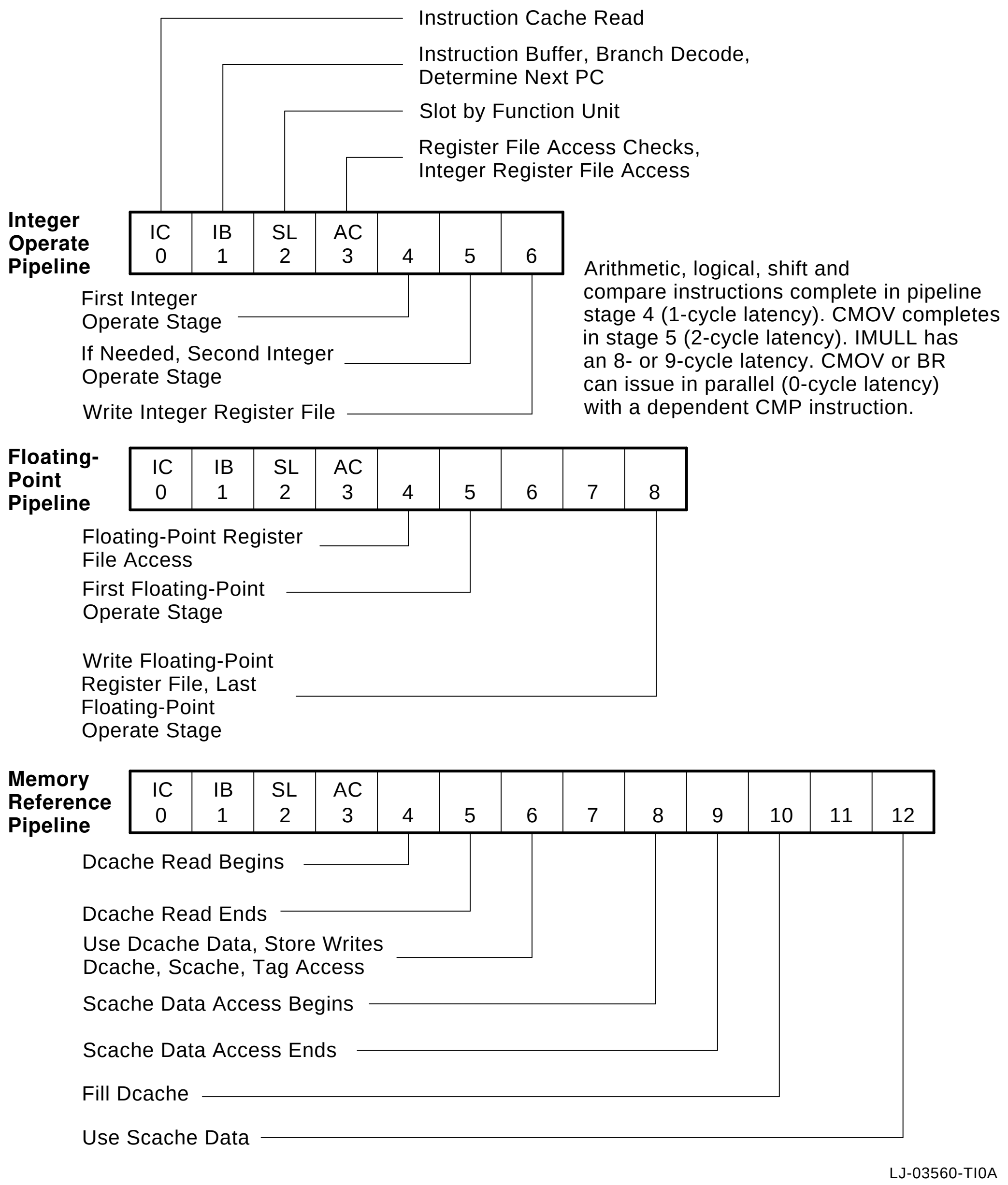
2.2 Pipeline Organization

The 21164 has a 7-stage (or 7-cycle) pipeline for integer operate and memory reference instructions, and a 9-stage pipeline for floating-point operate instructions. The Ibox maintains state for all pipeline stages to track outstanding register write operations.

Figure 2–2 shows the integer operate, memory reference, and floating-point operate pipelines for the Ibox, FPU, Ebox, and Mbox. The first four stages are executed in the Ibox. Remaining stages are executed by the Ebox, Fbox, Mbox, and Cbox. There are bypass paths that allow the result of one instruction to be used as a source operand of a following instruction before it is written to the register file.

Tables 2–2, 2–3, 2–4, 2–5, 2–6, and 2–7 provide examples of events at various stages of pipelining during instruction execution.

Figure 2–2 Instruction Pipeline Stages



2.2 Pipeline Organization

Table 2–2 Pipeline Examples—All Cases

Pipeline Stage	Events
0	Access Icache tag and data.
1	Buffer four instructions, check for branches, calculate branch displacements, and check for Icache hit.
2	Slot-swap instructions around so they are headed for pipelines capable of executing them. Stall preceding stages if all instructions in this stage cannot issue simultaneously because of function unit conflicts.
3	Check the operands of each instruction to see that the source is valid and available and that no write-write hazards exist. Read the IRF. Stall preceding stages if any instruction cannot be issued. All source operands must be available at the end of this stage for the instruction to issue.

Table 2–3 Pipeline Examples—Integer Add

Pipeline Stage	Events
4	Perform the add operation.
5	Result is available for use by an operate function in this cycle.
6	Write the IRF. Result is available for use by an operate function in this cycle.

Table 2–4 Pipeline Examples—Floating Add

Pipeline Stage	Events
4	Read the FRF.
5	First stage of Fbox add pipeline.
6	Second stage of Fbox add pipeline.
7	Third stage of Fbox add pipeline.
8	Fourth stage of Fbox add pipeline. Write the FRF.
9	Result is available for use by an operate function in this cycle. For instance, pipeline stage 5 of the user instruction can coincide with pipeline stage 9 of the producer (latency of 4).

Table 2–5 Pipeline Examples—Load (Dcache Hit)

Pipeline Stage[1]	Events
4	Calculate the effective address. Begin the Dcache data and tag store access.
5	Finish the Dcache data and tag store access. Detect Dcache hit. Format the data as required. Scache arbitration defaults to pipe E0 in anticipation of a possible miss.
6	Write the IRF or FRF. Data is available for use by an operate function in this cycle.

[1]Pipe E0 has not been defined at this point.

Table 2–6 Pipeline Examples—Load (Dcache Miss)

Pipeline Stage[1]	Events
4	Calculate the effective address. Begin the Dcache data and tag store access.
5	Finish the Dcache data and tag store access. Detect Dcache miss. Scache arbitration defaults to pipe E0 in anticipation of a possible miss. If there are load instructions in both E0 and E1, the load instruction in E1 would be delayed at least one more cycle because default arbitration speculatively assumes the load in E0 will miss.
6	Begin Scache tag read operation.
7	Finish Scache tag read operation. Begin detecting Scache hit.
8	Finish detecting Scache hit. Begin accessing the correct Scache data bank. (Bcache index at interface—Bcache access begins.)
9	Finish the Scache data bank access. Begin sending fill data from the Scache.
10	Finish sending fill data from the Scache. Begin Dcache fill. Format the data as required.
11	Finish the Dcache fill. Write the integer or floating-point register file.
12	Data is available for use by an operate function in this cycle.

[1]Pipes E0 and E1 have not been defined at this point.

Table 2–7 Pipeline Examples—Store (Dcache Hit)

Pipeline Stage	Events
4	Calculate the effective address. Begin the Dcache tag store access.
5	Finish the Dcache tag store access. Detect Dcache hit. Send store to the write buffer simultaneously.
6	Write the Dcache data store if hit (write begins this cycle).

2.2.1 Pipeline Stages and Instruction Issue

The 21164 pipeline divides instruction processing into four static and a number of dynamic stages of execution. The first four stages consist of the instruction fetch, buffer and decode, slotting, and issue-check logic. These stages are static in that instructions may remain valid in the same pipeline stage for multiple cycles while waiting for a resource or stalling for other reasons. Dynamic stages (Ebox and Fbox) always advance state and are unaffected by any stall in the pipeline. A pipeline stall may occur while zero instructions issue, or while some instructions of a set of four issue and the others are held at the issue stage. A pipeline stall implies that a valid instruction is (or instructions are) presented to be issued but cannot proceed.

Upon satisfying all issue requirements, instructions are issued into their slotted pipeline. After issuing, instructions cannot stall in a subsequent pipeline stage. The issue stage is responsible for ensuring that all resource conflicts are resolved before an instruction is allowed to continue. The only means of stopping instructions after the issue stage is an abort condition. (The term abort as used here is different from its use in the *Alpha Architecture Reference Manual*.)

2.2.2 Aborts and Exceptions

Aborts result from a number of causes. In general, they can be grouped into two classes, exceptions (including interrupts) and nonexceptions. The difference between the two is that exceptions require that the pipeline be drained of all outstanding instructions before restarting the pipeline at a redirected address. In either case, the pipeline must be flushed of all instructions that were fetched subsequent to the instruction that caused the abort condition (arithmetic exceptions are an exception to this rule). This includes aborting some instructions of a multiple-issued set in the case of an abort condition on the one instruction in the set.

The nonexception case does not need to drain the pipeline of all outstanding instructions ahead of the aborting instruction. The pipeline can be restarted immediately at a redirected address. Examples of nonexception abort conditions are branch mispredictions, subroutine call/return mispredictions, and replay traps. Data cache misses can cause aborts or issue stalls depending on the cycle-by-cycle timing.

In the event of an exception other than an arithmetic exception, the processor aborts all instructions issued after the exceptional instruction, as described in the preceding paragraphs. Due to the nature of some exception conditions, this may occur as late as the integer register file (IRF) write cycle. In the case of an arithmetic exception, the processor may execute instructions issued after the exceptional instruction.

After aborting, the address of the exceptional instruction or the immediately subsequent instruction is latched in the EXC_ADDR internal processor register (IPR). In the case of an arithmetic exception, EXC_ADDR contains the address of the instruction immediately after the last instruction executed. (Every instruction prior to the last instruction executed was also executed.) For machine check and interrupts, EXC_ADDR points to the instruction immediately following the last instruction executed. For the remaining cases, EXC_ADDR points to the exceptional instruction; where, in all cases, its execution should naturally restart.

When the pipeline is fully drained, the processor begins instruction execution at the address given by the PALcode dispatch. The pipeline is drained when all outstanding write operations to both the IRF and FRF have completed and all outstanding instructions have passed the point in the pipeline such that they are guaranteed to complete without an exception in the absence of a machine check.

Replay traps are aborts that occur when an instruction requires a resource that is not available at some point in the pipeline. These are usually Mbox resources whose availability could not be anticipated accurately at issue time (refer to Section 2.4). If the necessary resource is not available when the instruction requires it, the instruction is aborted and the Ibox begins fetching at exactly that instruction, thereby replaying the instruction in the pipeline. A slight variation on this is the load-miss-and-use replay trap in which an operate instruction is issued just as a Dcache hit is being evaluated to determine if one of the instruction's operands is valid. If the result is a Dcache miss, then the operate instruction is aborted and replayed.

2.2.3 Nonissue Conditions

There are two reasons for nonissue conditions. The first is a pipeline stall wherein a valid instruction or set of instructions are prepared to issue but cannot due to a resource conflict (register conflict or function unit conflict). These types of nonissue cycles can be minimized through code scheduling.

The second type of nonissue conditions consists of pipeline bubbles where there is no valid instruction in the pipeline to issue. Pipeline bubbles result from the abort conditions described in the previous section. In addition, a single pipeline bubble is produced whenever a branch type instruction is predicted to be taken, including subroutine calls and returns.

Pipeline bubbles are reduced directly by the instruction buffer hardware and through bubble squashing, but can also be effectively minimized through careful coding practices. Bubble squashing involves the ability of the first four pipeline stages to advance whenever a bubble or buffer slot is detected in the pipeline stage immediately ahead of it while the pipeline is otherwise stalled.

2.3 Scheduling and Issuing Rules

The following sections define the classes of instructions and provide rules for instruction slotting, instruction issuing, and latency.

2.3.1 Instruction Class Definition and Instruction Slotting

The scheduling and multiple issue rules presented here are performance related only; that is, there are no functional dependencies related to scheduling or multiple issuing. The rules are defined in terms of instruction classes. Table 2–8 specifies all of the instruction classes and the pipeline that executes the particular class. With a few additional rules, the table provides the information necessary to determine the functional resource conflicts that determine which instructions can issue in a given cycle.

Table 2–8 Instruction Classes and Slotting

Class Name	Pipeline	Instruction List
LD	E0[1] or E1[2]	All loads except LD*x*_L
ST	E0	All stores except ST*x*_C

[1]Ebox pipeline 0.

[2]Ebox pipeline 1.

(continued on next page)

Table 2–8 (Cont.) Instruction Classes and Slotting

Class Name	Pipeline	Instruction List
MBX	E0	LD*x*_L, MB, WMB, ST*x*_C, HW_LD-lock, HW_ST-cond, FETCH
RX	E0	RS, RC
MXPR	E0 or E1 (depends on the IPR)	HW_MFPR, HW_MTPR
IBR	E1	Integer conditional branches
FBR	FA[3]	Floating-point conditional branches
JSR	E1	Jump-to-subroutine instructions: JMP, JSR, RET, or JSR_COROUTINE, BSR, BR, HW_REI, CALLPAL
IADD	E0 or E1	ADDL, ADDL/V, ADDQ, ADDQ/V, SUBL, SUBL/V, SUBQ, SUBQ/V, S4ADDL, S4ADDQ, S8ADDL, S8ADDQ, S4SUBL, S4SUBQ, S8SUBL, S8SUBQ, LDA, LDAH
ILOG	E0 or E1	AND, BIS, XOR, BIC, ORNOT, EQV
SHIFT	E0	SLL, SRL, SRA, EXTQL, EXTLL, EXTWL, EXTBL, EXTQH, EXTLH, EXTWH, MSKQL, MSKLL, MSKWL, MSKBL, MSKQH, MSKLH, MSKWH, INSQL, INSLL, INSWL, INSBL, INSQH, INSLH, INSWH, ZAP, ZAPNOT
CMOV	E0 or E1	CMOVEQ, CMOVNE, CMOVLT, CMOVLE, CMOVGT, CMOVGE, CMOVLBS, CMOVLBC
ICMP	E0 or E1	CMPEQ, CMPLT, CMPLE, CMPULT, CMPULE, CMPBGE
IMULL	E0	MULL, MULL/V
IMULQ	E0	MULQ, MULQ/V
IMULH	E0	UMULH
FADD	FA	Floating-point operates, including CPYSN and CPYSE, except multiply, divide, and CPYS
FDIV	FA	Floating-point divide
FMUL	FM[4]	Floating-point multiply
FCPYS	FM or FA	CPYS, not including CPYSN or CPYSE

[3]Fbox add pipeline.

[4]Fbox multiply pipeline.

(continued on next page)

Table 2–8 (Cont.) Instruction Classes and Slotting

Class Name	Pipeline	Instruction List
MISC	E0	RPCC, TRAPB
UNOP	None	UNOP[5]

[5]UNOP is LDQ_U R31,0(Rx).

Slotting

The slotting function in the Ibox determines which instructions will be sent forward to attempt to issue. The slotting function detects and removes all static functional resource conflicts. The set of instructions output by the slotting function will issue if no register or other dynamic resource conflict is detected in stage 3 of the pipeline. The slotting algorithm follows:

> Starting from the first (lowest addressed) valid instruction in the INT16 in stage 2 of the 21164 Ibox pipeline, attempt to assign that instruction to one of the four pipelines (E0, E1, FA, FM). If it is an instruction that can issue in either E0 or E1, assign it to E0. However, if one of the following is true, assign it to E1:
>
> - E0 is not free and E1 is free.
> - The next integer instruction[1] in this INT16 can issue only in E0.
>
> If the current instruction is one that can issue in either FA or FM, assign it to FA unless FA is not free. If it is an FA-only instruction, it must be assigned to FA. If it is FM-only instruction, it must be assigned to FM. Mark the pipeline selected by this process as taken and resume with the next sequential instruction. Stop when an instruction cannot be allocated in an execution pipeline because any pipeline it can use is already taken.

The slotting logic does not send instructions forward out of logical instruction order because the 21164 always issues instructions in order. The slotting logic also enforces the special rules in the following list, stopping the slotting process when a rule would be violated by allocating the next instruction an execution pipeline:

- An instruction of class LD cannot be issued simultaneously with an instruction of class ST.

[1] In this context, an integer instruction is one that can issue in one or both of E0 or E1, but not FA or FM.

- All instructions are discarded at the slotting stage after a predicted-taken IBR or FBR class instruction, or a JSR class instruction.
- After a predicted not-taken IBR or FBR, no other IBR, FBR, or JSR class can be slotted together.
- The following cases are detected by the slotting logic:
  - From lowest address to highest within an INT16, with the following arrangement:

    ```
    I-instruction, F-instruction, I-instruction, I-instruction
    ```

    I-instruction is any instruction that can issue in one or both of E0 or E1. F-instruction is any instruction that can issue in one or both of FA or FM.
  - From lowest address to highest within an INT16, with the following arrangement:

    ```
    F-instruction, I-instruction, I-instruction, I-instruction
    ```

  When this type of case is detected, the first two instructions are forwarded to the issue point in one cycle. The second two are sent only when the first two have both issued, provided no other slotting rule would prevent the second two from being slotted in the same cycle.

2.3.2 Coding Guidelines

Code should be scheduled according to latency and function unit availability. This is good practice in most RISC architectures. Code alignment and the effects of split-issue[1] should be considered.

Instructions [a] (the LDL) and [b] (the first ADDL) in the following example are slotted together. Instruction [b] stalls (split-issue), thus preventing instruction [c] from advancing to the issue stage:

```
Code example showing              Code example showing
incorrect ordering                correct ordering

(1) [a] LDL   R2,0(R1)            (1) [d] LDL   R2,0(R1)
(3) [b] ADDL  R2,R3,R4            (1) [e] NOP
(4) [c] ADDL  R2,R5,R6            (3) [f] ADDL  R2,R3,R4
                                  (3) [g] ADDL  R2,R5,R6

NOTES: The instruction examples are assumed to begin on an
       INT16 alignment.  (n) = Expected execute cycle.
```

[1] Split-issue is the situation in which not all instructions sent from the slotting stage to the issue stage issue. One or more stalls result.

Eventually [b] issues when the result of [a] is returned from a presumed Dcache hit. Instruction [c] is delayed because it cannot advance to the issue stage until [b] issues.

In the improved sequence, the LDL [d] is slotted with the NOP [e]. Then the first ADDL [f] is slotted with the second ADDL [g] and those two instructions dual-issue. This sequence takes one less cycle to complete than the first sequence.

2.3.3 Instruction Latencies

After slotting, instruction issue is governed by the availability of registers for read or write operations, and the availability of the floating divide unit and the integer multiply unit. There are producer–consumer dependencies, producer–producer dependencies (also known as write-after-write conflicts), and dynamic function unit availability dependencies (integer multiply and floating divide). The Ibox logic in stage 3 of the 21164 pipeline detects all these conflicts.

The latency to produce a valid result for most instructions is fixed. The exceptions are loads that miss, floating-point divides, and integer multiplies. Table 2–9 gives the latencies for each instruction class. A latency of 1 means that the result may be used by an instruction issued one cycle after the producing instruction. Most latencies are only a property of the producer. An exception is integer multiply latencies. There are no variations in latency due to which a particular unit produces a given result relative to the particular unit that consumes it. In the case of integer multiply, the instruction is issued at the time determined by the standard latency numbers. The multiply's latency is dependent on which previous instructions produced its operands and when they executed.

Table 2–9 Instruction Latencies

Class	Latency	Additional Time Before Result Available to Integer Multiply Unit[1]
LD	Dcache hits, latency=2. Dcache miss/Scache hit, latency=8 or longer.[2]	1 cycle
ST	Store operations produce no result.	—
MBX	LD*x*_L always Dcache misses, latency depends on memory subsystem state. ST*x*_C, latency depends on memory subsystem state. MB, WMB, and FETCH produce no result.	—
RX	RS, RC, latency=1.	2 cycles
MXPR	HW_MFPR, latency=1, 2, or longer, depending on the IPR. HW_MTPR, produces no result.	1 or 2 cycles
IBR	Produces no result. (Taken branch issue latency minimum = 1 cycle, branch mispredict penalty = 5 cycles.)	—
FBR	Produces no result. (Taken branch issue latency minimum = 1 cycle, branch mispredict penalty = 5 cycles.)	—
JSR	All but HW_REI, latency=1. HW_REI produces no result. (Issue latency—minimum 1 cycle.)	2 cycles
IADD	Latency=1.	2 cycles
ILOG	Latency=1.[4]	2 cycles

[1]The multiplier is unable to receive data from Ebox bypass paths. The instruction issues at the expected time, but its latency is increased by the time it takes for the input data to become available to the multiplier. For example, an IMULL instruction issued one cycle later than an ADDL instruction, which produced one of its operands, has a latency of 10 (8 + 2). If the IMULL instruction is issued two cycles later than the ADDL instruction, the latency is 9 (8 + 1).

[2]When idle, Scache arbitration predicts a load miss in E0. If a load actually does miss in E0, it is sent to the Scache immediately. If it hits, and no other event in the Cbox affects the operation, the requested data is available for use in eight cycles. Otherwise, the request takes longer (possibly much longer, depending on the state of the Scache and Cbox). It should be possible to schedule some unrolled code loops for Scache by using a data access pattern that takes advantage of the Mbox load-merging function, achieving high throughput with large data sets.

[4]A special bypass provides an effective latency of 0 (zero) cycles for an ICMP or ILOG instruction producing the test operand of an IBR or CMOV instruction. This is true only when the IBR or CMOV instruction issues in the same cycle as the ICMP or ILOG instruction that produced the test operand of the IBR or CMOV instruction. In all other cases, the effective latency of ICMP and ILOG instruction is one cycle.

(continued on next page)

Table 2–9 (Cont.) Instruction Latencies

Class	Latency	Additional Time Before Result Available to Integer Multiply Unit[1]
SHIFT	Latency=1.	2 cycles
CMOV	Latency=2.	1 cycle
ICMP	Latency=1.[4]	2 cycles
IMULL	Latency=8, plus up to 2 cycles of added latency, depending on the source of the data.[1] Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 4 cycles plus the number of cycles added to the latency.	1 cycle
IMULQ	Latency=12, plus up to 2 cycles of added latency, depending on the source of the data.[1] Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 8 cycles plus the number of cycles added to the latency.	1 cycle
IMULH	Latency=14, plus up to 2 cycles of added latency, depending on the source of the data.[1] Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 8 cycles plus the number of cycles added to the latency.	1 cycle
FADD	Latency=4.	—
FDIV	Data-dependent latency: 15 to 31 single precision, 22 to 60 double precision. Next floating divide can be issued in the same cycle. The result of the previous divide is available, regardless of data dependencies.	—
FMUL	Latency=4.	—
FCPYS	Latency=4.	—

[1]The multiplier is unable to receive data from Ebox bypass paths. The instruction issues at the expected time, but its latency is increased by the time it takes for the input data to become available to the multiplier. For example, an IMULL instruction issued one cycle later than an ADDL instruction, which produced one of its operands, has a latency of 10 (8 + 2). If the IMULL instruction is issued two cycles later than the ADDL instruction, the latency is 9 (8 + 1).

[4]A special bypass provides an effective latency of 0 (zero) cycles for an ICMP or ILOG instruction producing the test operand of an IBR or CMOV instruction. This is true only when the IBR or CMOV instruction issues in the same cycle as the ICMP or ILOG instruction that produced the test operand of the IBR or CMOV instruction. In all other cases, the effective latency of ICMP and ILOG instruction is one cycle.

(continued on next page)

Table 2–9 (Cont.) Instruction Latencies

Class	Latency	Additional Time Before Result Available to Integer Multiply Unit[1]
MISC	RPCC, latency=2. TRAPB produces no result.	1 cycle
UNOP	UNOP produces no result.	—

[1]The multiplier is unable to receive data from Ebox bypass paths. The instruction issues at the expected time, but its latency is increased by the time it takes for the input data to become available to the multiplier. For example, an IMULL instruction issued one cycle later than an ADDL instruction, which produced one of its operands, has a latency of 10 (8 + 2). If the IMULL instruction is issued two cycles later than the ADDL instruction, the latency is 9 (8 + 1).

2.3.3.1 Producer–Producer Latency

Producer–producer latency, also known as write-after-write conflicts, cause issue-stalls to preserve write order. If two instructions write the same register, they are forced to do so in different cycles by the Ibox. This is necessary to ensure that the correct result is left in the register file after both instructions have executed. For most instructions, the order in which they write the register file is dictated by issue order. However IMUL, FDIV, and LD instructions may require more time than other instructions to complete. Subsequent instructions that write the same destination register are issue-stalled to preserve write ordering at the register file.

Conditions that involve an intervening producer–consumer conflict can occur commonly in a multiple-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 following code:

```
LDQ  R2,0(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
```

Producer–producer latency is generally determined by applying the rule that register file write operations must occur in the correct order (enforced by Ibox hardware). Two IADD or ILOG class instructions that write the same register issue at least one cycle apart. The same is true of a pair of CMOV-class instructions, even though their latency is 2. For IMUL, FDIV, and LD instructions, producer–producer conflicts with any subsequent instruction results in the second instruction being issue-stalled until the IMUL, FDIV, or LD instruction is about to complete. The second instruction is issued as soon as it is guaranteed to write the register file at least one cycle after the IMUL, FDIV, or LD instruction.

If a load writes a register, and within two cycles a subsequent instruction writes the same register, the subsequent instruction is issued speculatively, assuming the load hits. If the load misses, a load-miss-and-use trap is generated. This causes the second instruction to be replayed by the Ibox. When the second instruction again reaches the issue point, it is issue-stalled until the load fill occurs.

2.3.4 Issue Rules

The following is a list of conditions that prevent the 21164 from issuing an instruction:

- No instruction can be issued until all of its source and destination registers are clean; that is, all outstanding write operations to the destination register are guaranteed to complete in issue order and there are no outstanding write operations to the source registers, or those write operations can be bypassed.

  Technically, load-miss-and-use replay traps are an exception to this rule. The consumer of the load's result issues, and is aborted, because a load was predicted to hit and was discovered to miss just as the consumer instruction issued. In practice, the only difference is that the latency of the consumer may be longer than it would have been had the issue logic ‘‘known’’ the load would miss in time to prevent issue.
- An instruction of class LD cannot be issued in the second cycle after an instruction of class ST is issued.
- No LD, ST, MXPR (to an Mbox register), or MBX class instructions can be issued after an MB instruction has been issued until the MB instruction has been acknowledged by the Cbox.
- No LD, ST, MXPR (to an Mbox register), or MBX class instructions can be issued after a ST*x*_C (or HW_ST-cond) instruction has been issued until the Mbox writes the success/failure result of the ST*x*_C (HW_ST-cond) in its destination register.
- No IMUL instructions can be issued if the integer multiplier is busy.
- No floating-point divide instructions can be issued if the floating-point divider is busy.
- No instruction can be issued to pipe E0 exactly two cycles before an integer multiplication completes.

- No instruction can be issued to pipe FA exactly five cycles before a floating-point divide completes.
- No instruction can be issued to pipe E0 or E1 exactly two cycles before an integer register fill is requested (speculatively) by the Cbox, except IMULL, IMULQ, and IMULH instructions and instructions that do not produce any result.
- No LD, ST, or MBX class instructions can be issued to pipe E0 or E1 exactly one cycle before a integer register fill is requested (speculatively) by the Cbox.
- No instruction issues after a TRAPB instruction until all previously issued instructions are guaranteed to finish without generating a trap other than a machine check.

All instructions sent to the issue stage (stage 3) by the slotting logic (stage 2) are issued subject to the previous rules. If issue is prevented for a given instruction at the issue stage, all logically subsequent instructions at that stage are prevented from issuing automatically. The 21164 only issues instructions in order.

2.4 Replay Traps

There are no stalls after the instruction issue point in the pipeline. In some situations, an Mbox instruction cannot be executed because of insufficient resources (or some other reason). These instructions trap and the Ibox restarts their execution from the beginning of the pipeline. This is called a replay trap. Replay traps occur in the following cases:

- The write buffer is full when a store instruction is executed and there are already six write buffer entries allocated. The trap occurs even if the entry would have merged in the write buffer.
- A load instruction is issued in pipe E0 when all six MAF entries are valid (not available), or a load instruction issued in pipe E1 when five of the six MAF entries are valid. The trap occurs even if the load instruction would have hit in the Dcache or merged with an MAF entry.
- Alpha shared memory model order trap (Litmus test 1 trap): If a load instruction issues that address matches with any miss in the MAF, the load instruction is aborted through a replay trap regardless of whether the newly issued load instruction hits or misses in the Dcache. The address match is precise except that it includes the case in which a longword access matches within a quadword access. This ensures that the two loads execute in issue order.

- Load-after-store trap: A replay trap occurs if a load instruction is issued in the cycle immediately following a store instruction that hits in the Dcache, and both access the same location. The address match is exact for address bits <12:2> (longword granularity), but ignores address bits <42:13>.
- When a load instruction is followed, within one cycle, by any instruction that uses the result of that load, and the load misses in the Dcache, the consumer instruction traps and is restarted from the beginning of the pipeline. This occurs because the consumer instruction is issued speculatively while the Dcache hit is being evaluated. If the load misses in the Dcache, the speculative issue of the consumer instruction was incorrect. The replay trap generally brings the consumer instruction to the issue point before or simultaneously with the availability of fill data.

2.5 Miss Address File and Load-Merging Rules

The following sections describe the miss address file (MAF) and its load-merging function, and the load-merging rules that apply after a load miss.

2.5.1 Merging Rules

When a load miss occurs, each MAF entry is checked to see if it contains a load miss that addresses the same 32-byte Dcache block. If it does, and certain merging rules [2] are satisfied, then the new load miss is merged with an existing MAF entry. This allows the Mbox to service two or more load misses with one data fill from the Cbox. The merging rules for an individual MAF entry are as follows:

- Merging only occurs if the new load miss addresses a different INT8 from all loads previously entered or merged to that MAF entry.
- Merging only occurs if the new load miss is the same access size as the load instructions previously entered in that MAF entry. That is, quadword load instructions merge only with other quadword load instructions and longword load instructions merge only with other longword load instructions.
- In the case of longword load instructions, both <02> address bits must be the same. That is, longword load instructions with even addresses merge only with other even longword load instructions, and longword load instructions with odd addresses merge only with other odd longword load instructions.

[2] Merging rules result primarily from limitations of the implementation.

- The MAF does not merge floating-point and integer load misses in the same entry.
- Merging is prevented for the MAF entry a certain number of cycles after the Scache access corresponding to the MAF entry begins. Merging is prevented for that entry only if the Scache access hits. The minimum number of cycles of merging is three; the cycle in which the first load is issued, and the two subsequent cycles. This corresponds to the most optimistic case of a load miss being forwarded to the Scache without delay (accounting for the cycle saved by the bypass that sends new load misses directly to the Scache when there is nothing else pending).

2.5.2 Read Requests to the Cbox

When merging does not occur, a new MAF entry is allocated for the new load miss. Merging is done for two load instructions issued simultaneously, which both miss in effect as if they were issued sequentially with the load from Ebox pipe E0 first. The Mbox sends a read request to the Cbox for each MAF entry allocated.

A bypass is provided so that if the load instruction issues in Ebox pipe E0, and no MAF requests are pending, the load instruction's read request is sent to the Cbox immediately. Similarly, if a load instruction from Ebox pipe E1 misses, and there was no load instruction in pipe E0 to begin with, the E1 load miss is sent to the Cbox immediately. In either case, the bypassed read request is aborted if the load hits in the Dcache or merges in the MAF.

2.5.3 Load Instructions to Noncacheable Space

Merging is normally allowed for load instructions to noncacheable space (physical address bit <39> = 1). It is prevented when MAF_MODE<03>=1 (see Section 5.2.16). At the external interface, these read instructions tell the system environment which INT32 is addressed and which of the INT8s within the INT32 are actually accessed. Merging stops for a load instruction to noncacheable space as soon as the Cbox accepts the reference. This permits the system environment to access only those INT8s that are actually requested by load instructions. For memory-mapped INT4 registers, the system environment must return the result of reading each register within the INT8. This occurs because the 21164 only indicates those INT8s that are accessed, not the exact length and offset of the access within each INT8. Systems implementing memory-mapped registers with side effects from read instructions should place each such register in a separate INT8 in memory.

2.5.4 MAF Entries and MAF Full Conditions

There are six MAF entries for load misses and four for Ibox instruction fetches and prefetches. Load misses are usually the highest Mbox priority request.

If the MAF is full and a load instruction issues in pipe E0, or if five of the six MAF entries are valid and a load instruction issues in pipe E1, an MAF full trap occurs causing the Ibox to restart execution with the load instruction that caused the MAF overflow. When the load instruction arrives at the MAF the second time, an MAF entry may have become available. If not, the MAF full trap occurs again.

2.5.5 Fill Operation

Eventually, the Cbox provides the data requested for a given MAF entry (a fill). If the fill is integer data and not floating-point data, the Cbox requests that the Ibox allocate two consecutive ‘‘bubble’’ cycles in the Ebox pipelines. The first bubble prevents any instruction from issuing. The second bubble prevents only Mbox instructions (particularly load and store instructions) from issuing. The fill uses the first bubble cycle as it progresses down the Ebox/Mbox pipelines to format the data and load the register file. It uses the second bubble cycle to fill the Dcache.

An instruction typically writes the register file in pipeline stage 6 (see Figure 2–2). Because there is only one register file write port per integer pipeline, a no-instruction bubble cycle is required to reserve a register file write port for the fill. A load or store instruction accesses the Dcache in the second half of stage 4 and the first half of stage 5. The fill operation writes the Dcache, making it unavailable for other accesses at that time. Relative to the register file write operation, the Dcache (write) access for a fill occurs a cycle later than the Dcache access for a load hit. Only load and store instructions use the Dcache in the pipeline. Therefore, the second bubble reserved for a fill is a no-Mbox-instruction bubble.

The second bubble is a subset of the first bubble. When two fills are in consecutive cycles, as in an Scache hit, then three total bubbles are allocated: two no-instruction bubbles, followed by one no-Mbox-instruction bubble. The bubbles are requested speculatively before it is known whether the Scache or the optional external Bcache will hit.

For fills from the Cbox to floating-point registers, no cycle is allocated. Load instructions that conflict with the fill in the pipeline are forced to miss. Store instructions that conflict in the pipeline force the fill to be aborted in order to keep the Dcache available to the store operation. In all cases, the floating-point registers are filled as dictated by the associated MAF entry. The Fbox has separate write ports for fill data as is necessary for this fill scheme.

Up to two floating or integer registers may be written for each Cbox fill cycle. Fills deliver 32 bytes in two cycles: two INT8s per cycle. The MAF merging rules ensure that there is no more than one register to write for each INT8, so that there is a register file write port available for each INT8. After appropriate formatting, data from each INT8 is written into the IRF or FRF provided there is a miss recorded for that INT8.

Load misses are all checked against the write buffer contents for conflicts between new load instructions and previously issued store instructions. Refer to Section 2.7 for more information on write operations.

LDL_L and LDQ_L instructions always allocate a new MAF entry. No load instructions that follow an LDL_L or LDQ_L instruction are allowed to merge with it. After an LDL_L or LDQ_L instruction is issued, the Ibox does not issue any more Mbox instructions until the Mbox has successfully sent the LDL_L or LDQ_L instruction to the Cbox. This guarantees correct ordering between an LDL_L or LDQ_L instruction and a subsequent STL_C or STQ_C instruction even if they access different addresses.

2.6 Mbox Store Instruction Execution

Store instructions execute in the Mbox by:

1. Reading the Dcache tag store instruction in the pipeline stage in which a load instruction would read the Dcache
2. Checking for a hit in the next stage
3. Writing the Dcache data store instruction if there is a hit in the second (following) pipeline stage.

Load instructions are not allowed to issue in the second cycle after a store instruction (one bubble cycle). Other instructions can be issued in that cycle. Store instructions can issue at the rate of one per cycle because store instructions in the Dstream do not conflict in their use of resources. The Dcache tag store and Dcache data store are the principal resources. However, a load instruction uses the Dcache data store in the same early stage that it uses the Dcache tag store. Therefore, a load instruction would conflict with a store instruction if it were issued in the second cycle after any store instruction. Refer to Section 2.2 for more information on store instruction execution in the pipeline.

A load instruction that is issued one cycle after a store instruction in the pipeline creates a conflict if both access exactly the same memory location. This occurs because the store instruction has not yet updated the location when the load instruction reads it. This conflict is handled by forcing the load instruction to replay trap. The Ibox flushes the pipeline and restarts execution from the load instruction. By the time the load instruction arrives at the Dcache the second time, the conflicting store instruction has written the Dcache and the load instruction is executed normally.

Software should not load data immediately after storing it. The replay trap that is incurred ''costs'' seven cycles. The best solution is to schedule the load instruction to issue three cycles after the store. No issue stalls or replay traps will occur in that case. If the load instruction is scheduled to issue two cycles after the store instruction, it will be issue-stalled for one cycle. This is not an optimal solution, but is preferred over incurring a replay trap on the load instruction.

For three cycles during store instruction execution, fills from the Cbox are not placed in the Dcache. Register fills are unaffected. There are conflicts that make it impossible to fill the Dcache in each of these cycles. Fills are prevented in cycles in which a store instruction is in pipeline stage 4, 5, or 6. This always applies to fills of floating-point data. Fills of integer data allocate bubble cycles, such that an integer fill never conflicts with a store instruction in pipeline stages 4 or 5. Instead, a store instruction that would have conflicted in stage 4 or 5 is issue-stalled but an integer fill will conflict with a store instruction in pipeline stage 6.

If a store instruction is stalled at the issue point for any reason, it interferes with fills just as if it had been issued. This applies only to fills of floating-point data.

For each store instruction, a search of the MAF is done to detect load-before-store hazards. If a store instruction is executed, and a load of the same address is present in the MAF, two things happen:

1. Bits are set in each conflicting MAF entry to prevent its fill from being placed in the Dcache when it arrives, and to prevent subsequent load instructions from merging with that MAF entry.
2. Conflict bits are set with the store instruction in the write buffer to prevent the store instruction from being issued until all conflicting load instructions have been issued to the Cbox.

Conflict checking is done at the 32-byte block granularity. This ensures proper results from the load instructions and prevents incorrect data from being cached in the Dcache.

A check is performed for each new store against store instructions in the write buffer that have already been sent to the Cbox but have not been completed. Section 2.7 describes this process.

2.7 Write Buffer and the WMB Instruction

The following sections describe the write buffer and the WMB instruction.

2.7.1 The Write Buffer

The write buffer contains six fully associative 32-byte entries. The purpose of the write buffer is to minimize the number of CPU stall cycles by providing a finite, high-bandwidth resource for receiving store data. This is required because the 21164 can generate store data at the peak rate of one INT8 every CPU cycle. This is greater than the average rate at which the Scache can accept the data if Scache misses occur.

In addition to HW_ST and other store instructions, the STQ_C, STL_C, FETCH, and FETCH_M instructions are also written into the write buffer and sent offchip. However, unlike store instructions, these write buffer-directed instructions are never merged into a write buffer entry with other instructions. A write buffer entry is invalid if it does not contain one of these instructions.

2.7.2 The Write Memory Barrier (WMB) Instruction

The memory barrier (MB) instruction is suitable for ordering memory references of any kind. The WMB instruction forces ordering of write operations only (store instructions). The WMB instruction has a special effect on the write buffer. When it is executed, a bit is set in every write buffer entry containing valid store data that will prevent future store instructions from merging with any of the entries. Also, the next entry to be allocated is marked with a WMB flag. At this point, the entry marked with the WMB flag does not yet have valid data in it. When an entry marked with a WMB flag is ready to issue to the Cbox, the entry is not issued until every previously issued write instruction is complete. This ensures correct ordering between store instructions issued before the WMB instruction and store instructions issued after it.

Each write buffer entry contains a content-addressable memory (CAM) for holding physical address bits <39:05>, 32 bytes of data, eight INT4 mask bits (that indicate which of the eight INT4s in the entry contain valid data), and miscellaneous control bits. Among the control bits are the WMB flag, and a no-merge bit, which indicates that the entry is closed to further merging.

2.7.3 Entry-Pointer Queues

Two entry-pointer queues are associated with the write buffer: a free-entry queue and a pending-request queue. The free-entry queue contains pointers to available invalid write buffer entries. The pending-request queue contains pointers to valid write buffer entries that have not yet been issued to the Cbox. The pending-request queue is ordered in allocation order.

Each time the write buffer is presented with a store instruction, the physical address generated by the instruction is compared to the address in each valid write buffer entry that is open for merging. If the address is in the same INT32 as an address in a valid write buffer entry (that also contains a store instruction), and the entry is open for merging, then the new store data is merged into that entry and the entry's INT4 mask bits are updated. If no matching address is found, or all entries are closed to merging, then the store data is written into the entry at the top of the free-entry queue. This entry is validated, and a pointer to the entry is moved from the free-entry queue to the pending-request queue.

2.7.4 Write Buffer Entry Processing

When two or more entries are in the pending-request queue, the Mbox requests that the Cbox process the write buffer entry at the head of the pending-request queue. Then the Mbox removes the entry from the pending-request queue without placing it in the free-entry queue. When the Cbox has completely processed the write buffer entry, it notifies the Mbox, and the now invalid write buffer entry is placed in the free-entry queue. The Mbox may request that a second write buffer entry be processed while waiting for the Cbox to finish the first. The write buffer entries are invalidated and placed in the free-entry queue in the order that the requests complete. This order may be different from the order in which the requests were made.

The Mbox sends write requests from the write buffer to the Cbox. The Cbox processes these requests according to the cache coherence protocol. Typically, this involves loading the target block into the Scache, making it writable, and then writing it. Because the Scache is write-back, this completes the operation.

The Mbox requests that a write buffer entry be processed every 64 cycles, even if there is only one valid entry. This ensures that write instructions do not wait forever to be written to memory. (This is triggered by a free running timer.)

When an LDL_L or LDQ_L instruction is processed by the Mbox, the Mbox requests processing of the next pending write buffer request. This increases the chances of the write buffer being empty when an STL_C or STQ_C instruction is issued.

The Mbox continues to request that write buffer entries be processed as long as one of the following occurs:

- One buffer contains an STQ_C, STL_C, FETCH, or FETCH_M instruction
- One buffer is marked by a WMB flag
- An MB instruction is being executed by the Mbox.

This ensures that these instructions complete as quickly as possible.

Every store instruction that does not merge in the write buffer is checked against every valid entry. If any entry is an address match, then the WMB flag is set on the newly allocated write buffer entry. This prevents the Mbox from concurrently sending two write instructions to exactly the same block in the Cbox.

Load misses are checked in the write buffer for conflicts. The granularity of this check is an INT32. Any load instruction matching any write buffer entry's address is considered a hit even if it does not access an INT4 marked for update in that write buffer entry. If a load hits in the write buffer, a conflict bit is set in the load instruction's MAF entry, which prevents the load instruction from being issued to the Cbox before the conflicting write buffer entry has been issued and completed. At the same time, the no-merge bit is set in every write buffer entry with which the load hit. A write buffer flush flag is also set. The Mbox continues to request that write buffer entries be processed until all the entries that were ahead of, and including, the conflicting write instructions at the time of the load hit have been processed.

Some write instructions cannot be processed in the Scache without external environment involvement. To support this, the Mbox retransmits a write instruction at the Cbox's request. This situation arises when the Scache block is not dirty when the write instruction is issued, or when the access misses in the Scache.

2.7.5 Ordering of Noncacheable Space Write Instructions

Special logic ensures that write instructions to noncacheable space are sent offchip in the order in which their corresponding buffers were allocated (placed in the pending-request queue).

2.8 Performance Measurement Support–Performance Counters

The 21164 contains a performance recording feature. The implementation of this feature provides a mechanism to count various hardware events and causes an interrupt upon counter overflow. Interrupts are triggered six cycles after the event, and therefore, the exception PC may not reflect the exact instruction causing counter overflow. Three counters are provided to allow accurate comparison of two variables under a potentially nonrepeatable experimental condition. Counter inputs include:

- Issues
- Nonissues
- Total cycles
- Pipe dry
- Pipe freeze
- Mispredicts and cache misses
- Counts for various instruction classifications

In addition, the 21164 provides one signal-pin input (**perf_mon_h**) to measure external events at a maximum rate determined by the selected system clock speed (see Table 5–12).

For information about counter control, refer to the following IPR descriptions:

- Hardware interrupt clear (HWINT_CLR) register (see Section 5.1.23)
- Interrupt summary register (ISR) (see Section 5.1.24)
- Performance counter (PMCTR) register (see Section 5.1.27)
- Bcache control (BC_CONTROL) register bits <24:19> (see Section 5.3.4 and Table 5–31)

2.9 Floating-Point Control Register

Figure 2–3 shows the format of the floating-point control register (FPCR) and Table 2–10 describes the fields.

Figure 2–3 Floating-Point Control Register (FPCR) Format

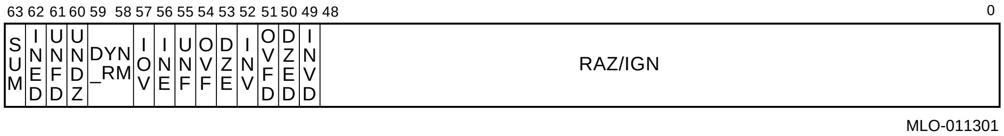


Table 2–10 Floating-Point Control Register Bit Descriptions

Bit	Description (Meaning When Set)
<63>	Summary bit (SUM). Records bitwise OR of FPCR exception bits. Equal to FPCR<57 | 56 | 55 | 54 | 53 | 52>
<62>	Inexact disable (INED). Suppress INE trap and place correct IEEE nontrapping result in the destination register if the 21164 is capable of producing correct IEEE nontrapping result.
<61>	Underflow disable (UNFD). Subset support: Suppress UNF trap if UNDZ is also set and the /S qualifier is set on the instruction.
<60>	Underflow to zero (UNDZ). When set together with UNFD, on underflow, the hardware places a true zero (all 64 bits zero) in the destination register rather than the denormal number specified by the IEEE standard.
<59,58>	Dynamic routing mode (DYN). Indicates the rounding mode to be used by an IEEE floating-point operate instruction when the instruction's function field specifies dynamic mode (/D). The assignments are:

DYN	IEEE Rounding Mode Selected
00	Chopped rounding mode
01	Minus infinity
10	Normal rounding
11	Plus infinity

Bit	Description (Meaning When Set)
<57>	Integer overflow (IOV). An integer arithmetic operation or a conversion from floating to integer overflowed the destination precision.
<56>	Inexact result (INE). A floating arithmetic or conversion operation gave a result that differed from the mathematically exact result.

(continued on next page)

Table 2–10 (Cont.) Floating-Point Control Register Bit Descriptions

Bit	Description (Meaning When Set)
<55>	Underflow (UNF). A floating arithmetic or conversion operation underflowed the destination exponent.
<54>	Overflow (OVF). A floating arithmetic or conversion operation overflowed the destination exponent.
<53>	Division by zero (DZE). An attempt was made to perform a floating divide operation with a divisor of zero.
<52>	Invalid operation (INV). An attempt was made to perform a floating arithmetic, conversion, or comparison operation, and one or more of the operand values were illegal.
<51>	Overflow disable (OVFD). Not supported.
<50>	Division by zero disable (DZED). Not supported.
<49>	Invalid operation disable (INVD). Not supported.
<48:0>	Reserved. Read as zero; ignored when written.

2.10 Design Examples

The 21164 can be designed into many different uniprocessor and multiprocessor system configurations. Figures 2–4, 2–5, and 2–6 illustrate three possible configurations. These configurations employ additional system/memory controller chipsets.

Figure 2–4 shows a typical uniprocessor system with a board-level cache. This system configuration could be used in standalone or networked workstations.

Figure 2–4 Typical Uniprocessor Configuration

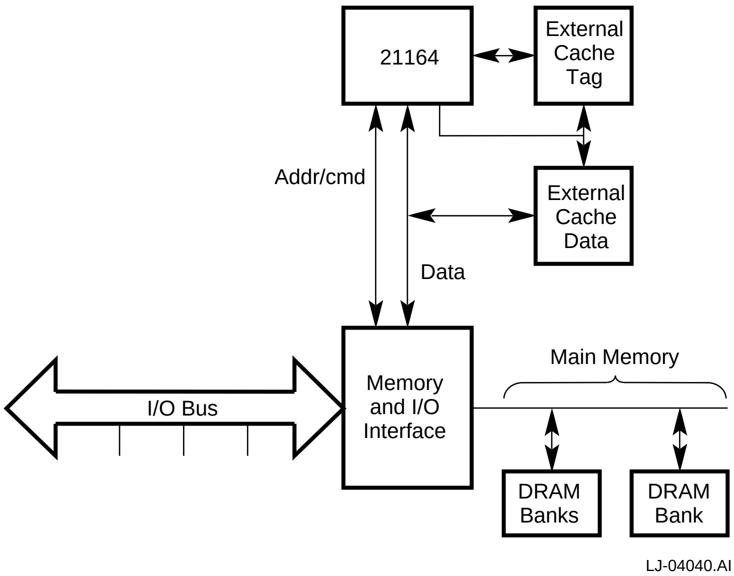


Figure 2–5 shows a typical multiprocessor system, each processor with a board-level cache. Each interface controller must employ a duplicate tag store to maintain cache coherency. This system configuration could be used in a networked database server application.

2.10 Design Examples

Figure 2–5 Typical Multiprocessor Configuration

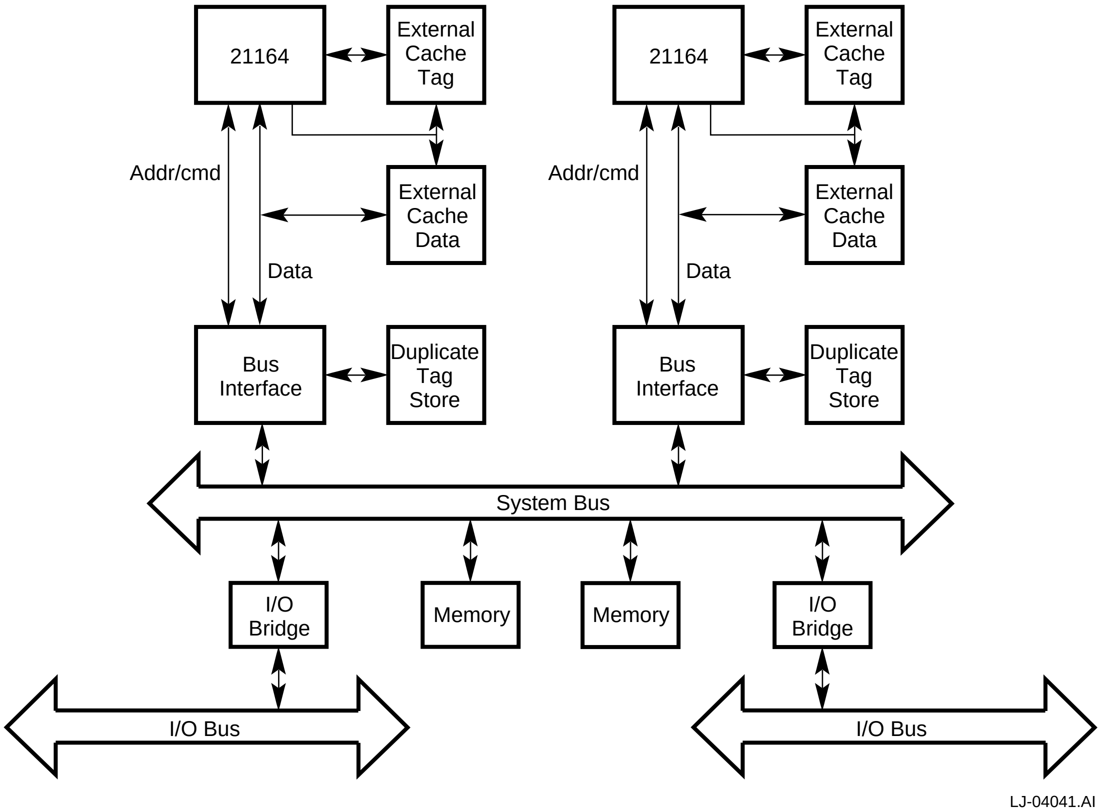
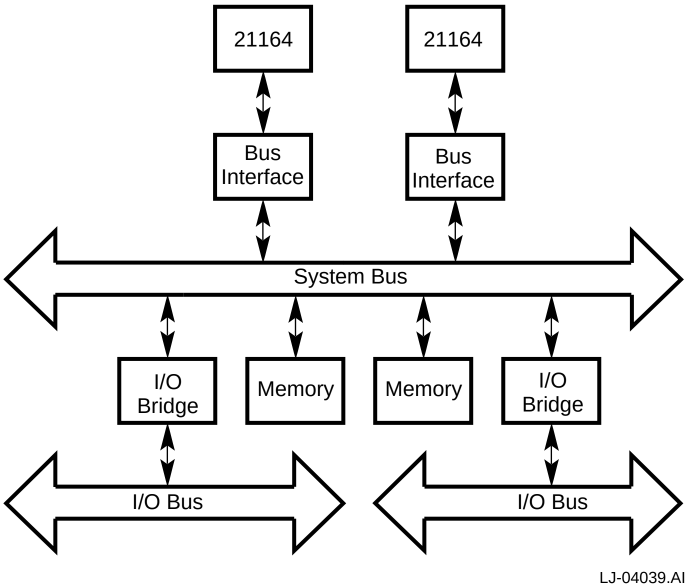


Figure 2–6 shows a cacheless multiprocessor system. This system configuration could be used in high-bandwith dedicated server applications.

Figure 2–6 Cacheless Multiprocessor Configuration



3 Hardware Interface

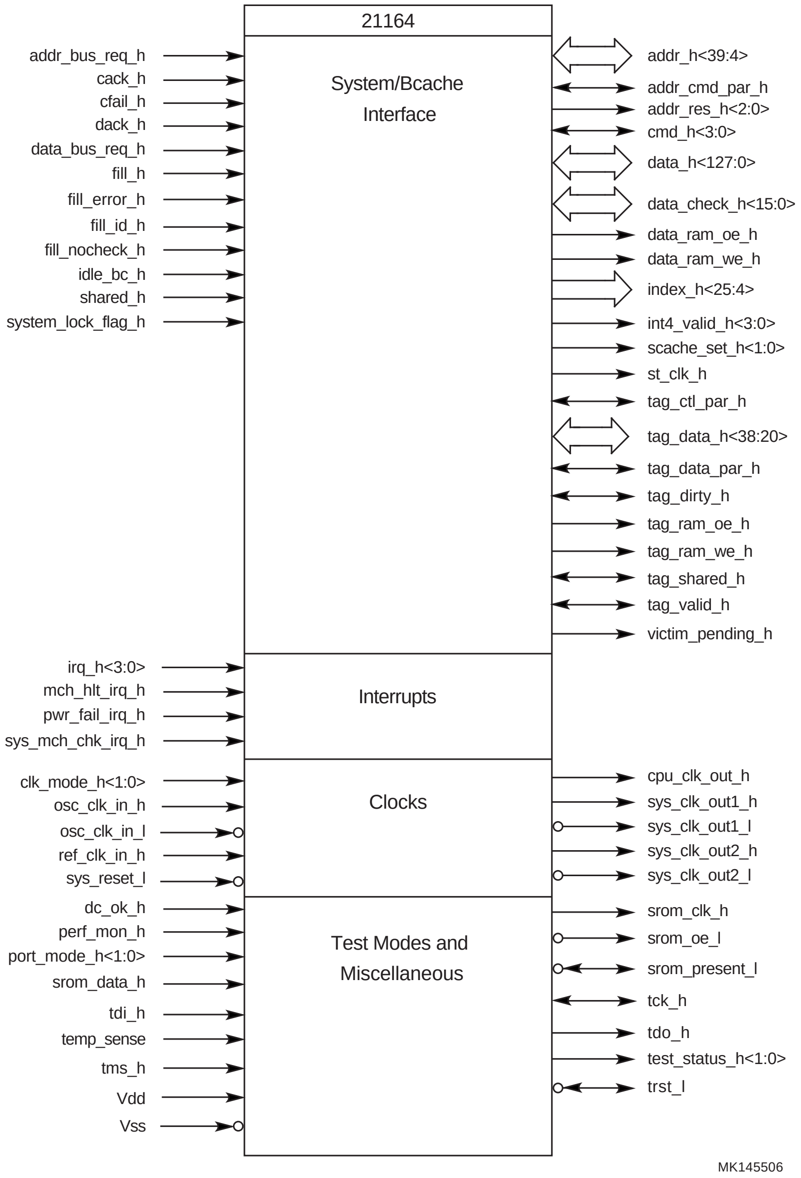
This chapter contains the 21164 microprocessor logic symbol and provides a list of signal names and their functions.

3.1 Alpha 21164 Microprocessor Logic Symbol

Figure 3–1 shows the logic symbol for the 21164 chip.

3.1 Alpha 21164 Microprocessor Logic Symbol

Figure 3–1 Alpha 21164 Microprocessor Logic Symbol



3.2 Alpha 21164 Signal Names and Functions

The 21164 is contained in a 499-pin IPGA package. Of these pins, 292 are used for functional signals. There are two spare (unused) signal pins. The remaining pins are used for power (104) and ground (101).

The following table defines the 21164 signal types referred to in this section:

Signal Type	Definition
B	Bidirectional
I	Input only
O	Output only

The remaining two tables describe the function of each 21164 external signal. Table 3–1 lists all signals in alphanumeric order. This table provides full signal descriptions. Table 3–2 lists signals by function and provides an abbreviated description.

Table 3–1 Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
addr_h<39:4>	B	36	Address bus. These bidirectional signals provide the address of the requested data or operation between the 21164 and the system. If bit 39 is asserted, then the reference is to noncached, I/O memory space.
addr_bus_req_h	I	1	Address bus request. The system interface uses this signal to gain control of the **addr_h<39:4>**, **addr_cmd_par_h**, and **cmd_h<3:0>** pins (see Figure 4–30).
addr_cmd_par_h	B	1	Address command parity. This is the odd parity bit on the current command and address buses. The 21164 takes a machine check if a parity error is detected. The system should do the same if it detects an error.

(continued on next page)

3.2 Alpha 21164 Signal Names and Functions

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
addr_res_h<1:0>	O	2	Address response bits <1> and <0>. For system commands, the 21164 uses these pins to indicate the state of the block in the Scache:

Bits	Command	Meaning
00	NOP	Nothing.
01	NOACK	Data not found or clean.
10	ACK/Scache	Data from Scache.
11	ACK/Bcache	Data from Bcache.

Signal	Type	Count	Description
addr_res_h<2>	O	1	Address response bit <2>. For system commands, the 21164 uses this pin to indicate if the command hits in the Scache or onchip load lock register.
cack_h	I	1	Command acknowledge. The system interface uses this signal to acknowledge any one of the commands driven by the 21164.
cfail_h	I	1	Command fail. This signal has two uses. It can be asserted during a cack cycle of a WRITE BLOCK LOCK command to indicate that the write operation is not successful. In this case, both **cack_h** and **cfail_h** are asserted together. It can also be asserted instead of **cack_h** to force an instruction fetch/decode unit (Ibox) timeout event. This causes the 21164 to do a partial reset and trap to the machine check (MCHK) PALcode entry point, which indicates a serious hardware error.
clk_mode_h<1:0>	I	2	Clock test mode. These signals specify a relationship between **osc_clk_in_h,l** and the CPU cycle time. These signals should be deasserted in normal operation mode.
cmd_h<3:0>	B	4	Command bus. These signals drive and receive the commands from the command bus. The following tables define the commands that can be driven on the **cmd_h<3:0>** bus by the 21164 or the system. For additional information, refer to Section 4.1.1.1.

(continued on next page)

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description

21164 Commands to System:

cmd_h <3:0>	Command	Meaning
0000	NOP	Nothing.
0001	LOCK	Lock register address.
0010	FETCH	The 21164 passes a FETCH instruction to the system.
0011	FETCH_M	The 21164 passes a FETCH_M instruction to the system.
0100	MEMORY BARRIER	MB instruction.
0101	SET DIRTY	Dirty bit set if shared bit is clear.
0110	WRITE BLOCK	Request to write a block.
0111	WRITE BLOCK LOCK	Request to write a block with lock.
1000	READ MISS0	Request for data.
1001	READ MISS1	Request for data.
1010	READ MISS MOD0	Request for data; modify intent.
1011	READ MISS MOD1	Request for data; modify intent.
1100	BCACHE VICTIM	Bcache victim should be removed.
1101	—	Reserved.
1110	READ MISS MOD STC0	Request for data, ST*x*_C data.
1111	READ MISS MOD STC1	Request for data, ST*x*_C data.

(continued on next page)

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
			System Commands to 21164:

cmd_h <3:0>	Command	Meaning
0000	NOP	Nothing.
0001	FLUSH	Remove block from caches; return dirty data.
0010	INVALIDATE	Invalidate the block from caches.
0011	SET SHARED	Block goes to the shared state.
0100	READ	Read a block.
0101	READ DIRTY	Read a block; set shared.
0111	READ DIRTY/INV	Read a block; invalidate.

Signal	Type	Count	Description
cpu_clk_out_h	O	1	CPU clock output. This signal is used for test purposes.
dack_h	I	1	Data acknowledge. The system interface uses this signal to control data transfer between the 21164 and the system.
data_h<127:0>	B	128	Data bus. These signals are used to move data between the 21164, the system, and the Bcache.
data_bus_req_h	I	1	Data bus request. If the 21164 samples this signal asserted on the rising edge of sysclk *n*, then the 21164 does not drive the data bus on the rising edge of sysclk *n*+1. Before asserting this signal, the system should assert **idle_bc_h** for the correct number of cycles. If the 21164 samples this signal deasserted on the rising edge of sysclk *n*, then the 21164 drives the data bus on the rising edge of sysclk *n*+1. For timing details, refer to Section 4.11.4.

(continued on next page)

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
data_check_h<15:0>	B	16	Data check. These signals set even byte parity or INT8 ECC for the current data cycle. Refer to Section 4.14.1 for information on the purpose of each **data_check_h** bit.
data_ram_oe_h	O	1	Data RAM output enable. This signal is asserted for Bcache read operations.
data_ram_we_h	O	1	Data RAM write-enable. This signal is asserted for any Bcache write operation. Refer to Section 5.3.5 for timing details.
dc_ok_h	I	1	dc voltage OK. Must be deasserted until dc voltage reaches proper operating level. After that, **dc_ok_h** is asserted.
fill_h	I	1	Fill warning. If the 21164 samples this signal asserted on the rising edge of sysclk *n*, then the 21164 provides the address indicated by **fill_id_h** to the Bcache on the rising edge of sysclk *n*+1. The Bcache begins to write in that sysclk. At the end of sysclk *n*+1, the 21164 waits for the next sysclk and then begins the write operation again if **dack_h** is not asserted. Refer to Section 4.11.3 for timing details.
fill_error_h	I	1	Fill error. If this signal is asserted during a fill from memory, it indicates to the 21164 that the system has detected an invalid address or hard error. The system still provides an apparently normal read sequence with correct ECC/parity though the data is not valid. The 21164 traps to the machine check (MCHK) PALcode entry point and indicates a serious hardware error. **fill_error_h** should be asserted when the data is returned. Each assertion produces a MCHK trap.
fill_id_h	I	1	Fill identification. Asserted with **fill_h** to indicate which register is used. The 21164 supports two outstanding load instructions. If this signal is asserted when the 21164 samples **fill_h** asserted, then the 21164 provides the address from miss register 1. If it is deasserted, then the address in miss register 0 is used for the read operation.
fill_nocheck_h	I	1	Fill checking off. If this signal is asserted, then the 21164 does not check the parity or ECC for the current data cycle on a fill.

(continued on next page)

3.2 Alpha 21164 Signal Names and Functions

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
idle_bc_h	I	1	Idle Bcache. When asserted, the 21164 finishes the current Bcache read or write operation but does not start a new read or write operation until the signal is deasserted. The system interface must assert this signal in time to idle the Bcache before fill data arrives.
index_h<25:4>	O	22	Index. These signals index the Bcache.
int4_valid_h<3:0>	O	4	INT4 data valid. During write operations to noncached space, these signals are used to indicate which INT4 bytes of data are valid. This is useful for noncached write operations that have been merged in the write buffer.

int4_valid_h<3:0>	Write Meaning
*xxx*1	**data_h<31:0>** valid
*xx*1*x*	**data_h<63:32>** valid
*x*1*xx*	**data_h<95:64>** valid
1*xxx*	**data_h<127:96>** valid

During read operations to noncached space, these signals indicate which INT8 bytes of a 32-byte block need to be read and returned to the processor. This is useful for read operations to noncached memory.

int4_valid_h<3:0>	Read Meaning
*xxx*1	**data_h<63:0>** valid
*xx*1*x*	**data_h<127:64>** valid
*x*1*xx*	**data_h<191:128>** valid
1*xxx*	**data_h<255:192>** valid

Note: For both read and write operations, multiple **int4_valid_h<3:0>** bits can be set simultaneously.

(continued on next page)

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
irq_h<3:0>	I	4	System interrupt requests. These signals have multiple modes of operation. During normal operation, these level-sensitive signals are used to signal interrupt requests. During initialization, these signals are used to set up the CPU cycle time divisor for **sys_clk_out1_h,l** as follows:

irq_h				
<3>	**<2>**	**<1>**	**<0>**	**Ratio**
Low	Low	High	High	3
Low	High	Low	Low	4
Low	High	Low	High	5
Low	High	High	Low	6
Low	High	High	High	7
High	Low	Low	Low	8
High	Low	Low	High	9
High	Low	High	Low	10
High	Low	High	High	11
High	High	Low	Low	12
High	High	Low	High	13
High	High	High	Low	14
High	High	High	High	15

Signal	Type	Count	Description
mch_hlt_irq_h	I	1	Machine halt interrupt request. This signal has multiple modes of operation. During initialization, this signal is used to set up **sys_clk_out2_h,l** delay (see Table 4–3). During normal operation, it is used to signal a halt request.
osc_clk_in_h **osc_clk_in_l**	I I	1 1	Oscillator clock inputs. These signals provide the differential clock input that is the fundamental timing of the 21164. These signals are driven at twice the desired internal clock frequency. (Under normal operating conditions the CPU cycle time is one-half the frequency of **osc_clk_in**.)

(continued on next page)

3.2 Alpha 21164 Signal Names and Functions

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
perf_mon_h	I	1	Performance monitor. This signal can be used as an input to the 21164 internal performance monitoring hardware from offchip events (such as bus activity). Refer to Section 5.1.27 for information on the PMCTR register.
port_mode_h<1:0>	I	2	Select test port interface modes (normal, manufacturing, and debug). For normal operation, both signals must be deasserted.
pwr_fail_irq_h	I	1	Power failure interrupt request. This signal has multiple modes of operation. During initialization, this signal is used to set up **sys_clk_out2_h,l** delay (see Table 4–3). During normal operation, this signal is used to signal a power failure.
ref_clk_in_h	I	1	Reference clock input. Optional. Used to synchronize the timing of multiple microprocessors to a single reference clock. If this signal is not used, it must be tied to **Vdd** for proper operation.
scache_set_h<1:0>	O	2	Secondary cache set. During a read miss request, these signals indicate the Scache set number that will be filled when the data is returned. This information can be used by the system to maintain a duplicate copy of the Scache tag store.
shared_h	I	1	Keep block status shared. For systems without a Bcache, when a WRITE BLOCK/NO VICTIM PENDING or WRITE BLOCK LOCK command is acknowledged, this pin can be used to keep the block status shared or private in the Scache.
srom_clk_h	O	1	Serial ROM clock. Supplies the clock that causes the SROM to advance to the next bit. The cycle time of this clock is 128 times the cycle time of the CPU clock.
srom_data_h	I	1	Serial ROM data. Input for the SROM.
srom_oe_l	O	1	Serial ROM output enable. Supplies the output enable to the SROM.
srom_present_l[1]	B	1	Serial ROM present. Indicates that SROM is present and ready to load the Icache.

[1]This signal is shown as bidirectional. However, for normal operation it is input only. The output function is used during manufacturing test and verification only.

(continued on next page)

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
st_clk_h	O	1	STRAM clock. Clock for Bcache synchronously timed RAMs (STRAMs). This signal is synchronous with **index_h<25:4>** during private read and write operations, and with **sys_clk_out1_h,l** during read and fill operations.
sys_clk_out1_h **sys_clk_out1_l**	O O	1 1	System clock outputs. Programmable system clock (**cpu_clk_out_h** divided by a value of 3 to 15) is used for board-level cache and system logic.
sys_clk_out2_h **sys_clk_out2_l**	O O	1 1	System clock outputs. A version of **sys_clk_out1_h,l** delayed by a programmable amount from 0 to 7 CPU cycles.
sys_mch_chk_irq_h	I	1	System machine check interrupt request. This signal has multiple modes of operation. During initialization, it is used to set up **sys_clk_out2_h,l** delay (see Table 4–3). During normal operation, it is used to signal a machine interrupt check request.
sys_reset_l	I	1	System reset. This signal protects the 21164 from damage during initial power-up. It must be asserted until **dc_ok_h** is asserted. After that, it is deasserted and the 21164 begins its reset sequence.
system_lock_flag_h	I	1	System lock flag. During fills, the 21164 logically ANDs the value of the system copy with its own copy to produce the true value of the lock flag.
tag_ctl_par_h	B	1	Tag control parity. This signal indicates odd parity for **tag_valid_h**, **tag_shared_h**, and **tag_dirty_h**. During fills, the system should drive the correct parity based on the state of the valid, shared, and dirty bits.
tag_data_h<38:20>	B	19	Bcache tag data bits. This bit range supports 1M-byte to 64M-byte Bcaches.
tag_data_par_h	B	1	Tag data parity bit. This signal indicates odd parity for **tag_data_h<38:20>**.
tag_dirty_h	B	1	Tag dirty state bit. During fills, the system should assert this signal if the 21164 request is a READ MISS MOD, and the shared bit is not asserted. Refer to Table 4–6 for information about Bcache protocol.
tag_ram_oe_h	O	1	Tag RAM output enable. This signal is asserted during any Bcache read operation.

(continued on next page)

3.2 Alpha 21164 Signal Names and Functions

Table 3–1 (Cont.) Alpha 21164 Signal Descriptions

Signal	Type	Count	Description
tag_ram_we_h	O	1	Tag RAM write-enable. This signal is asserted during any tag write operation. During the first CPU cycle of a write operation, the write pulse is deasserted. In the second and following CPU cycles of a write operation, the write pulse is asserted if the corresponding bit in the write pulse register is asserted. Bits **BC_WE_CTL<8:0>** control the shape of the pulse (see Section 5.3.5).
tag_shared_h	B	1	Tag shared bit. During fills, the system should drive this signal with the correct value to mark the cache block as shared. See Table 4–6 for information about Bcache protocol.
tag_valid_h	B	1	Tag valid bit. During fills, this signal is asserted to indicate that the block has valid data. See Table 4–6 for information about Bcache protocol.
tck_h	B	1	JTAG boundary scan clock.
tdi_h	I	1	JTAG serial boundary scan data-in signal.
tdo_h	O	1	JTAG serial boundary scan data-out signal.
temp_sense	I	1	Temperature sense. This signal is used to measure the die temperature and is for manufacturing use only. For normal operation, this signal must be left disconnected.
test_status_h<1:0>	O	2	Icache test status. These signals are used for manufacturing test purposes only to extract Icache test status information from the chip. **test_status_h<0>** is asserted if ICSR<39> is true, on Ibox timeout, or remains asserted if the Icache built-in self-test (BiSt) fails. Also, **test_status_h<0>** outputs the value written by PALcode to **test_status_h<1>** through IPR access. For additional information, refer to Section 12.2.2.
tms_h	I	1	JTAG test mode select signal.
trst_l[1]	B	1	JTAG test access port (TAP) reset signal.
victim_pending_h	O	1	Victim pending. When asserted, this signal indicates that the current read miss has generated a victim.

[1]This signal is shown as bidirectional. However, for normal operation it is input only. The output function is used during manufacturing test and verification only.

3.2 Alpha 21164 Signal Names and Functions

Table 3–2 lists signals by function and provides an abbreviated description.

Table 3–2 Alpha 21164 Signal Descriptions by Function

Signal	Type	Count	Description
Clocks			
clk_mode_h<1:0>	I	2	Clock test mode.
cpu_clk_out_h	O	1	CPU clock output.
osc_clk_in_h,l	I	2	Oscillator clock inputs.
ref_clk_in_h	I	1	Reference clock input.
st_clk_h	O	1	Bcache STRAM clock output.
sys_clk_out1_h,l	O	2	System clock outputs.
sys_clk_out2_h,l	O	2	System clock outputs.
sys_reset_l	I	1	System reset.
Bcache			
data_h<127:0>	B	128	Data bus.
data_check_h<15:0>	B	16	Data check.
data_ram_oe_h	O	1	Data RAM output enable.
data_ram_we_h	O	1	Data RAM write-enable.
index_h<25:4>	O	22	Index.
tag_ctl_par_h	B	1	Tag control parity.
tag_data_h<38:20>	B	19	Bcache tag data bits.
tag_data_par_h	B	1	Tag data parity bit.
tag_dirty_h	B	1	Tag dirty state bit.
tag_ram_oe_h	O	1	Tag RAM output enable.
tag_ram_we_h	O	1	Tag RAM write-enable.
tag_shared_h	B	1	Tag shared bit.
tag_valid_h	B	1	Tag valid bit.

(continued on next page)

Table 3–2 (Cont.) Alpha 21164 Signal Descriptions by Function

Signal	Type	Count	Description
System Interface			
addr_h<39:4>	B	36	Address bus.
addr_bus_req_h	I	1	Address bus request.
addr_cmd_par_h	B	1	Address command parity.
addr_res_h<2:0>	O	3	Address response.
cack_h	I	1	Command acknowledge.
cfail_h	I	1	Command fail.
cmd_h<3:0>	B	4	Command bus.
dack_h	I	1	Data acknowledge.
data_bus_req_h	I	1	Data bus request.
fill_h	I	1	Fill warning.
fill_error_h	I	1	Fill error.
fill_id_h	I	1	Fill identification.
fill_nocheck_h	I	1	Fill checking off.
idle_bc_h	I	1	Idle Bcache.
int4_valid_h<3:0>	O	4	INT4 data valid.
scache_set_h<1:0>	O	2	Secondary cache set.
shared_h	I	1	Keep block status shared.
system_lock_flag_h	I	1	System lock flag.
victim_pending_h	O	1	Victim pending.
Interrupts			
irq_h<3:0>	I	4	System interrupt requests.
mch_hlt_irq_h	I	1	Machine halt interrupt request.
pwr_fail_irq_h	I	1	Power failure interrupt request.
sys_mch_chk_irq_h	I	1	System machine check interrupt request.

(continued on next page)

Table 3–2 (Cont.) Alpha 21164 Signal Descriptions by Function

Signal	Type	Count	Description
Test Modes and Miscellaneous			
dc_ok_h	I	1	dc voltage OK.
perf_mon_h	I	1	Performance monitor.
port_mode_h<1:0>	I	2	Select test port interface modes (normal, manufacturing, and debug).
srom_clk_h	O	1	Serial ROM clock.
srom_data_h	I	1	Serial ROM data.
srom_oe_l	O	1	Serial ROM output enable.
srom_present_l[1]	B	1	Serial ROM present.
tck_h	B	1	JTAG boundary scan clock.
tdi_h	I	1	JTAG serial boundary scan data in.
tdo_h	O	1	JTAG serial boundary scan data out.
temp_sense	I	1	Temperature sense.
test_status_h<1:0>	O	2	Icache test status.
tms_h	I	1	JTAG test mode select.
trst_l[1]	B	1	JTAG test access port (TAP) reset.

[1]This signal is shown as bidirectional. However, for normal operation is is input only. The output function is used during manufacturing test and verification only.

4

Clocks, Cache, and External Interface Functional Description

This chapter describes the 21164 microprocessor external interface, which includes the backup cache (Bcache) and system interfaces. It also describes the clock circuitry, locks, interrupt signals, and ECC/parity generation. It is organized as follows:

- Introduction to the external interface
- Clocks
- Physical address considerations
- Bcache structure and operation
- Cache coherency
- Locks mechanisms
- 21164-to-Bcache transactions
- 21164-initiated system transactions
- System-initiated transactions
- Data bus and command/address bus contention
- 21164 interface restrictions
- 21164/system race conditions
- Data integrity, Bcache errors, and command/address errors
- Interrupts

Chapter 3 lists and defines all 21164 hardware interface signal pins. Chapter 9 describes the 21164 hardware interface electrical requirements.

4.1 Introduction to the External Interface

A 21164-based system can be divided into three major sections:

- 21164 microprocessor
- Optional external Bcache
- System interface logic
  - Optional duplicate tag store
  - Optional lock register
  - Optional victim buffers

The 21164 external interface is flexible and mandates few design rules, allowing a wide range of prospective systems. The interface includes a 128-bit bidirectional data bus, a 36-bit bidirectional address bus, and several control signals.

Read and write speeds of the optional Bcache array can be programmed by means of register bits. Read and write speeds are independent of each other and the system interface clock frequency.

The cache system supports a selectable 32-byte or 64-byte block size.

Figure 4–1 shows a simplified view of the external interface. The function and purpose of each signal is described in Chapter 3.

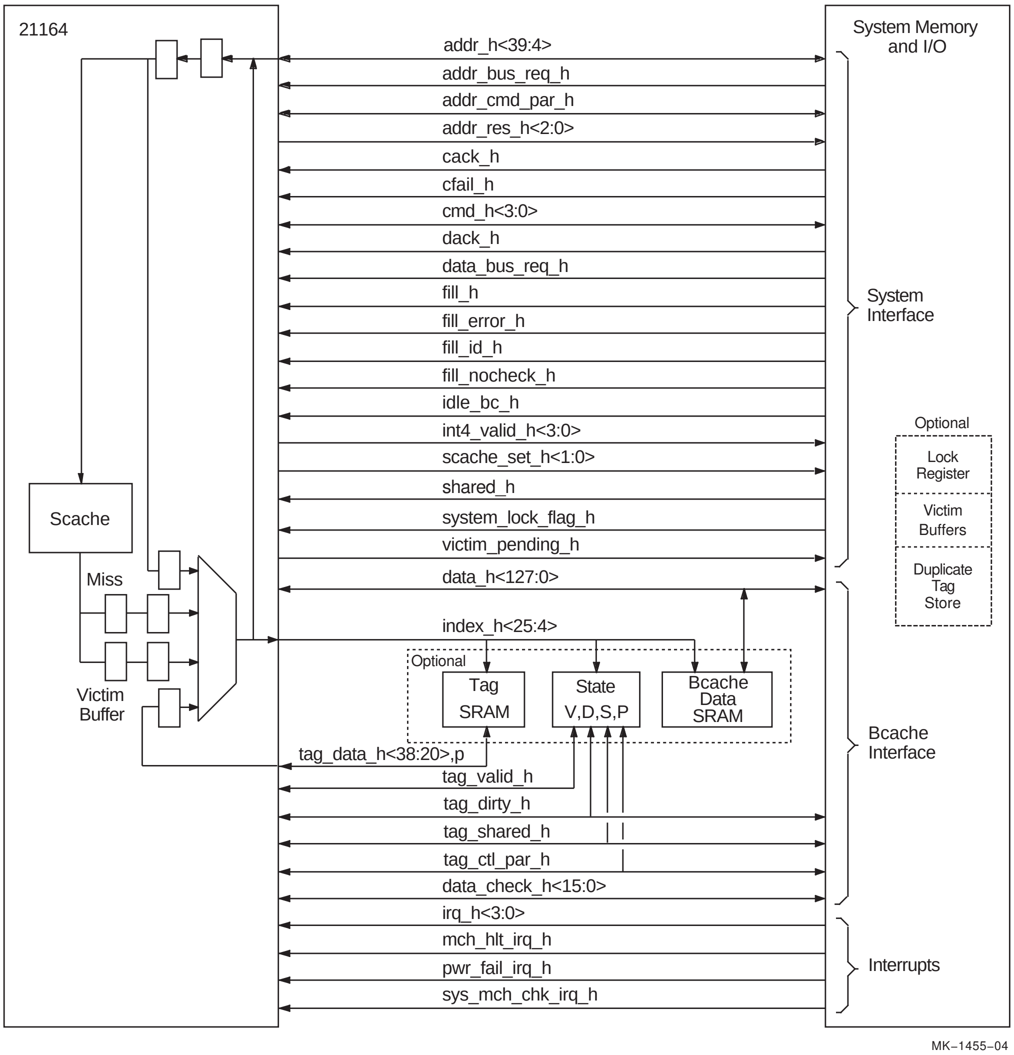
4.1.1 System Interface

This section describes the system or external bus interface. The system interface is made up of bidirectional address and command buses, a data bus that is shared with the Bcache interface, and several control signals.

The system interface is under the control of the bus interface unit (BIU) in the Cbox. The system interface is a 128-bit bidirectional data bus.

The cycle time of the system interface is programmable to speeds of 3 to 15 times the CPU cycle time (sysclk ratio). All system interface signals are driven or sampled by the 21164 on the rising edge of signal **sys_clk_out1_h**. In this chapter, this edge is sometimes referred to as ‘‘sysclk.’’ Precisely when interface signals rise and fall does not matter as long as they meet the setup and hold times specified in Chapter 9.

Figure 4–1 Alpha 21164 System/Bcache Interface



4.1 Introduction to the External Interface

4.1.1.1 Commands and Addresses

The 21164 can take up to two commands from the system at a time. The Scache or Bcache or both are probed to determine what must be done with the command.

- If nothing is to be done, the 21164 acknowledges receiving the command.
- If a Bcache read, set shared, or invalidate operation is required, the 21164 performs the task as soon as the Bcache becomes free. The 21164 acknowledges receiving the command at the start of the Bcache transaction.

There are two miss and two victim buffers in the BIU. They can hold one or two miss addresses and one or two Scache victim addresses or up to two shared write operations at a time.

- A miss occurs when the 21164 searches its caches but does not find the addressed block. The 21164 can queue two misses to the system.
- An Scache victim occurs when the 21164 deallocates a dirty block from the Scache.

4.1.2 Bcache Interface

The 21164 includes an interface and control for an optional backup cache (Bcache). The Bcache interface is made up of the following:

- A 128-bit data bus (which it shares with the system interface)
- Index address bits (**index_h<25:4>**)
- Tag and state bits for determining hit and coherence
- SRAM output and write control signals

4.2 Clocks

The 21164 develops three clock signals that are available at output pins:

Signal	Description
cpu_clk_out_h	A 21164 internal clock that may or may not drive the system clock.
sys_clk_out1_h,l	A clock of programmable speed supplied to the external interface.
sys_clk_out2_h,l	A delayed copy of **sys_clk_out1_h,l**. The delay is programmable and is an integer number of **cpu_clk_out_h** periods.

The 21164 may use **ref_clk_in_h** as a reference clock when generating **sys_clk_out1_h,l** and **sys_clk_out2_h,l**. The behavior of the programmable clocks during the reset sequence is described in Section 7.1.

4.2.1 CPU Clock

The 21164 uses the differential input clock lines **osc_clk_in_h,l** as a source to generate its CPU clock. The input signals **clk_mode_h<1:0>** control generation of the CPU clock as listed in Table 4–1 and as shown in Figure 4–2.

Table 4–1 CPU Clock Generation Control

Mode	clk_mode_h<1:0>		Divisor	Description
Normal	0	0	2[1]	Usual operation—CPU clock frequency is ½ input frequency.
Chip test	0	1	1	CPU clock frequency is the same as the input clock frequency to accommodate chip testers.
Module test	1	0	4[1]	CPU clock frequency is ¼ input frequency to accommodate module testers.
Reset	1	1	—	Initializes CPU clock, allowing system clock to be synchronized to a stable reference clock.

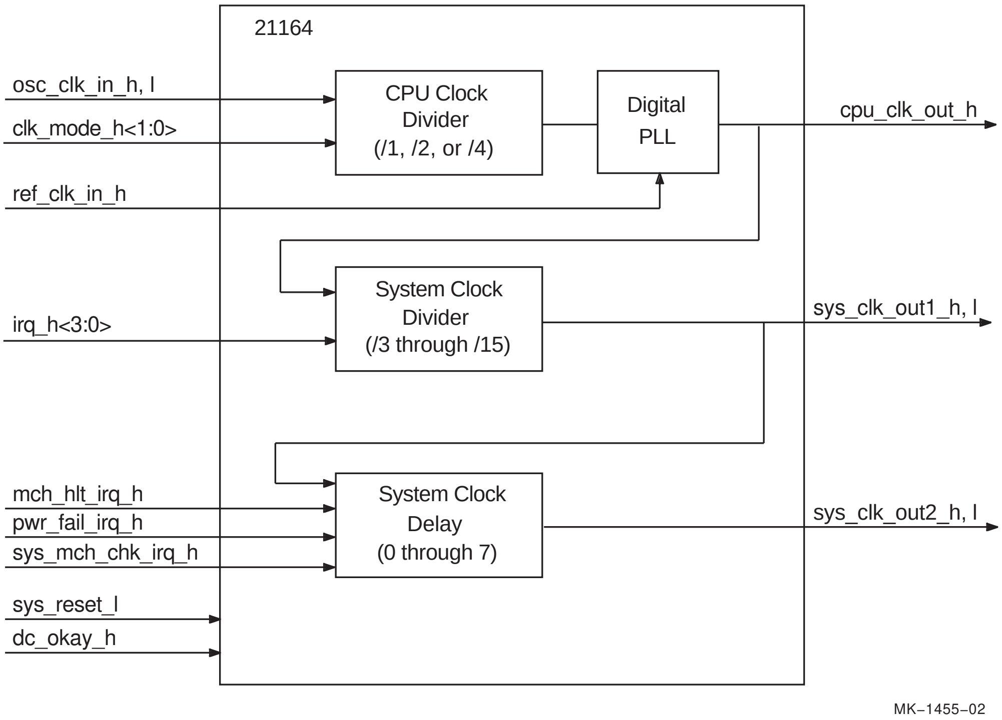
[1]Divide by 2 or 4 should be used to obtain the best internal clock.

Caution

A clock source should always be provided on **osc_clk_in_h,l** when signal **dc_ok_h** is asserted.

4.2 Clocks

Figure 4–2 Clock Signals and Functions



4.2.2 System Clock

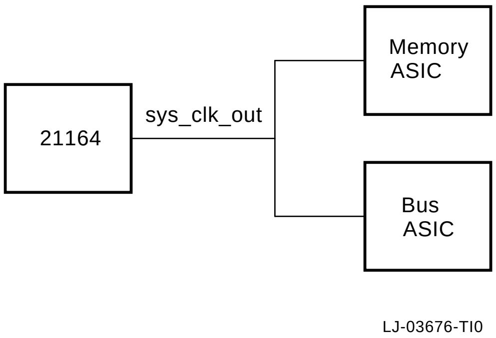
The CPU clock is the source clock used to generate the system clock **sys_clk_out1_h,l**. The system clock divider controls the frequency of **sys_clk_out1_h,l**. The divisor, 3 to 15, is obtained from the four interrupt lines **irq_h<3:0>** at power-up as listed in Table 4–2. The system clock frequency is determined by dividing the ratio into the CPU clock frequency. Refer to Section 7.2 for information on sysclk behavior during reset.

Table 4–2 System Clock Divisor

irq_h<3>	irq_h<2>	irq_h<1>	irq_h<0>	Ratio
Low	Low	High	High	3
Low	High	Low	Low	4
Low	High	Low	High	5
Low	High	High	Low	6
Low	High	High	High	7
High	Low	Low	Low	8
High	Low	Low	High	9
High	Low	High	Low	10
High	Low	High	High	11
High	High	Low	Low	12
High	High	Low	High	13
High	High	High	Low	14
High	High	High	High	15

Figure 4–3 shows the 21164 driving the system clock on a uniprocessor system.

Figure 4–3 Alpha 21164 Uniprocessor Clock



4.2.3 Delayed System Clock

The system clock **sys_clk_out1_h,l** is the source clock for the delayed system clock **sys_clk_out2_h,l**. These clock signals provide flexible timing for system use. The delay unit, 0 to 7, is obtained from the three interrupt signals: **mch_hlt_irq_h**, **pwr_fail_irq_h**, and **sys_mch_chk_irq_h** at power-up, as listed in Table 4–3. The output of this programmable divider is symmetric if the divisor is even. The output is asymmetric if the divisor is odd. When the divisor is odd, the clock is high for an extra cycle. Refer to Section 7.2 for information on sysclk behavior during reset.

Table 4–3 System Clock Delay

sys_mch_chk_irq_h	pwr_fail_irq_h	mch_hlt_irq_h	Delay Cycles
Low	Low	Low	0
Low	Low	High	1
Low	High	Low	2
Low	High	High	3
High	Low	Low	4
High	Low	High	5
High	High	Low	6
High	High	High	7

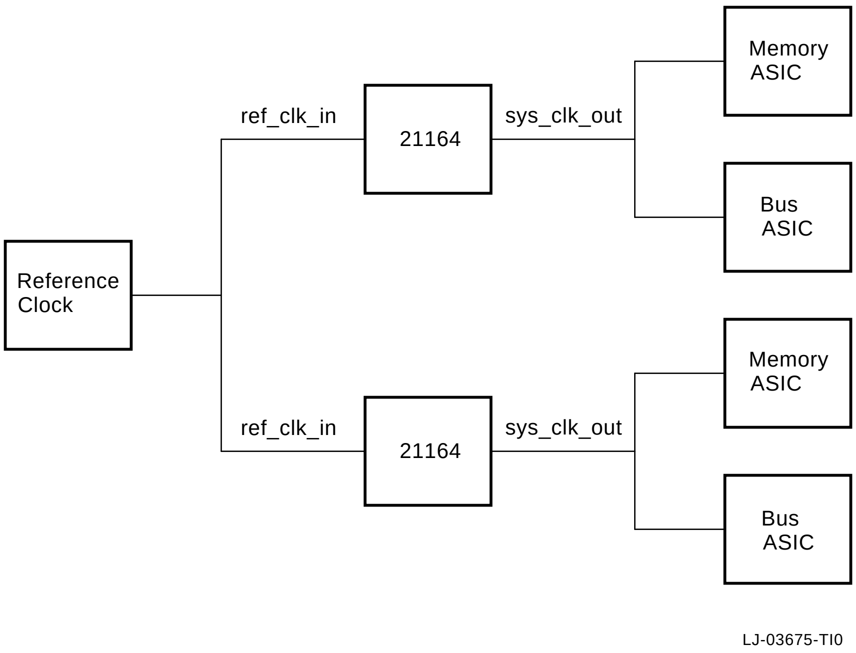
4.2.4 Reference Clock

The 21164 provides a reference clock input so that other CPUs and system devices can be synchronized in multiprocessor systems. If a clock is asserted on signal **ref_clk_in_h**, then the **sys_clk_out1_h,l** signals are synchronized to that reference clock. The reference clock input should be connected to **Vdd** if the input is not to be used.

The 21164 synchronizes the **sys_clk_out1_h** frequency with the **ref_clk_in_h** signal by means of a digital phase-locked loop (DPLL). The DPLL does not lock the two frequencies, but rather, creates a window. To accomplish this, the frequency of signal **sys_clk_out1** must be slightly higher, but no greater than 0.35% higher, than that of signal **ref_clk_in_h**. This causes the rising edge of **sys_clk_out1** to drift back toward the rising edge of **ref_clk_in_h**. The 21164 detects when the edges meet and stalls the internal clock generator for one **osc_clk_in** cycle. This moves the rising edge of **sys_clk_out1** back in front of **ref_clk_in_h**.

Figure 4–4 shows a multiprocessor 21164 system synchronized to a reference clock.

Figure 4–4 Alpha 21164 Reference Clock for Multiprocessor Systems



4.2.4.1 Reference Clock Examples

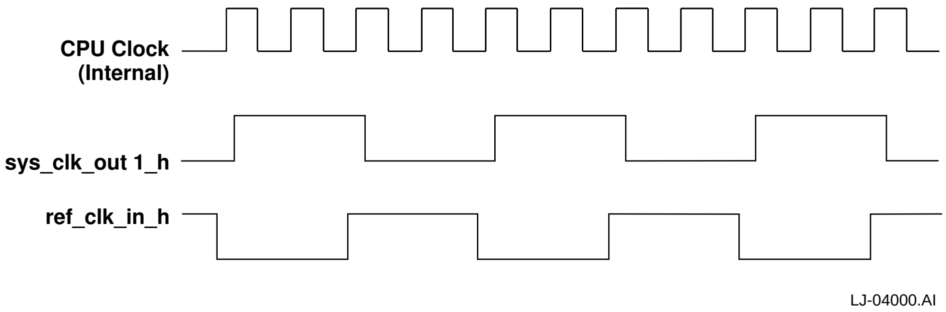
This section contains example calculations of setting time in systems that use the DPLL for synchronization.

After **sys_clk_out1_h,l** has stabilized (20 cycles after **irq_h<3:0>** have settled) there will be a delay before **sys_clk_out1_h,l** comes into lock with **ref_clk_in_h**. The two cases for this event are described in Section 4.2.4.1.1 and Section 4.2.4.1.2.

4.2.4.1.1 Case 1: ref_clk_in_h Initially Sampled Low by DPLL

When the DPLL initially samples **ref_clk_in_h** in the low state, as shown in Figure 4–5, it slips its internal cycle repeatedly until it samples **ref_clk_in_h** in the high state. After it samples **ref_clk_in_h** in the high state, the DPLL stays in lock mode.

Figure 4–5 ref_clk_in_h Initially Sampled Low



Note

The timing diagram shows a **sys_clk_out1_h,l** ratio of 4.

The worst case (slowest) maximum rate at which the DPLL will slip its internal cycle (the frequency of phase slips) is calculated from the lock range specification of 0.35%. In effect, an average of 0.35% period is added to each **sys_clk_out1_h,l** period until lock mode is reached.

$$SettlingTime = \frac{RefClockLowRatio * RefClockPeriod}{0.0035}$$

Note

The reference clock low ratio equals the portion of the reference clock period that **ref_clk_in_h** is low.

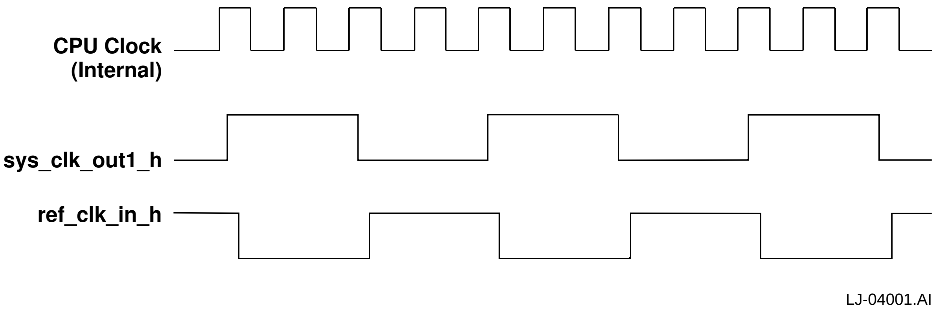
Assuming the worst case **ref_clk_in_h** duty cycle is 60/40 to 40/60:

$$SettlingTime = \frac{0.6 * RefClockPeriod}{0.0035} = 171 * RefClockPeriod$$

Depending upon the **sys_clk_out1_h,l** ratio, the DPLL may come into lock much more quickly. The DPLL may insert phase slips more frequently at smaller **sys_clk_out1_h,l** ratios.

4.2.4.1.2 Case 2: ref_clk_in_h Initially Sampled High by DPLL When the DPLL initially samples **ref_clk_in_h** in the high state, as shown in Figure 4–6, it will not slip its internal cycle until it samples **ref_clk_in_h** in the low state. After it samples **ref_clk_in_h** in the low state, the DPLL stays in lock mode.

Figure 4–6 ref_clk_in_h Initially Sampled High



The rate at which **sys_clk_out1_h,l** gains on **ref_clk_in_h** depends on the difference in frequency of the two signals. Assuming that:

> **ref_clk_in_h** is nominally selected to run 0.175% slower than **sys_clk_out1_h,l** (in the center of the specified lock range),
>
> and that worst case deviation of 200 PPM from the specified frequency for **ref_clk_in_h** and **osc_clk_in_h,l**,

Then the worst case (smallest) frequency difference is calculated to be,
$0.00175 - 200PPM - 200PPM = 0.00135 = 0.135\%$

$$SettlingTime = \frac{RefClockHighRatio * RefClockPeriod}{0.00135}$$

Note

The reference clock high ratio equals the portion of the **ref_clk_in_h** period that **ref_clk_in_h** is high.

4.2 Clocks

Assuming the worst case **ref_clk_in_h** duty cycle is 60/40 to 40/60:

$$SettlingTime = \frac{0.6 * RefClockPeriod}{0.00135} = 444 * RefClockPeriod$$

4.3 Physical Address Considerations

This section lists and describes the physical address regions. Cache and data wrapping characteristics of physical addresses are also described.

4.3.1 Physical Address Regions

Physical memory of the 21164 is divided into three regions:

1. The first region is the first half of the physical address space. It is treated by the 21164 as memory-like.
2. The second region is the second half of the physical address space except for a 1M-byte region reserved for Cbox IPRs. It is treated by the 21164 as noncachable.
3. The third region is the 1M-byte region reserved for Cbox IPRs.

In the first region, write invalidate caching, write merging, and load merging are all permitted. All 21164 accesses in this region are 32- or 64-byte depending on the programmable block size.

The 21164 does not cache data accessed in the second and third region of the physical address space; 21164 read accesses in these regions are always INT32 requests. Load merging is permitted, but the request includes a mask to tell the system environment which INT8s are accessed. Write merging is permitted. Write accesses are INT32 requests with a mask indicating which INT4s are actually modified. The 21164 never writes more than 32 bytes at a time in noncached space.

The 21164 does not broadcast accesses to the Cbox IPR region if they map to a Cbox IPR. Accesses in this region, that are not to a defined Cbox IPR, produce UNDEFINED results. The system should not probe this region.

Table 4–4 shows the 21164 physical memory regions.

Table 4–4 Physical Memory Regions

Region	Address Range	Description
Memory-like	00 0000 0000– 7F FFFF $FFFF_{16}$	Write invalidate cached, load, and store merging allowed.
Noncacheable	80 0000 0000– FF FFEF $FFFF_{16}$	Not cached, load merging limited.
IPR region	FF FFF0 0000– FF FFFF $FFFF_{16}$	Accesses do not appear on the interface unless an undefined location is accessed (which produces UNDEFINED results).

4.3.2 Data Wrapping

The 21164 requires that wrapped read operations be performed on INT16 boundaries. READ, READ DIRTY, and FLUSH commands are all wrapped on INT16 boundaries as described here. The valid wrap orders for 64-byte blocks are selected by **addr_h<5:4>**. They are:

0, 1, 2, 3
1, 0, 3, 2
2, 3, 0, 1
3, 2, 1, 0

For 32-byte blocks, the valid wrap orders are selected by **addr_h<4>**. They are:

0, 1
1, 0

Similarly, when the system interface supplies a command that returns data from the 21164 caches, the values that the system drives on **addr_h<5:4>** determine the order in which data is supplied by the 21164.

WRITE BLOCK and WRITE BLOCK LOCK commands from the 21164 are not wrapped. They always write INT16 0, 1, 2, and 3. BCACHE VICTIM commands provide the data with the same wrap order as the read miss that produced them.

4.3.3 Noncached Read Operations

Read operations to physical addresses that have **addr_h<39>** asserted are not cached in the Dcache, Scache, or Bcache. They are merged like other read operations in the miss address file (MAF). To prevent several read operations to noncached memory from being merged into a single 32-byte bus request, software must insert memory barrier (MB) instructions or set MAF_MODE IPR bit (IO_NMERGE). The MAF merges as many Dstream read operations together as it can and sends the request to the BIU through the Scache.

Rather than merging two 32-byte requests into a single 64-byte request, the BIU requests a READ MISS from the system. Signals **int4_valid_h<3:0>** indicate which of the four quadwords are being requested by software. The system should return the fill data to the 21164 as usual. The 21164 does not write the Dcache, Scache, or Bcache with the fill data. The requested data is written in the register file or Icache.

Note

A special case using **int4_valid_h<3:0>** occurs during an Icache fill. In this case the entire returned block is valid although **int4_valid_h<3:0>** indicates zero.

4.3.4 Noncached Write Operations

Write operations to physical addresses that have **addr_h<39>** asserted are not written to any of the caches. These write operations are merged in the write buffer before being sent to the system. If software does not want write operations to merge, it must insert MB or WMB instructions between them.

When the write buffer decides to write data to noncached memory, the BIU requests a WRITE BLOCK. During each data cycle, **int4_valid_h<3:0>** indicates which INT4s within the INT16 are valid.

4.4 Bcache Structure

The 21164 supports a 1M-byte, 2M-byte, . . . , 32M-byte and 64M-byte Bcache. The size is under program control and is specified by BC_CONF<2:0> (BC_SIZE<2:0>).

The Bcache block size may consist of 32-byte or 64-byte blocks. The Scache also supports either 32-byte or 64-byte blocks. The block size must be the same for both and is selected using SC_CTL<SC_BLK_SIZE>.

Industry-standard static RAMs (SRAMs) may be connected to the 21164 without many extra components although fanout buffers may be required for the index lines. The SRAMs are directly controlled by the 21164, and the Bcache data lines are connected to the 21164 data bus.

The 21164 partitions physical address (**addr_h<39:5>**) into an index field and a tag field. The 21164 presents **index_h<25:4>** and **tag_data_h<38:20>** to the Bcache interface. The tag size required is Bcache_size/block_size.

The system designer uses the signal lines needed for a particular size Bcache. For example the smallest Bcache (1 MB) needs **index_h<19:4>** to address the cache block while the tag field would be **tag_data_h<38:20>**.

Only those bits that are actually needed for the amount of cached system main memory need to be stored in the Bcache tag, although the 21164 uses all the relevant tag address bits for that Bcache size on its tag compare. A larger Bcache uses more index bits and fewer tag address bits.

The CPU data bus is 16 bytes wide (128 bits) and thus each Bcache transaction requires two data cycles for a 32-byte block or four data cycles for a 64-byte block.

4.4.1 Duplicate Tag Store

In systems that have a Bcache, it is possible to build a full copy of the Bcache tag store. This data can then be used to filter requests coming off the system bus to the 21164.

In systems without a Bcache it is possible to build a full or partial copy of the Scache tag store and to model the contents of the Scache victim buffers.

4.4 Bcache Structure

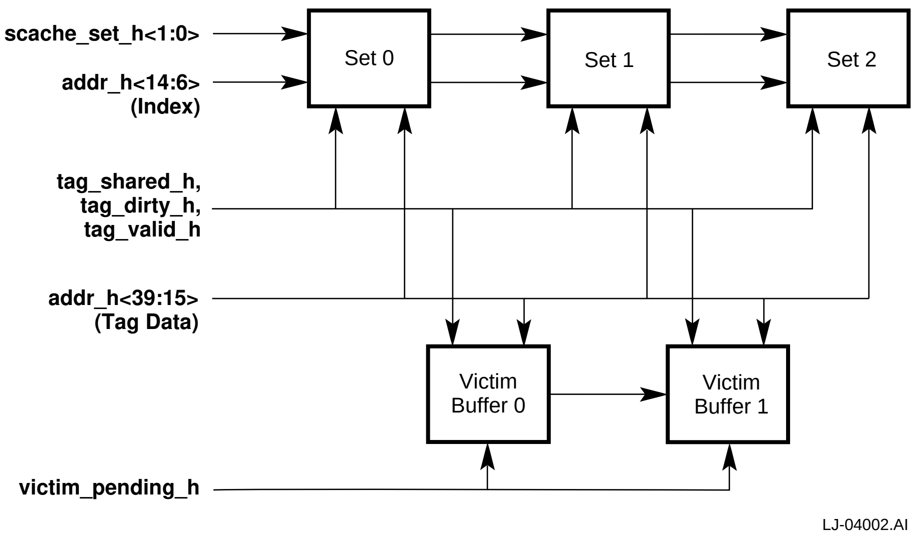
4.4.1.1 Full Duplicate Tag Store

The complete Bcache duplicate tag store would contain an entry for each Bcache block and each victim buffer. Each entry would contain state bits for the VALID, SHARED, and DIR TY status bits along with part or all of **addr_h<38:20>** for a Bcache block. The part of **addr_h<38:20>** stored in an entry depends upon the size of the Bcache.

In a system without a Bcache a full Scache duplicate tag store may be maintained. The full Scache duplicate tag store should contain three sets of 512 entries—one for each of the three Scache sets. It should also have two entries for the two Scache victim buffers. Signal **victim_pending_h** is used to indicate that the current READ command displaced a dirty block from the Scache, **scache_set_h<1:0>**, into the Scache victim buffer. The Scache duplicate tag store should be updated accordingly.

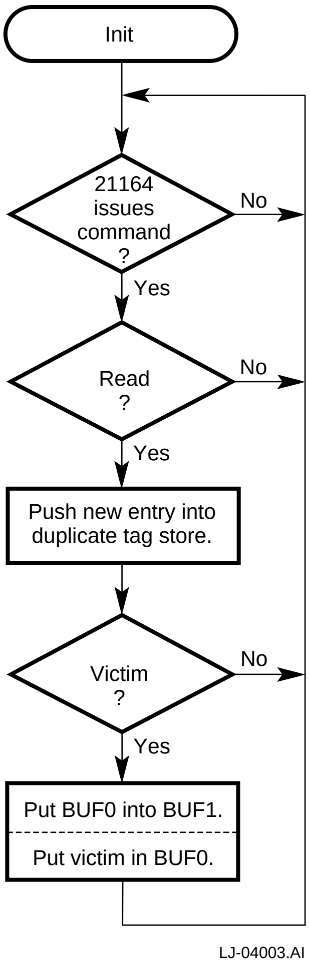
Figure 4–7 is a simplified diagram showing the signal lines of interest.

Figure 4–7 Full Scache Duplicate Tag Store



The system should use the algorithm shown in Figure 4–8 to maintain the duplicate tag store.

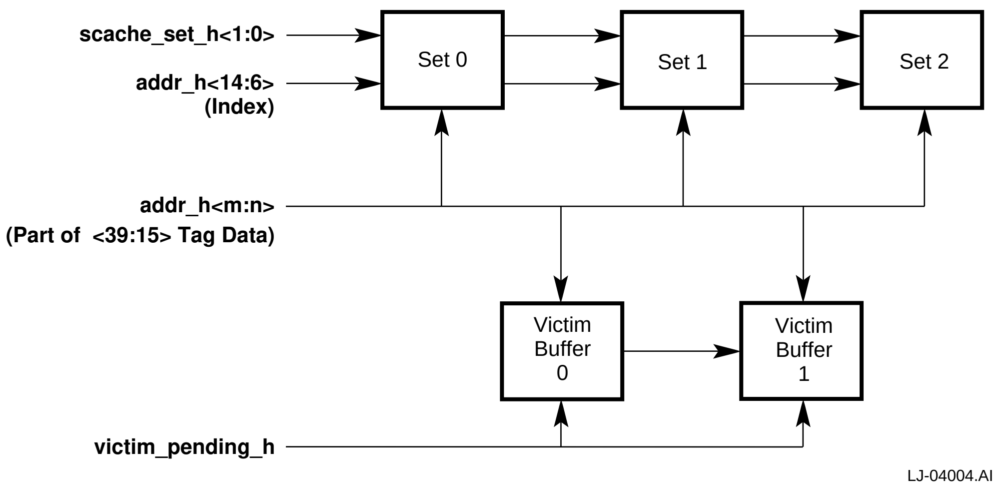
Figure 4–8 Duplicate Tag Store Algorithm



4.4.1.2 Partial Scache Duplicate Tag Store

System designers may also choose to build a partial Scache duplicate tag store such as that shown in Figure 4–9. This store contains one or more bits of tag data for each block in the Scache, and for the two victim buffers inside 21164. If a system bus transaction hits in the partial duplicate tag store, then the block may be in the Scache. If a system bus transaction misses in the partial duplicate tag store, then the block is not in the Scache. Signal **victim_pending_h** is used to indicate that the current READ command displaced a dirty block from the Scache, **scache_set_h<1:0>**, into the Scache victim buffer. The Scache duplicate tag store should be updated acordingly.

Figure 4–9 Partial Scache Duplicate Tag Store



4.4.2 Bcache Victim Buffers

A Bcache victim is generated when the 21164 deallocates a dirty block from the Bcache. Each time a Bcache victim is produced, the 21164 asserts **victim_pending_h** and stops reading the Bcache until the system takes the current victim. Then Bcache transactions resume.

External logic may help improve system performance by implementing any number of victim buffers that act as temporary storage that can be written faster and with lower latency than system memory. The victim buffers hold Bcache victims and enable the Bcache location to be filled with data from the desired address. Data in the victim buffers will be written to memory at a later time. This action reduces the time that the 21164 is waiting for data.

4.5 Systems Without a Bcache

Systems that do not employ a Bcache should leave the bidirectional signals **tag_data_par_h**, **tag_dirty_h**, **tag_valid_h**, **tag_shared_h**, and **tag_data_h<38:20>** disconnected. Pull-down structures within the 21164 prevent these signals from attaining undefined logic levels.

In systems with no Bcache, the Scache block size must be set to 64 bytes.

In systems with no Bcache, signal **idle_bc_h** is not required and should be permanently deasserted.

4.6 Cache Coherency

Cache coherency is a concern for single and multiprocessor 21164-based systems as there may be several caches on a processor module and several more in multiprocessor systems.

The system hardware designer need not be concerned about Icache and Dcache coherency. Coherency of the Icache is a software concern—it is flushed with an IMB (PALcode) instruction. The 21164 maintains coherency between the Dcache and the Scache.

If the system does not have a Bcache, the system designer must create mechanisms in the system interface logic to support cache coherency between the Scache, main memory, and other caches in the system.

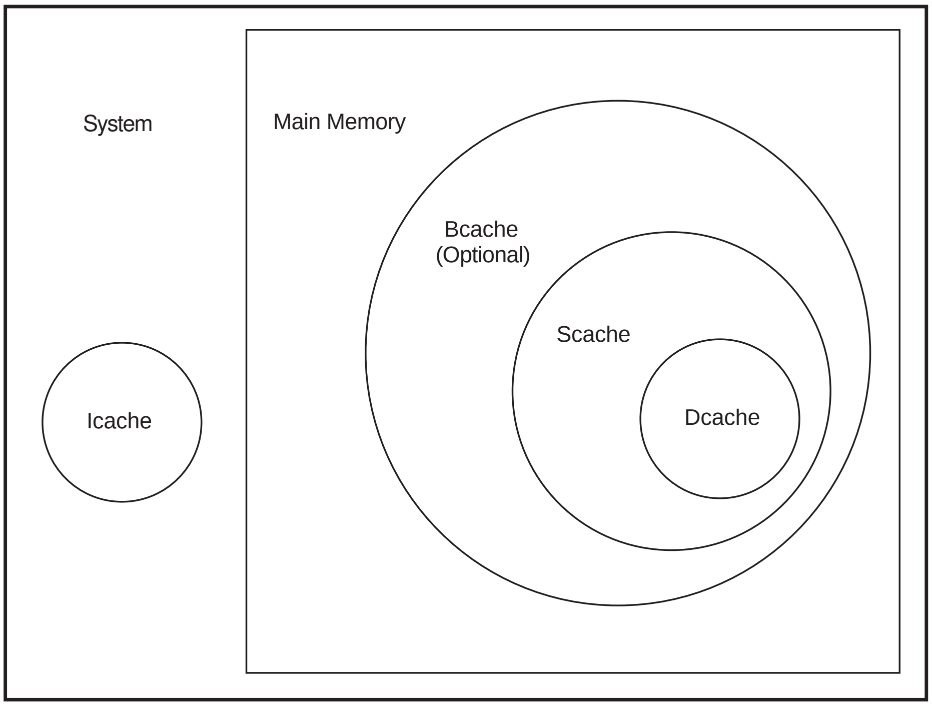
If the system has a Bcache, the 21164 maintains cache coherency between the Scache and the Bcache. The Scache is a subset of the Bcache. In this case the designer must create mechanisms in the system interface logic to support cache coherency between the Bcache, main memory, and other caches in the system.

4.6.1 Cache Coherency Basics

The 21164 systems maintain the cache coherency and hierarchy shown in Figure 4–10.

4.6 Cache Coherency

Figure 4–10 Cache Subset Hierarchy



The following tasks must be performed to maintain cache coherency:

- The Cbox in the 21164 maintains coherency in the Dcache and keeps it as a subset of the Scache.
- If an optional Bcache is present, then the 21164 maintains the Scache as a subset of the Bcache. The Scache is set-associative but is kept a subset of the larger externally implemented direct-mapped Bcache.
- System logic must help the 21164 to keep the Bcache coherent with main memory and other caches in the system.
- The Icache is not a subset of any cache and also is not kept coherent with the memory system.

The 21164 requires the system to allow only one change to a block at a time. This means that if the 21164 gains the bus to read or write a block, no other node on the bus should be allowed to access that block until the data has been moved.

4.6 Cache Coherency

The 21164 provides hardware mechanisms to support several cache coherency protocols. The protocols can be separated into two classes: write invalidate cache coherency protocol and flush cache coherency protocol.

Write Invalidate Cache Coherency Protocol

The write invalidate cache coherency protocol is best suited for shared memory multiprocessors.

The write invalidate protocol allows for shared data in the cache. If a Bcache (optional) is used, then a duplicate tag store is required. If a Bcache is not used, the duplicate tag store is not required but the module designer may include an Scache duplicate tag store.

Requiring the duplicate tag store if there is a Bcache allows the 21164 to process system commands in the Bcache without probing to see if the block is present (system logic knows the block is present). This results in higher performance for these transactions.

If a Bcache is not used, the module designer may include an Scache duplicate tag store to improve system performance.

Flush Cache Coherency Protocol

This protocol is best suited for low-cost single-processor systems. It is typically used by an I/O subsystem to ensure that data coherence is maintained when DMA transactions are performed. Flush protocol does not allow shared data in the cache.

Flush protocol does not require a duplicate tag store. Because the duplicate tag store is optional for this protocol, the Bcache is probed for each transaction to determine if the block is present. If the block is present, the requested action is taken; if the block is not present, the command is still acknowledged, but no other action is taken.

Section 4.6.2 and Section 4.6.3 describe the write invalidate cache coherency protocol in more detail while Section 4.6.4 and Section 4.6.5 provide a more detailed description of flush cache coherency protocol. The system commands that are used to maintain cache coherency are described in more detail in Section 4.10.

4.6.2 Write Invalidate Cache Coherency Protocol Systems

All 21164-based systems that implement the write invalidate cache protocol must have the combinations of components listed in Table 4–5. For example, a system such as that listed in write invalidate (3), having an Scache and Bcache, is required to have a Bcache duplicate tag store and a lock register.

Table 4–5 Components for 21164 Write Invalidate Systems

Cache Protocol	Scache	Scache Duplicate Tag	Bcache	Bcache Duplicate Tag	Lock Register
Write invalidate (1)	Yes	No	No	No	No
Write invalidate (2)	Yes	Yes (full or partial)	No	No	Required
Write invalidate (3)	Yes	No	Yes	Required (full)	Required

Write Invalidate 1

This system has no external cache, duplicate tag store, or lock register. The 21164 must be made aware of all memory data transactions that occur on the system bus. System logic uses an INVALIDATE, READ DIRTY, or READ DIRTY/INVALIDATE transaction to the 21164 to maintain cache coherency and to support the lock mechanism.

Write Invalidate 2

This system has an external Scache duplicate tag store and lock register. System logic uses the duplicate Scache tag store and lock register to partially or completely filter out unneeded transactions to the 21164. System logic maintains the lock mechanism status and initiates transactions that affect Scache coherency.

Write Invalidate 3

This system has an external Bcache, Bcache duplicate tag store and lock register. An Scache duplicate tag store is not needed because the Scache is a subset of the Bcache. This system operates similarly to the write invalidate 2 system, except that the cache is larger. Write invalidate systems with a Bcache require a full Bcache duplicate tag store because the 21164 assumes that a duplicate tag store has been used to completely filter out unneeded transactions. Therefore, the 21164 does not probe the Bcache when system commands are received, but assumes that they will hit in the Bcache.

4.6.3 Write Invalidate Cache Coherency States

Each processor in the system must be able to read and write data as if all transactions were going onto the system bus to memory or I/O modules. Therefore, the system bus is the point at which cache coherency must be maintained.

Table 4–6 describes the Bcache states that determine cache coherency protocol for 21164 systems.

Table 4–6 Bcache States for Cache Coherency Protocols

Valid[1]	Shared[1]	Dirty[1]	State of Cache Line
0	X	X	Not valid.
1	0	0	Valid for read or write operations. This cache line contains the only cached copy of the block and the copy in memory is identical to this line.
1	0	1	Valid for read or write operations. This cache line contains the only cached copy of the block. The contents of the block have been modified more recently than the copy in memory.
1	1	0	Valid for read or write operations. This block may be in another CPU's cache.
1	1	1	Valid for read or write operations. This block may be in another CPU's cache. The contents of the block have been modified more recently than the copy in memory.

[1]The **tag_valid_h**, **tag_shared_h**, and **tag_dirty_h** signals are described in Table 3–1.

Note

Unlike some other systems, the 21164 will not take an update to a shared block but instead will invalidate the block.

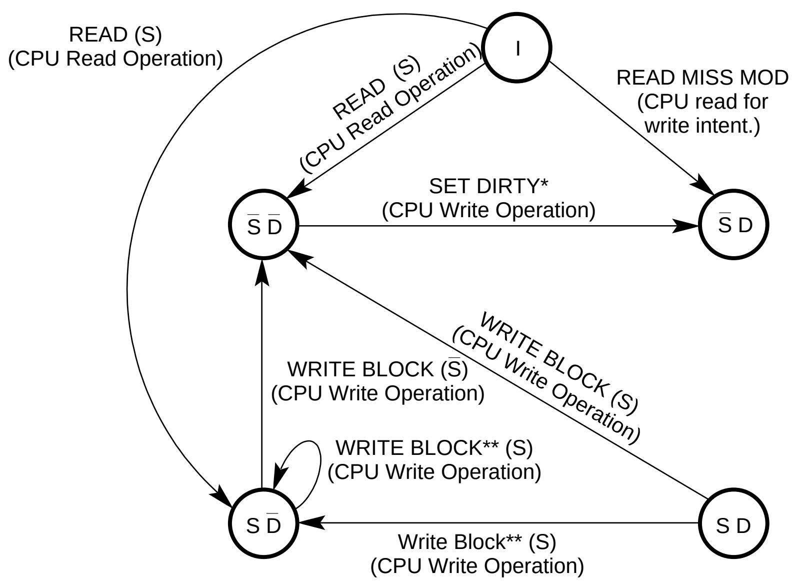
4.6.3.1 Write Invalidate Protocol State Machines

Figure 4–11 shows the 21164 cache state transitions that can occur as a result of 21164 transactions to the system. Figure 4–12 shows the 21164 cache state transitions maintained by the 21164 as a result of transactions by other nodes on the system bus. These two figures both represent the same state machine. They show transitions caused by the 21164, and by the system, separately for clarity.

Note

The abbreviations ''I,S,D'' indicate the INVALID, SHARED, and DIRTY states.

Figure 4–11 Write Invalidate Protocol: 21164 State Transitions

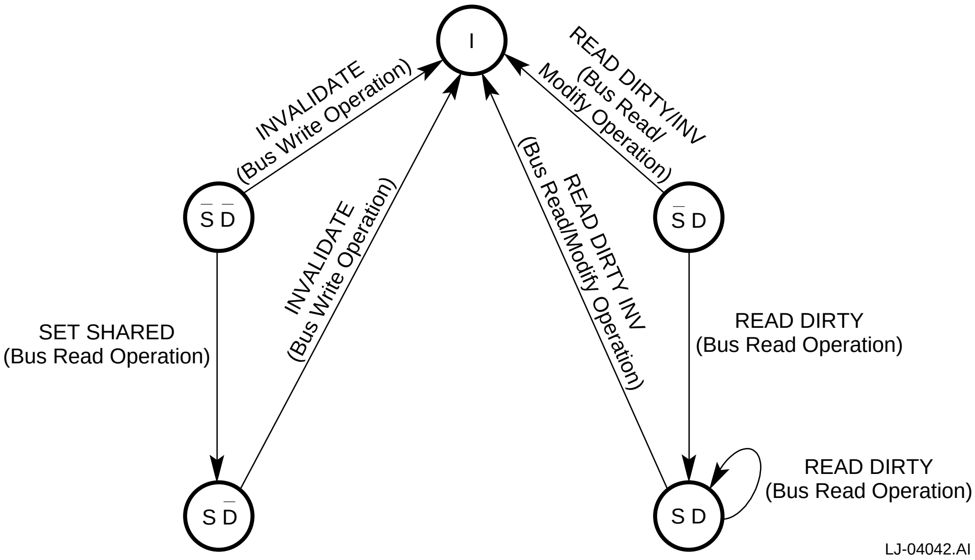


* Optionally this transition can be configured to occur without a SET DIRTY command being issued externally.

** Only allowed in no_Bcache systems.

LJ-04036.AI

Figure 4–12 Write Invalidate Protocol: System/Bus State Transitions



4.6.4 Flush Cache Coherency Protocol Systems

All 21164-based systems that implement the flush cache protocol must have the combinations of components listed in Table 4–7. For example, a system such as that listed in flush (3), having a Bcache and a Bcache duplicate tag store, is required to have a lock register.

4.6 Cache Coherency

Table 4–7 Components for 21164 Flush Cache Protocol Systems

Cache Protocol	Scache	Scache Duplicate Tag	Bcache	Bcache Duplicate Tag	Lock Register
Flush protocol (1)	Yes	No	No	No	No
Flush protocol (1.5)	Yes	Yes (full or partial)	No	No	Required
Flush protocol (2)	Yes	No	Yes	No	No
Flush protocol (3)	Yes	No	Yes	Yes (partial/full)	Required

Flush-Based 1

This system has no external cache, duplicate tag store, or lock register. System logic notifies the 21164 of all memory data read operations that occur on the system bus by using the interface READ command. The 21164 returns data if the block is dirty.

System logic notifies the 21164 of all memory data write operations that occur on the system bus by using the interface FLUSH command. The 21164 invalidates the block in cache, provides the data to the system if the block was dirty, and updates the lock mechanism status.

Flush-Based 1.5

This system has no external cache, but does contain a partial or full duplicate tag store for the Scache and the onchip Scache victim buffers. The SET_DIRTY and LOCK commands should be enabled. The LOCK register is required.

System logic notifies the 21164 of all memory data read operations that hit in the duplicate tag store by using the READ command. The 21164 provides the system with a copy of the dirty data.

System logic notifies the 21164 of all memory data write operations that hit in the duplicate tag store by using the FLUSH command. The 21164 provides the dirty data and then invalidates the block.

Flush-Based 2

This system has an external cache but no duplicate tag store or lock register. System logic and 21164 operation is identical to operation for the flush-based 1 system.

Flush-Based 3

This system has an external cache, a Bcache duplicate tag store, and lock register. System logic notifies the 21164 of all memory data read operations that occur on the system bus to addresses that are valid in the Bcache duplicate tag store. System logic uses the READ command and the 21164 returns data if the block is dirty.

System logic uses the FLUSH command to notify the 21164 of all memory data write transactions that occur on the system bus to addresses that are valid in the Bcache duplicate tag store. If the block is dirty, the 21164 provides the block data and invalidates the block in cache in any case.

System logic updates its lock mechanism status.

Flush-based systems with a Bcache do not require a full Bcache duplicate tag because the 21164 always probes the Bcache in response to system commands.

4.6.5 Flush-Based Protocol State Machines

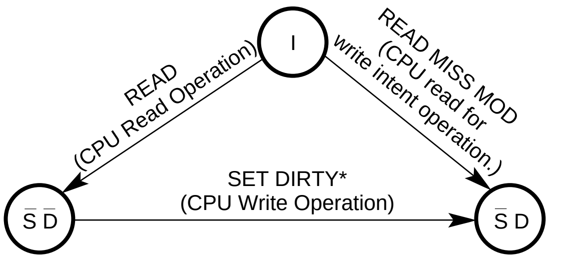
Figure 4–13 shows the 21164 cache state transitions that can occur as a result of transactions with the system. Figure 4–14 shows the 21164 cache state transitions maintained by the 21164 as a result of transactions by other nodes on the system bus. These two figures both represent the same state machine. They show transitions caused by the 21164, and by the system, separately for clarity.

Note

The abbreviations ''I,S,D'' indicate the INVALID, SHARED, and DIRTY states.

4.6 Cache Coherency

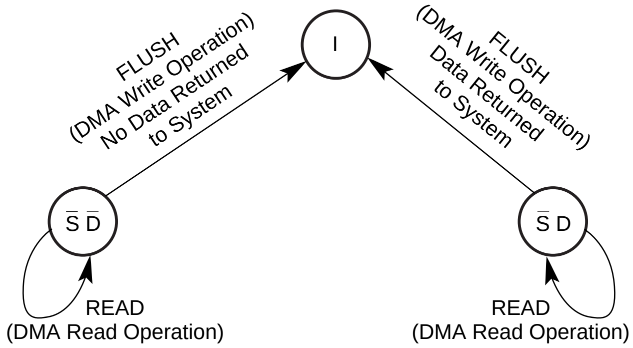
Figure 4–13 Flush-Based Protocol 21164 States



*Optionally this transition can be configured to occur without a SET DIRTY command being issued externally. Refer to BC_CONTROL<EI_CMD_GRP2>.

LJ-04038.AI

Figure 4–14 Flush-Based Protocol System/Bus States



LJ-04037.AI

4.6.6 Cache Coherency Transaction Conflicts

Cache coherency conflicts that can occur during system operation are described here. Systems should be designed to avoid these conflicts.

4.6.6.1 Case 1

If the 21164 requests a READ MISS MOD transaction, it expects the block to be returned $\overline{\mathrm{SHARED}}$, DIRTY. However, if the system returns the data SHARED, $\overline{\mathrm{DIRTY}}$, the 21164 follows with a WRITE BLOCK command. This might cause a multiprocessor system to have live-lock problems, a condition that can cause long delays in writing from the 21164 to memory.

4.6.6.2 Case 2

If the 21164 attempts to write a clean/private block of memory, it sends a SET DIRTY command to the system. The system could be sending a SET SHARED or INVALIDATE command to the 21164 at the same time for the same block. The bus is the coherence point in the system; therefore, if the bus has already changed the state of the block to shared, setting the dirty bit is incorrect. The 21164 will not resend the SET DIRTY command when the ownership of the ADDRESS/CMD bus is returned. The write will be restarted and will use the new tag state to generate a new system request.

Another possibility is for the system to send an INVALIDATE instruction at the same time the 21164 is attempting to do a WRITE BLOCK transaction to the same block. In this case, the 21164 aborts the WRITE BLOCK transaction, services the INVALIDATE instruction, then restarts the write transaction, which produces a READ MISS command.

In both of these cases, if the SET DIRTY or WRITE BLOCK transaction is started by the 21164 and then interrupted by the system, the 21164 resumes the same transaction unless the system request was to the same block as the request the 21164 had started. In this case, the 21164 request is restarted internally by the CPU and it is UNPREDICTABLE what transaction the 21164 presents next to the system.

4.7 Lock Mechanisms

The LD*x*_L instruction is forced to miss in the Dcache. When the Scache is read, the BIU's lock IPR is loaded with the physical address and the lock flag set. The BIU sends a LOCK command to the system so that it can load its own lock register. The system lock register is used only if the locked block is displaced from the cache system.

The lock flag is cleared if any of the following events occur:

- Any write operation from the bus addresses the locked block (FLUSH, INVALIDATE, or READ DIRTY/INV).
- An ST*x*_C is executed by the processor.
- The locked block is refilled from memory and **system_lock_flag_h** is cleared.

The system copy of the lock register is required on systems that have a duplicate tag store to filter write traffic. The direct-mapped Icache, Dcache, and Bcache; along with the subsetting rules, branch prediction, and Istream prefetching, can cause a lock to always fail because of constant Scache thrashing of the locked block. Each time a block is loaded into the Scache, the value of the lock register is logically ANDed with the value of signal **system_lock_flag_h**. If the locked block is displaced from the cache system, the 21164 does not ''see'' bus write operations to the locked block. In this case, the system's copy of the lock register corrects the processor copy of the lock flag when the block is filled into the cache, using signal **system_lock_flag_h**.

Systems that do not have duplicate tag stores, and send all probe traffic to the 21164, are not required to implement a lock register or lock flag. Such systems should permanently assert signal **system_lock_flag_h**.

When the ST*x*_C instruction is issued, the Ibox stops issuing memory-type instructions. The store updates the Dcache in the usual way, and places itself in the write buffer. It is not merged with other pending write operations. The write buffer is flushed.

When the write buffer arrives at an ST*x*_C instruction in cached memory, it probes the Scache to check the block state. When the ST*x*_C passes through the Scache, an INVALIDATE command is sent to the Dcache. If the lock flag is clear, the ST*x*_C fails. If the block is $\overline{\text{SHARED}}, \text{DIRTY}$, the write buffer writes the ST*x*_C data into the Scache. Success is written to the register file and the Ibox begins issuing memory instructions again. If the block is in the shared state, the BIU requests a WRITE BLOCK transaction. If the system CACKs the WRITE BLOCK transaction, the Scache is written and the Ibox starts as previously stated.

When the write buffer arrives at an ST*x*_C instruction in noncached memory, it probes the Scache to check the block state. The Scache misses, the state of the lock flag is ignored, and the BIU requests a WRITE BLOCK LOCK transaction. If the system CACKs the WRITE BLOCK LOCK transaction, the Ibox starts as stated previously. If **cfail_h** is asserted along with **cack_h**, then the ST*x*_C fails.

4.8 Alpha 21164-to-Bcache Transactions

When initiating an Istream or Dstream data transaction, the 21164 first tries the Icache or Dcache, respectively. If that access is unsuccessful, then the Scache will be tried next. If that fails, then the 21164 tries the Bcache.

The 21164 interface to the system and Bcache is in the Cbox. The Cbox provides address and control signals for transactions to and from the Bcache and the system interface logic. The Cbox also transfers data across the 128-bit bidirectional data bus.

The 21164 controls all Bcache transactions and will be able to process read and write hits to the Bcache without assistance from the system. When system logic writes to or reads from the Bcache, it transfers data to and from the Bcache but only under the direct control of the 21164.

Note

Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.11.

4.8.1 Bcache Timing

Bcache cycle time may be faster than, identical to, or slower than, that of the sysclk. If the system is involved in a Bcache transaction, each read or write operation starts on a sysclk edge. It is the responsibility of the system to control the rate of Bcache transactions by using the **dack_h** signal. Read and write operations that are private to the 21164 and Bcache may start on any CPU clock. There is no relation between sysclk and private Bcache accesses.

Bcache timing is configured using the BC_CONFIG and BC_CONTROL IPRs. Section 5.3.5 and Section 5.3.4 show the layout of these registers. These registers are normally configured by 21164 initialization code.

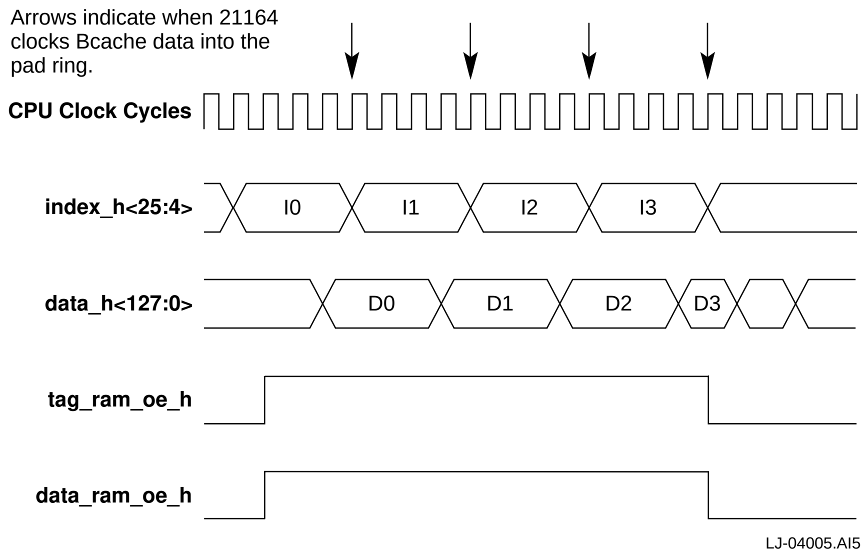
Bcache read timing and write timing are programmable. Read speed is selected using BC_CONFIG<7:4> (BC_RD_SPD<3:0>). Write speed is selected using BC_CONFIG<11:8> (BC_WR_SPD<3:0>).

4.8 Alpha 21164-to-Bcache Transactions

4.8.2 Bcache Read Transaction (Private Read Operation)

Figure 4–15 shows an example of the timing for a private read operation to Bcache by the 21164. BC_CONFIG<BC_RD_SPD> (read speed) is set to 4 CPU cycles, the minimum read time (maximum read speed).

Figure 4–15 Bcache Read Transaction



The index increments through four 16-byte addresses, each being asserted for four CPU cycles. The Bcache logic waits BC_CONFIG<BC_RD_SPD<3:0>> cycles before recieving the data.

The 21164 always delays one cycle before asserting the **tag_ram_oe_h** and **data_ram_oe_h** lines. The lines are deasserted when the fourth index address is deasserted.

4.8.3 Wave Pipeline

The wave pipeline is implemented to improve performance for systems that use 64-byte block size. It is not supported for systems with 32-byte block size.

The wave pipeline is controlled using BC_CONFIG<7:4> (BC_RD_SPD<3:0>) and BC_CTL<18:17> (BC_WAVE<1:0>).

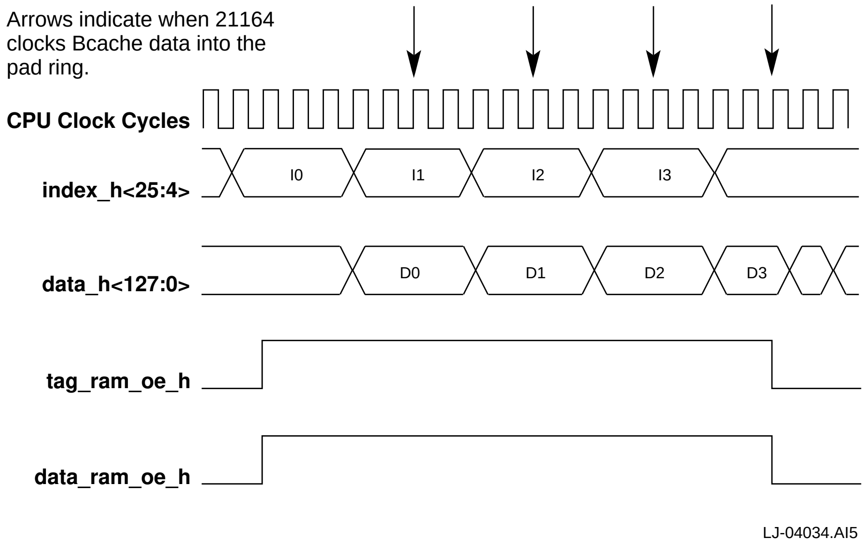
BC_CONFIG<7:4> (BC_RD_SPD<3:0>) is set to the latency of the Bcache read transaction. BC_CTL<18:17> (BC_WAVE<1:0>) is set to the number of cycles to subtract from BC_RD_SPD to get the Bcache repetition rate.

For example, if BC_RD_SPD is set to 6 and BC_WAVE<1:0> is set to 2, it takes 6 cycles for valid data to arrive at the pins, but a new read starts every 4 cycles.

The read repetition rate must be greater than 3. For example it is not permitted to set BC_RD_SPD to 5 and BC_WAVE<1:0> to 2.

The example shown in Figure 4–16 has BC_RD_SPD=6, BC_WAVE<1:0>=2.

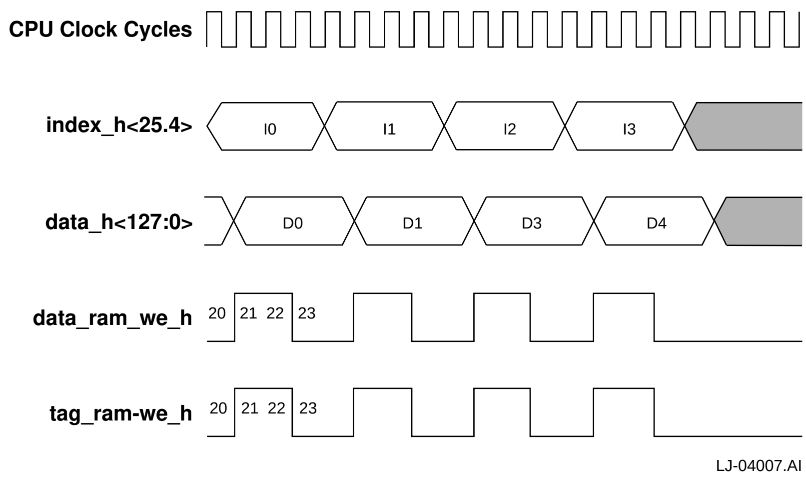
Figure 4–16 Wave Pipeline Timing Diagram



4.8.4 Bcache Write Transaction (Private Write Operation)

Figure 4–17 shows an example of the timing for a private write operation to Bcache by the 21164. BC_CONFIG<BC_WR_SPD> (write speed) is set to 4 CPU cycles, the minimum time.

Figure 4–17 Bcache Write Transaction



The index increments through four 16-byte addresses, each being asserted for four cycles. The 21164 always delays one cycle then drives the data associated with each index.

Signals **tag_ram_we_h** and **data_ram_we_h** are asserted high for two cycles because the BC_CONFIG<28:20> (BC_WE_CTL<8:0>) is set to 6. BC_CONFIG<22:21> being set causes the write-enable lines to be asserted during the second and third CPU cycles. BC_CONFIG<20,23> being clear causes the write-enable lines to not be asserted during the first and fourth CPU cycles.

The Bcache maximum read or write time is 15 cycles. The minimum read or write time is 4 cycles; except that in 32-byte mode, the minimum read time is 5 cycles. So the index and data can be asserted from 4 to 15 cycles. The write-enable signals can be asserted from 0 to 9 cycles. If BC_CONFIG(BC_WE_CTL) is set to 0, the write-enable signals will not be asserted. If the 9-bit field is set to $1FF_{16}$, then the write-enable signals will be asserted for 9 CPU cycles.

4.8.5 Selecting Bcache Options

Table 4–8 lists the variables to consider when designing and implementing a Bcache.

Table 4–8 Bcache Options

Parameter	Selection
Sysclk ratio (3–15)	____CPU cycles
Cache protocol, write invalidate or flush	____
Cache block size 64/32	____–byte block
ECC or byte parity	____
Bcache present?	____
Bcache size (1M byte to 64M bytes)	____M byte
Bcache read speed (4–15)	____CPU cycles
Bcache wave pipelining (0–3)	____CPU cycles
Bcache victim buffer?	____
Bcache write speed (4–15)	____
Bcache read to write spacing (1–7)	____
Bcache fill write pulse offset (1–7)	____
Bcache write pulse (bit mask 9–0)	____
Enable LOCK and SET DIRTY commands?	____
Enable memory barrier (MB) commands?	____

4.9 Alpha 21164-Initiated System Transactions

This section describes how commands are used to move data between the 21164 and its cache system.

Note

Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.11.

The 21164 starts an external transaction when:

- It encounters a ''miss.''
- A LOCK command is invoked.
- A WRITE command is directed at a shared block.
- A WRITE command is directed at a clean block in Scache.
- The CPU addresses a noncached region of memory.
- The 21164 executes a FETCH, FETCH_M, or MB instruction.

For example, the sequence for a 21164-initiated transaction caused by a Bcache miss is:

- At the start of a Bcache transaction, the 21164 checks the tag and tag control status of the target block.
- If there is a tag mismatch or the Valid bit is clear, a Bcache miss has occurred and the 21164 starts an external READ MISS transaction that tells the system logic to access and return data.
- System logic acknowledges acceptance of the command from the 21164 by asserting **cack_h**.
- Because the transaction is a read operation, requiring a FILL transaction, the transaction is broken (pended) while system logic obtains the FILL data.
- At a later time, the system asserts **fill_h**.
- The 21164 will assert the tag and tag control bits, and will control the write action during the FILL transaction.
- The system logic provides the data. As each of the two (or four) data cycles becomes valid, the system logic asserts **dack_h** to cause the 21164 to sample to data and write it into the Bcache.

4.9 Alpha 21164-Initiated System Transactions

Interface commands from the 21164 to the system are driven on the **cmd_h<3:0>** signals. Table 4–9 lists and describes the set of interface commands.

Table 4–9 Alpha 21164-Initiated Interface Commands

Command	cmd_h <3:0>	Description
NOP	0000	The NOP command is driven by the owner of the **cmd_h** bus when it has no tasks queued.
LOCK	0001	The LOCK command is used to load the system lock register with a new lock register address. The state of the system lock register flag is used on each fill to update the 21164's copy of the lock flag. Refer to Section 4.7 for more information.
FETCH	0010	The 21164 passes a FETCH instruction to the system when the FETCH instruction is executed.
FETCH_M	0011	The 21164 passes a FETCH_M instruction to the system when the FETCH_M instruction is executed.
MEMORY BARRIER	0100	The 21164 issues the MEMORY BARRIER command when an MB instruction is executed. This command should be used to synchronize read and write accesses with other processors in the system. The 21164 stops issuing memory reference instructions and waits for the command to be acknowledged before continuing.
SET DIRTY	0101	Dirty bit set if shared bit is clear. The 21164 uses the SET DIRTY command when it wants to write a clean, private block in its Scache and it wants the dirty bit set in the duplicate tag store. The 21164 does not proceed with the write until a CACK response is received from the system. When the CACK is received, the 21164 attempts to set the dirty bit. If the shared bit is still clear, the dirty bit is set and the write operation is completed. If the shared bit is set, the dirty bit is not set and the 21164 requests a WRITE BLOCK transaction. The copy of the dirty bit in the Bcache is not updated until the block is removed from the Scache.

(continued on next page)

Table 4–9 (Cont.) Alpha 21164-Initiated Interface Commands

Command	cmd_h <3:0>	Description
WRITE BLOCK	0110	Request to write a block. When the 21164 wants to write a block of data back to memory, it drives the command, address, and first INT16 of data on a sysclk edge. The 21164 outputs the next INT16 of data when **dack_h** is received. When the system asserts **cack_h**, the 21164 removes the command and address from the bus and begins the write of the Scache. Signal **cack_h** can be asserted before all the data is removed.
WRITE BLOCK LOCK	0111	Request to write a block with lock. This command is identical to a WRITE BLOCK command except that the **cfail_h** signal may be asserted by the system, indicating that the data cannot be written. This command is only used for ST*x*_C in noncached space.
READ MISS0	1000	Request for data. This command indicates that the 21164 has probed its caches and that the addressed block is not present.
READ MISS1	1001	Request for data. This command indicates that the 21164 has probed its caches and that the addressed block is not present.
READ MISS MOD0	1010	Request for data; modify intent. This command indicates that the 21164 plans to write to the returned cache block. Normally, the dirty bit should be set when the tag status is returned to the 21164 on a Bcache fill.
READ MISS MOD1	1011	Request for data; modify intent. This command indicates that the 21164 plans to write to the returned cache block. Normally, the dirty bit should be set when the tag status is returned to the 21164 on a Bcache fill.

(continued on next page)

Table 4–9 (Cont.) Alpha 21164-Initiated Interface Commands

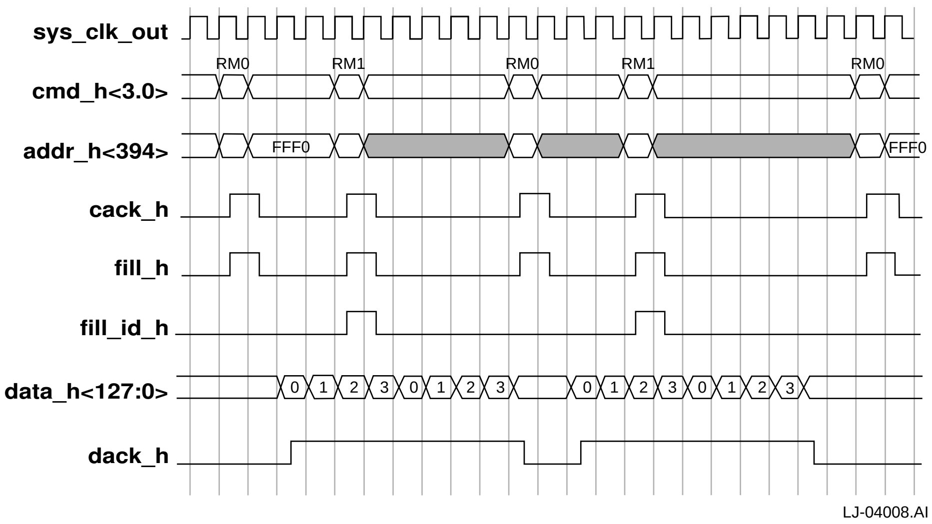
Command	cmd_h <3:0>	Description
BCACHE VICTIM	1100	Bcache victim should be removed. If there is a victim buffer in the system, this command is used to pass the address of the victim to the system. The READ MISS command that produced the victim precedes the BCACHE VICTIM command. Signal **victim_pending_h** is asserted during the READ MISS command to indicate that a BCACHE VICTIM command is waiting, and that the Bcache is starting the read of the victim data. If the system does not have a victim buffer, the BCACHE VICTIM command precedes the READ MISS commands. The BCACHE VICTIM command is driven, along with the address of the victim. At the same time, the Bcache is read to provide the victim data. If the system does have a victim buffer, and it asserts signal **dack_h** any time before the BCACHE VICTIM command is driven, then address bits **addr_h<5:4>** of the address sent with the BCACHE VICTIM command are UNPREDICTABLE. The system must use the values of **addr_h<5:4>** that were sent with the READ MISS command that produced the victim.
—	1101	Spare.
READ MISS MOD STC0	1110	Request for data, ST*x*_C data.
READ MISS MOD STC1	1111	Request for data, ST*x*_C data.

4.9 Alpha 21164-Initiated System Transactions

4.9.1 READ MISS—No Bcache

A read operation to the Dcache misses causing a read operation to the Scache, which also misses. After the Scache miss there is no Bcache probe—the 21164 sends a READ MISS command to the system. The system acknowledges receipt of the READ MISS by asserting **cack_h** as shown in Figure 4–18.

Figure 4–18 READ MISS—No Bcache Timing Diagram



4.9.2 READ MISS—Bcache

The 21164 starts a Bcache read operation on any CPU clock. The index is asserted to the RAM for a programmable number of CPU cycles in the range of 4 to 15. The tag is accessed at the same time. At the end of the first read operation, the 21164 latches the data and tag information and begins the read operation of the next 16 bytes of data. The tag is checked for a hit. If there is a miss, a READ MISS or READ MISS MOD command, along with the address, is queued to the **cmd_h<3:0>** bus. It appears on the interface at the next sysclk edge.

Figure 4–19 shows the timing of a Bcache read and the resulting READ MISS MOD request. The system immediately asserts **cack_h** to acknowledge the command. This allows the 21164 to make additional READ MISS requests. It is also possible for the system to defer assertion of **cack_h** until the fill data is returned. This allows the system to use **cmd_h<0>** for the value of **fill_id_h**. The assertion of **cack_h** should arrive no later than the last fill **dack_h**.

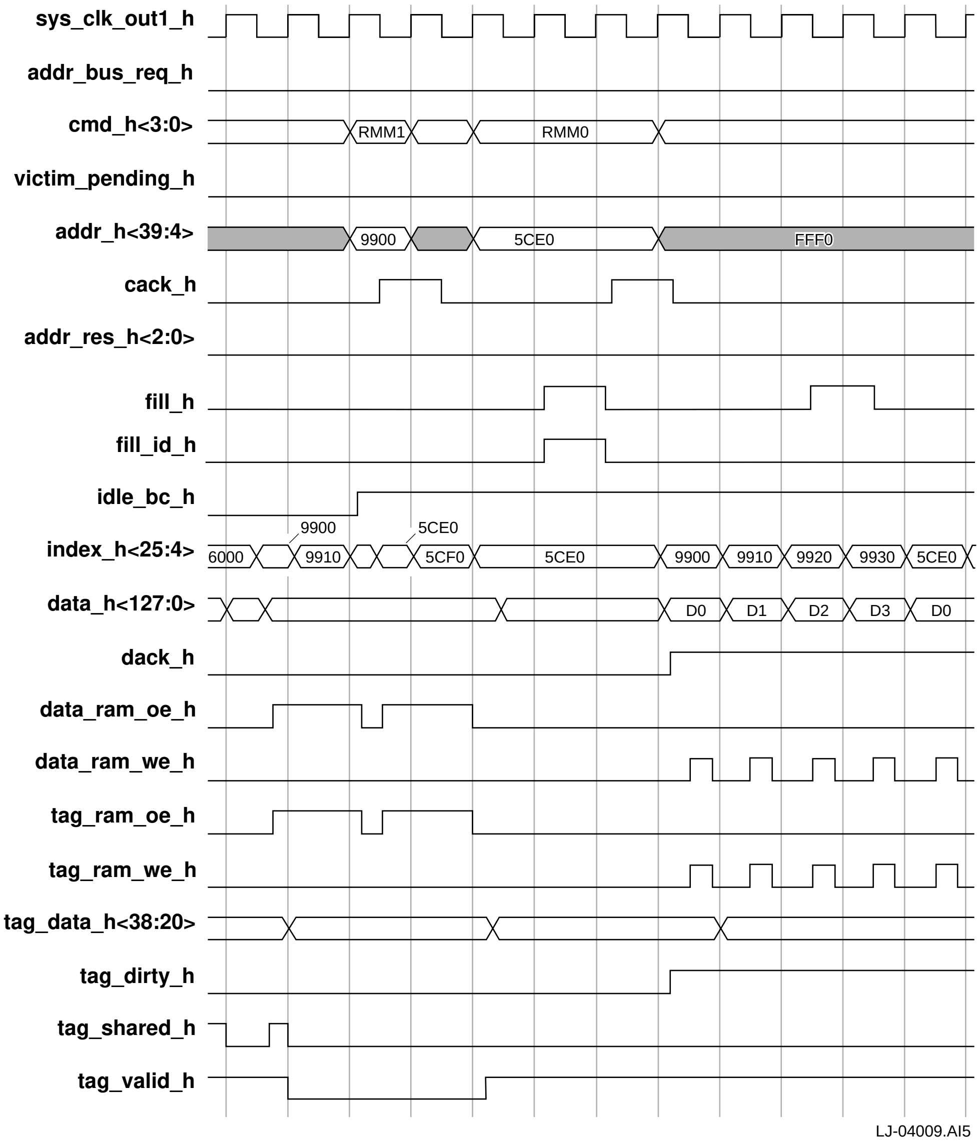
The only difference between a READ MISS and a READ MISS MOD sequence on the bus is that **tag_dirty_h** should be asserted during the Bcache fill associated with a READ MISS MOD.

Note

A READ MISS command with **int4_valid_h<3:0>** of zero is a request for Istream data while **int4_valid_h<3:0>** of non-zero is a request for Dstream data.

4.9 Alpha 21164-Initiated System Transactions

Figure 4–19 READ MISS MOD—Bcache Timing Diagram



4.9.3 FILL

Signals **fill_h**, **fill_id_h**, and **fill_error_h** are used to control the return of fill data to the 21164 and the Bcache, if it is present. Signal **idle_bc_h** must be used to stop CPU requests in the Bcache in such a way that the Bcache will be idle when the fill data arrives (but not the FILL command). Signal **fill_h** should be asserted at least two sysclk periods before the fill data arrives. Signal **fill_id_h** should be asserted at the same time to indicate whether the FILL is for a READ MISS0 or READ MISS1 operation. The 21164 uses this information to select the correct fill address. Figure 4–19 shows the timing of a FILL command. Refer also to Section 4.11.3 for more information on using signals **idle_bc_h** and **fill_h**.

If signals **fill_h** and **fill_id_h** are asserted at the rising edge of sysclk N, then at the rising edge of sysclk N+1, the 21164 tristates **data_h<127:0>**, asserts the Bcache index, and begins a Bcache write operation. The system should drive the data onto the data bus and assert **dack_h** before the end of the sysclk cycle. At the end of the write time, the 21164 waits for the next sysclk edge. If **dack_h** has not been asserted, the Bcache write operation starts again at the same index. If **dack_h** is asserted, the index advances to the next part of the fill and the write operation begins again. The system must provide the data and **dack_h** signal at the correct sysclk edges to complete the fill correctly. For example, if the Bcache requires 17 ns to write, and the sysclk is 12 ns, then two sysclk cycles are required for each write operation.

The 21164 calculates and asserts **tag_valid_h** and writes the Bcache tag store with each INT16 of data. The system is required to drive signals **tag_shared_h**, **tag_dirty_h**, and **tag_ctl_par_h** with the correct value for the entire FILL transaction.

At the end of the FILL transaction, the 21164 will not assert **data_ram_oe_h** or begin to drive the data bus until the fifth CPU cycle after the sysclk that loads the last DACK. If systems require more time to turn off their drivers, they must use **idle_bc_h** in combination with **data_bus_req_h** to stop 21164 requests, and not send any system requests.

4.9.4 READ MISS with Victim

The 21164 supports two models for removing displaced dirty blocks from the Bcache. The first assumes that the system does not contain a victim buffer. In this case, the victim must be read from the Bcache before the new block can be requested. In the second case, where the system has a victim buffer, the 21164 requests the new block from memory while it starts to read the victim from the Bcache. The VICTIM command and address follows the miss request.

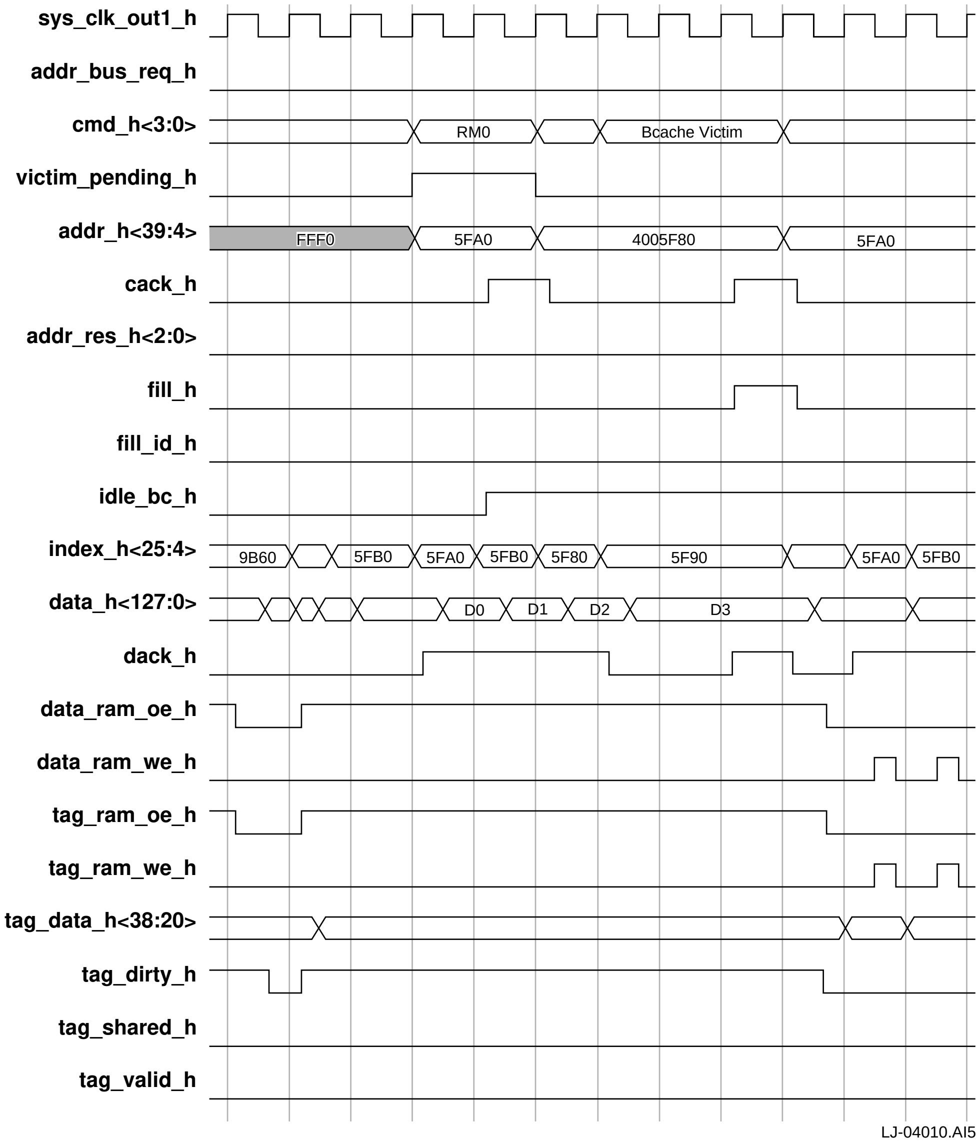
In either case, the 21164 treats a miss/victim as a single transaction. If the assertion of **addr_bus_req_h** or **idle_bc_h** causes the BIU sequencer to reset, both the READ MISS and BCACHE VICTIM transactions are restarted from the beginning. For example, if the 21164 is operating in victim first mode, and it sends a BCACHE VICTIM command to the system, then the system sends an INVALIDATE request to the 21164. The 21164 processes the INVALIDATE request and then restarts the READ operation and resends the BCACHE VICTIM command and data, and then processes the READ MISS.

Sections 4.9.4.1 and 4.9.4.2 describe each of these methods of victim processing.

4.9.4.1 READ MISS with Victim (Victim Buffer)

When the miss is detected, if the system has a victim buffer, the 21164 waits for the next sysclk, then asserts a READ MISS command, the read miss address, the **victim_pending_h** signal, and indexes the Bcache to begin the read operation of the victim. When the system asserts **cack_h**, the 21164 sends out a NOP command along with the victim address. In the following cycle the BCACHE_VICTIM command is driven. Each assertion of **dack_h** causes the Bcache index to advance to the next part of the block. Figure 4–20 shows the timing of a READ MISS command with a victim.

Figure 4–20 READ MISS with Victim (Victim Buffer) Timing Diagram



4.9.4.2 READ MISS with Victim (Without Victim Buffer)

If the system does not contain a victim buffer, the 21164 stops reading the Bcache as soon as the miss is detected. This occurs while the second INT16 data is on **data_h<127:0>**, as shown in Figure 4–21.

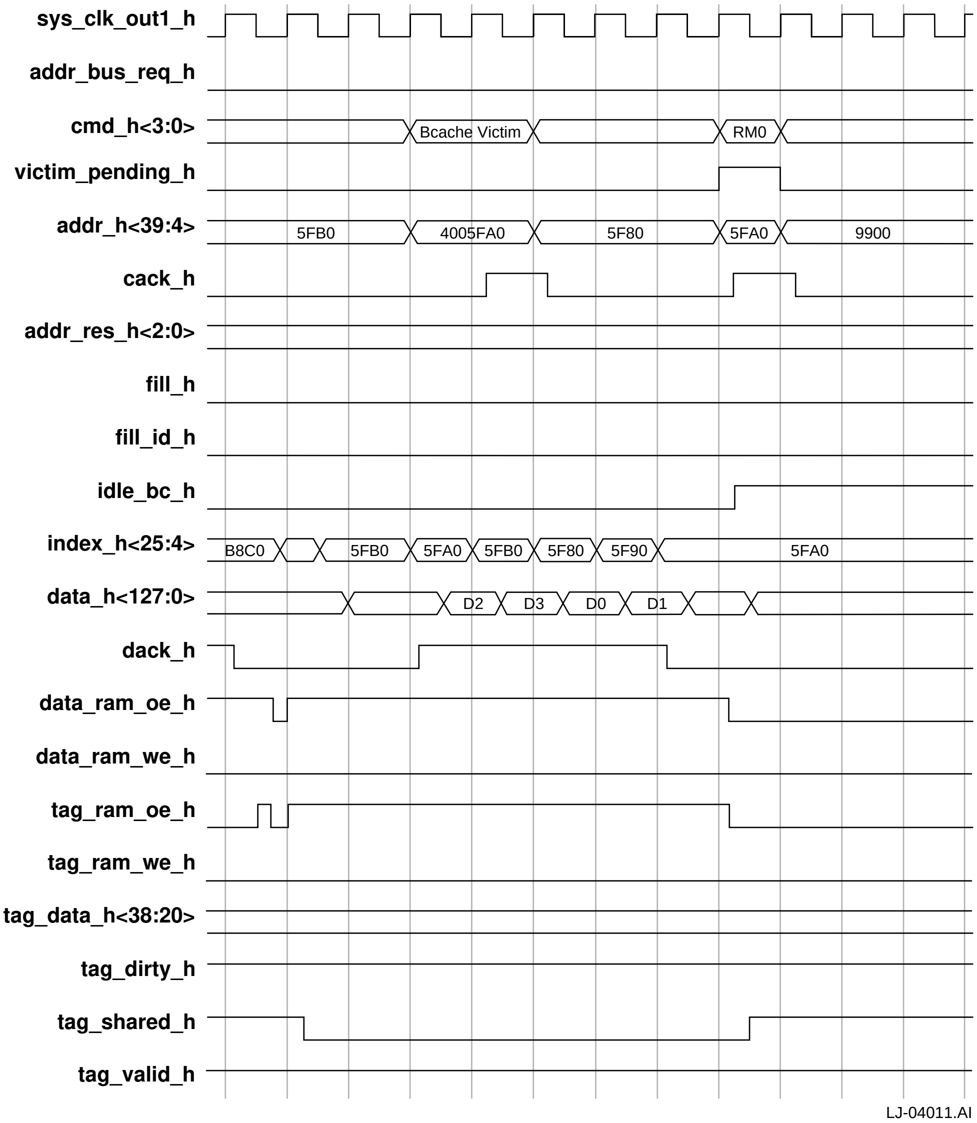
A BCACHE VICTIM command is asserted at the next sysclk along with the victim address. A Bcache read operation of the victim is also started at the sysclk edge.

When **dack_h** is received for the first INT16 of the victim, the 21164 begins reading the next INT16 of the victim. **cack_h** can be sent any time before the last **dack_h** is asserted or with the last **dack_h** assertion.

The 21164 sends the READ MISS command after the last **dack_h** is received. Figure 4–21 shows the timing of a victim being removed.

Notice the data wrap sequence of this transaction—D2, D3, D0, and D1.

Figure 4–21 READ MISS with Victim (without Victim Buffer) Timing Diagram



4.9.5 WRITE BLOCK and WRITE BLOCK LOCK

The WRITE BLOCK command is used to complete write operations to shared data, to remove Scache victims in systems without a Bcache, and to complete write operations to noncached memory.

The WRITE BLOCK LOCK command follows the same protocol. The LOCK qualifier allows the system to be more ‘‘conservative’’ on interlocked write operations to noncached memory space. Refer to Section 4.7 for more information on lock mechanisms.

The WRITE BLOCK command to cached memory regions that source data from the Scache sends data to the system and also causes the data to be written in the Bcache.

The 21164 asserts the WRITE BLOCK command, along with the address and the first 16 bytes of data, at the start of a sysclk. If the system removes ownership of the **cmd_h<3:0>** bus, the 21164 retains the WRITE command and waits for bus ownership to be returned. If the block in question is invalidated, the 21164 restarts the write operation. This results in the READ MISS MOD request instead.

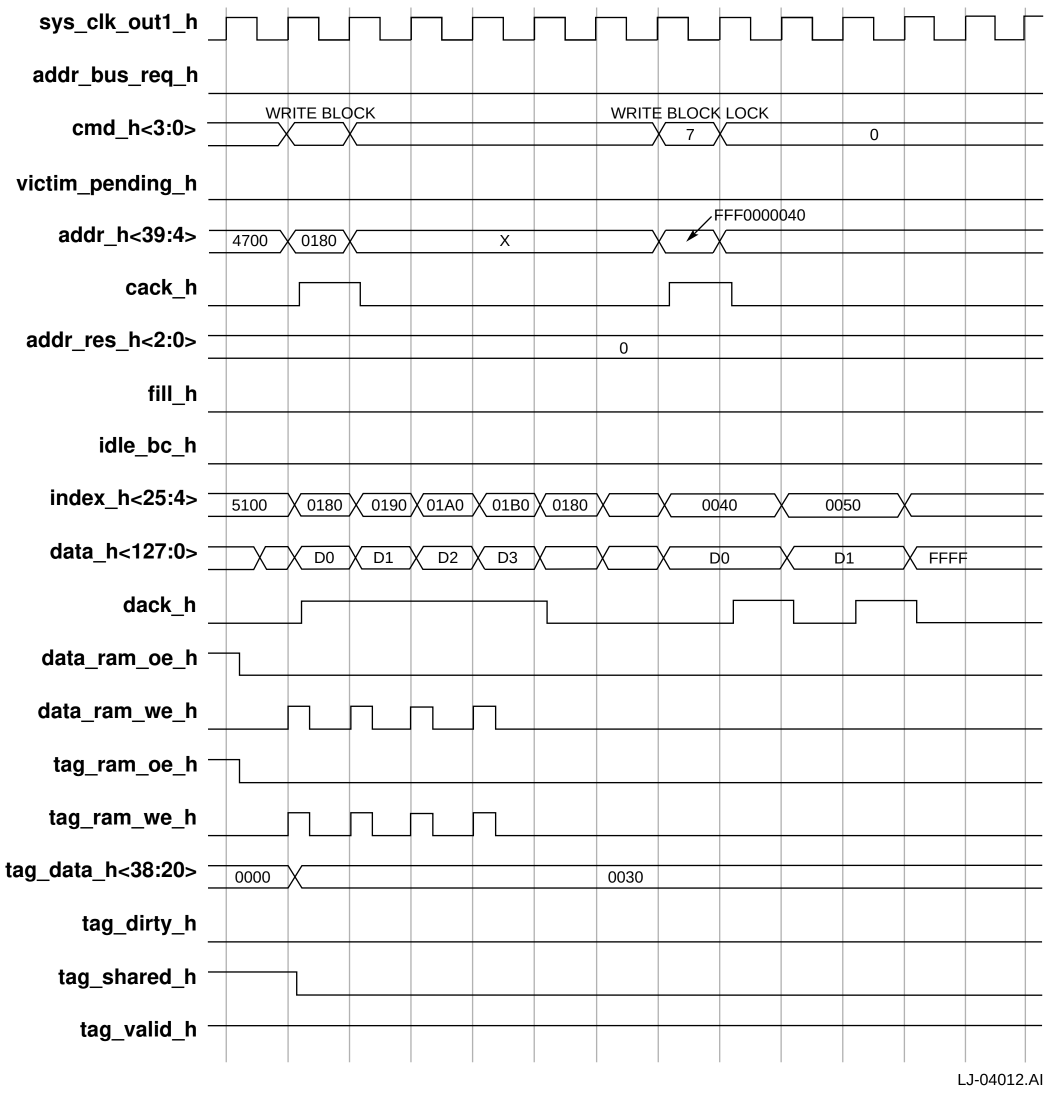
When the system takes the first part of the data, it asserts **dack_h**. This causes the 21164 to drive the next 16 bytes of data on the same sysclk edge.

If the system asserts **cack_h**, the 21164 outputs the next command in the next sysclk. Receipt of signal **cack_h** indicates to the 21164 that the write operation will be taken, and that it is safe to update the Scache with the new version of the block.

During each cycle, the **int4_valid_h<3:0>** signals indicate which INT4 parts of the write operation are really being written by the processor. For write operations to cached memory, all of the data is valid. For write operations to noncached memory, only those INT4 with the **int4_valid_h<*n*>** signal asserted are valid. See the definition for **int4_valid_h<*n*>** in Table 3–1.

Figure 4–22 shows the timing of a WRITE BLOCK command.

Figure 4–22 WRITE BLOCK Timing Diagram



4.9.6 SET DIRTY and LOCK

Figure 4–23 shows the timing of a SET DIRTY and a LOCK operation.

The 21164 uses the SET DIRTY transaction to inform a duplicate tag store that a cached block is changing from the $\overline{\mathrm{SHARED}}$, $\overline{\mathrm{DIRTY}}$ state to the $\overline{\mathrm{SHARED}}$, DIRTY state. When **cack_h** is received from the system, the 21164 sets the dirty bit. If a SET SHARED or INVALIDATE command is received for the same block, the 21164 responds with a WRITE BLOCK or READ MISS MOD command.

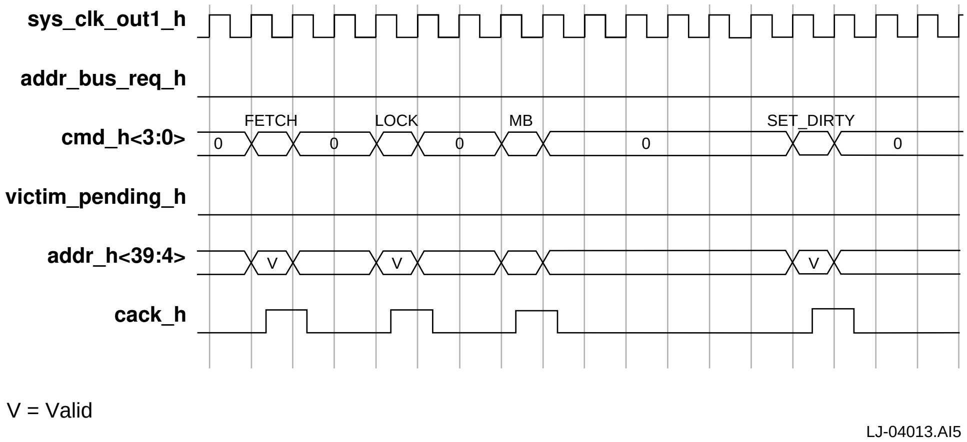
The SET DIRTY and LOCK commands must be enabled in any system that contains a duplicate tag store. The 21164 uses the SET DIRTY command to update the dirty bit in the duplicate tag store.

The 21164 uses the LOCK command to pass the address of a LD*x*_L to the system. A system lock register is required in any system that filters write traffic with a duplicate tag store. If the locked block is displaced from the 21164 caches, the 21164 uses the value of the system lock register to determine if the LD*x*_L/ST*x*_C sequence should pass or fail.

The system may use BC_CONTROL<EI_CMD_GRP2> to modify operation for these commands.

- If BC_CONTROL<EI_CMD_GRP2> is set, the 21164 is allowed to issue SET DIRTY and LOCK commands to the system interface. The system logic acknowledges receipt of these commands.
- If BC_CONTROL<EI_CMD_GRP2> is clear, the SET_DIRTY command will never be driven by the 21164. It is UNPREDICTABLE if the LOCK command is driven. However, the system should never assert **cack_h** for the command when BC_CONTROL<EI_CMD_GRP2> is clear.

Figure 4–23 SET DIRTY and LOCK Timing Diagram



4.9.7 Memory Barrier (MB)

The 21164 may encounter a memory barrier (MB) instruction when executing the instruction stream. The action taken by the 21164 depends upon the state of BC_CONTROL<3> (EI_CMD_GRP3).

- If BC_CONTROL<EI_CMD_GRP3> is set, the 21164 drains its pipeline and buffers, then issues an MB command to the system interface. The system logic must empty its buffers and complete all pending transactions before acknowledging receipt for the MB command by asserting **cack_h**.
- If BC_CONTROL<EI_CMD_GRP3> is clear, the 21164 never drives a MB command to the interface command pins.

Note

The address presented on **addr_h<39:4>** during a MB transaction is UNPREDICTABLE.

4.9.7.1 When to Use a MEMORY BARRIER Command

If the system interface buffers invalidate between the duplicate tag store and the 21164, then the system interface must enable the MB command and drain all invalidates before asserting **cack_h** in response to an MB command.

4.9.8 FETCH

The 21164 passes a FETCH command to the system when it executes a FETCH instruction. The system responds to the command by asserting **cack_h**. This command acts as a ‘‘hint’’ to the system. The system may respond with optional behavior as a result of this hint (refer to the *Alpha Architecture Reference Manual*).

4.9.9 FETCH_M

The 21164 passes a FETCH_M (fetch with modify intent) command to the system when it executes a FETCH_M instruction.

4.10 System-Initiated Transactions

System commands to the 21164, are driven on the **cmd_h<3:0>** signal lines. Before driving these signals, the system must gain control of the command and address buses by using **addr_bus_req_h**, as described in Section 4.11.1. The algorithm used by the 21164 for accepting system commands to be processed in parallel by the 21164 is presented in Section 4.10.1.

System-initiated commands may be separated into two protocol groups. The group of commands used by write invalidate protocol systems is listed and described in Section 4.10.2. The group of commands used by flush-based protocol systems is listed and described in Section 4.10.3.

Note

Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.11.

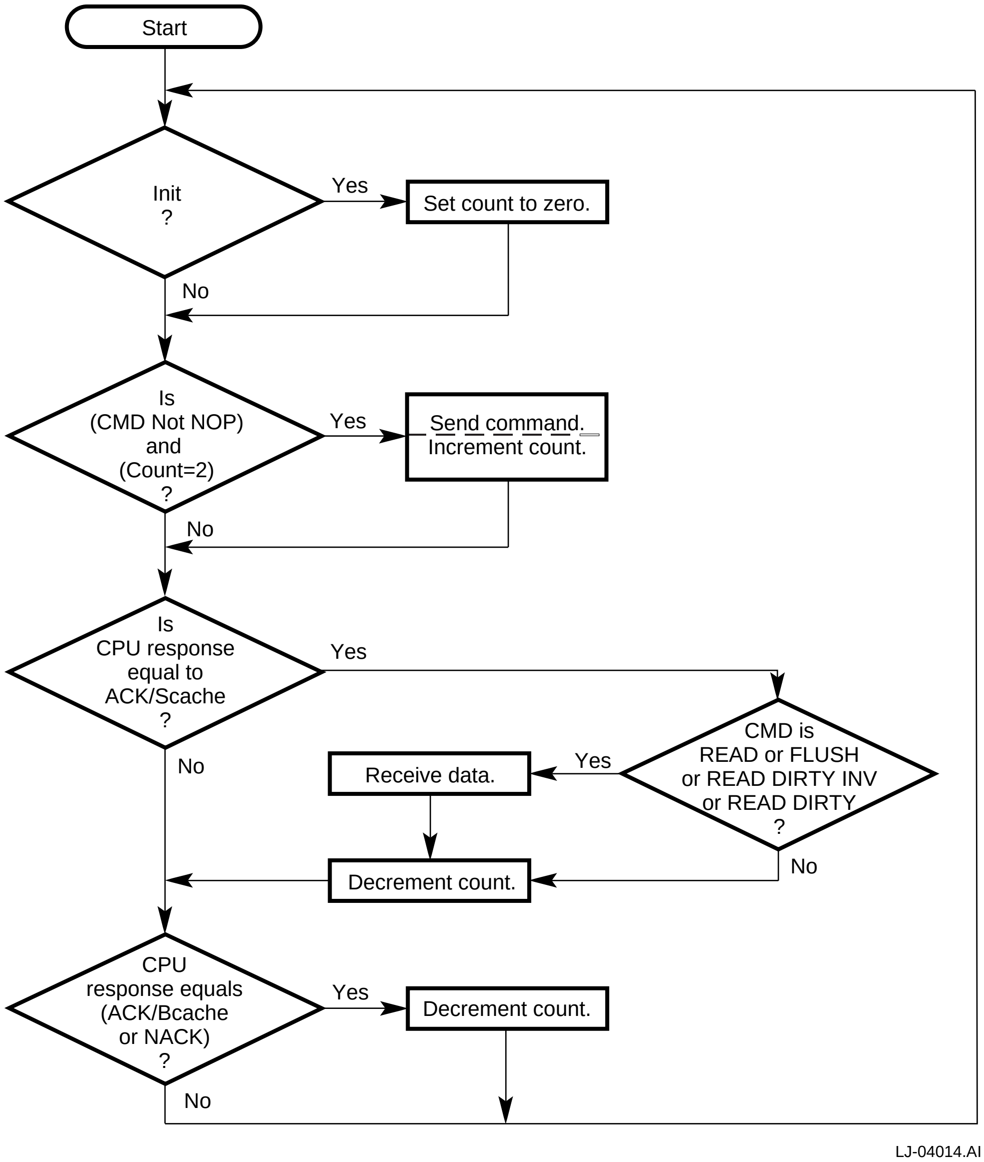
4.10.1 Sending Commands to the 21164

The rules used by the Cbox BIU to process commands sent by the system to the 21164 are listed in Section 4.13.1.

The 21164 can hold two outstanding commands from the system at any time. The algorithm used by the system to send commands to the 21164 without overflowing the two Cbox BIU command buffers is shown in Figure 4–24.

4.10 System-Initiated Transactions

Figure 4–24 Algorithm for System Sending Commands to the 21164



4.10.2 Write Invalidate Protocol Commands

All 21164-based systems that use the write invalidate protocol are expected to use the READ DIRTY, READ DIRTY/INVALIDATE, INVALIDATE, and SET SHARED commands to keep the state of each block up to date. These commands are defined in Table 4–10.

Table 4–10 System-Initiated Interface Commands (Write Invalidate Protocol)

Command	cmd_h <3:0>	Description
NOP	0000	The NOP command is driven by the owner of the **cmd_h<3:0>** bus when it has no tasks queued.
INVALIDATE	0010	Remove the block. When the system issues the INVALIDATE command, the 21164 probes its Scache. If the block is found, the 21164 responds with ACK/Scache and invalidates the block. If the block is not found, and the system does not contain a Bcache, the 21164 responds with a NOACK. If the system contains a Bcache, the system is assumed to have filtered all requests by using the duplicate tag store. Therefore, the block is assumed to be present in the Bcache. The 21164 responds with ACK/Bcache, and the block is changed to the invalid state without probing.
SET SHARED	0011	Block goes to the shared state. The SET SHARED command is used by the system to change the state of a block in the cache system to shared. The shared bit in the Scache is set if the block is present. The Bcache tag is written to the shared not dirty state. The 21164 assumes that this action is correct, because the system would have sent a READ DIRTY command if the dirty bit were set. If the block is found in the Scache, the 21164 responds with ACK/Scache. Otherwise, if the system contains a Bcache, the block is assumed to be in the Bcache, and the 21164 responds with ACK/Bcache. If the system does not contain a Bcache, and the block is not found in the Scache, the 21164 responds with NOACK.

(continued on next page)

4.10 System-Initiated Transactions

Table 4–10 (Cont.) System-Initiated Interface Commands (Write Invalidate Protocol)

Command	cmd_h <3:0>	Description
READ DIRTY	0101	Read a block; set shared. The READ DIRTY command probes the Scache to see if the requested block is present and dirty. If the block is not found, or if the block is clean, and the system does not contain a Bcache, the 21164 responds with NOACK. If the block is found and dirty in the Scache, the 21164 responds with ACK/Scache and drives the data on the **data_h<127:0>** bus. If the block is not found in the Scache, and the system contains a Bcache, the block is assumed to be in the Bcache. The 21164 responds with ACK/Bcache, indexes the Bcache to read the block, and changes the block status to the shared dirty state.
READ DIRTY/ INVALIDATE	0111	Read a block; invalidate. This command is identical to the READ DIRTY command except that if the block is present in the caches, it will be invalidated from the caches.

4.10.2.1 Alpha 21164 Responses to Write Invalidate Protocol Commands

The 21164 responses on **addr_res_h<1:0>** to write invalidate protocol commands are listed in Table 4–11.

Table 4–11 Alpha 21164 Responses on addr_res_h<1:0> to Write Invalidate Protocol Commands

Bcache	Scache	addr_res_h<1:0>
	INVALIDATE and SET SHARED Commands	
No Bcache	Scache_Miss	NOACK
No Bcache	Scache_Hit	ACK/Scache
Bcache_Hit/Miss	Scache_Hit/Miss	ACK/Bcache
	READ DIRTY and READ DIRTY/INVALIDATE Commands	
No Bcache	Scache_Miss	NOACK
No Bcache	Scache_Hit,Not Dirty	NOACK
No Bcache	Scache_Hit,Dirty	ACK/Scache
Bcache	Scache_Hit,Dirty	ACK/Scache
Bcache	Scache_Miss	ACK/Bcache

The signal **addr_res_h<2>** allows a system without a duplicate tag store to determine if a block is present in the Scache or lock register. The system logic can use this information to correctly assert **tag_shared_h** in a multiprocessor system.

The 21164 responds to the READ, FLUSH, READ DIRTY, SET SHARED and READ DIRTY/INVALIDATE commands on **addr_res_h<2>**, as listed in Table 4–12.

Table 4–12 Alpha 21164 Responses on addr_res_h<2> to 21164 Commands

Scache	Lock Register	addr_res_h<2>
Miss	Miss	0
Miss	Hit	1
Hit	Miss	1
Hit	Hit	1

Table 4–13 presents the 21164 best-case response time to system commands in a write invalidate protocol system.

4.10 System-Initiated Transactions

Table 4–13 Alpha 21164 Minimum Response Time to Write Invalidate Protocol Commands

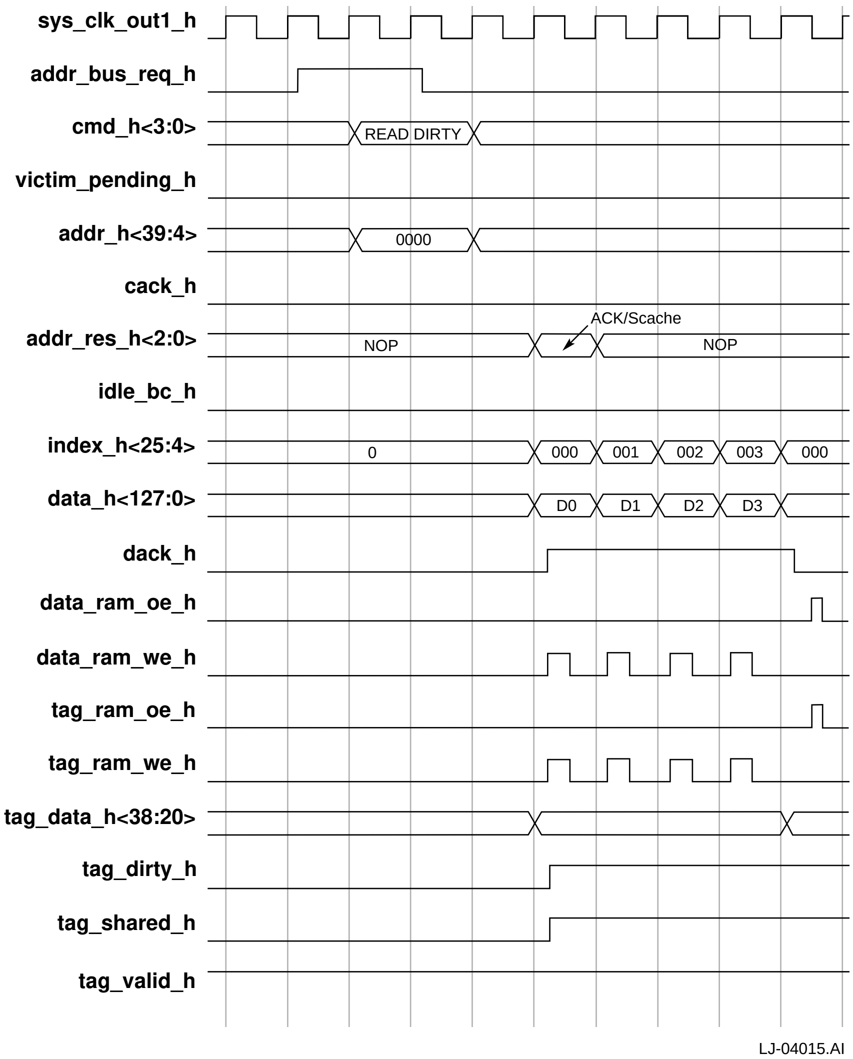
Cache Status	Response	Number of sys_clk_out1_h,l Cycles
No Bcache	NOACK	8 CPU cycles rounded up to next **sys_clk_out1_h,l** cycles
No Bcache	ACK/Scache	12 CPU cycles rounded up to next **sys_clk_out1_h,l** cycles
Bcache	NOACK, ACK/Scache, ACK/Bcache	10 CPU cycles rounded up to next **sys_clk_out1_h,l** cycles

4.10.2.2 READ DIRTY and READ DIRTY/INVALIDATE

The READ DIRTY command is used to read modified data from the cache system. The block status changes from $\mathrm{DIRTY}, \overline{\mathrm{SHARED}}$ to $\mathrm{DIRTY}, \mathrm{SHARED}$. Figure 4–25 shows the timing of a READ DIRTY command that hits in the Scache. The 21164 drives data starting at the rising edge of the sysclk that drives **addr_res_h<2:0>**. The Bcache data and tag state are updated as each INT16 is passed to the system. If the data had not been found in the Scache, the Bcache would have been indexed on the rising edge of the syclk that drove **addr_res_h<2:0>**. The index would advance to the next INT16 data as **dack_h** pulses arrive. The Bcache tag would be written with the updated state during the second INT16 data cycle.

The READ DIRTY/INVALIDATE command is identical to the READ DIRTY command except that the block is changed to $\overline{\mathrm{VALID}}$ rather than to SHARED.

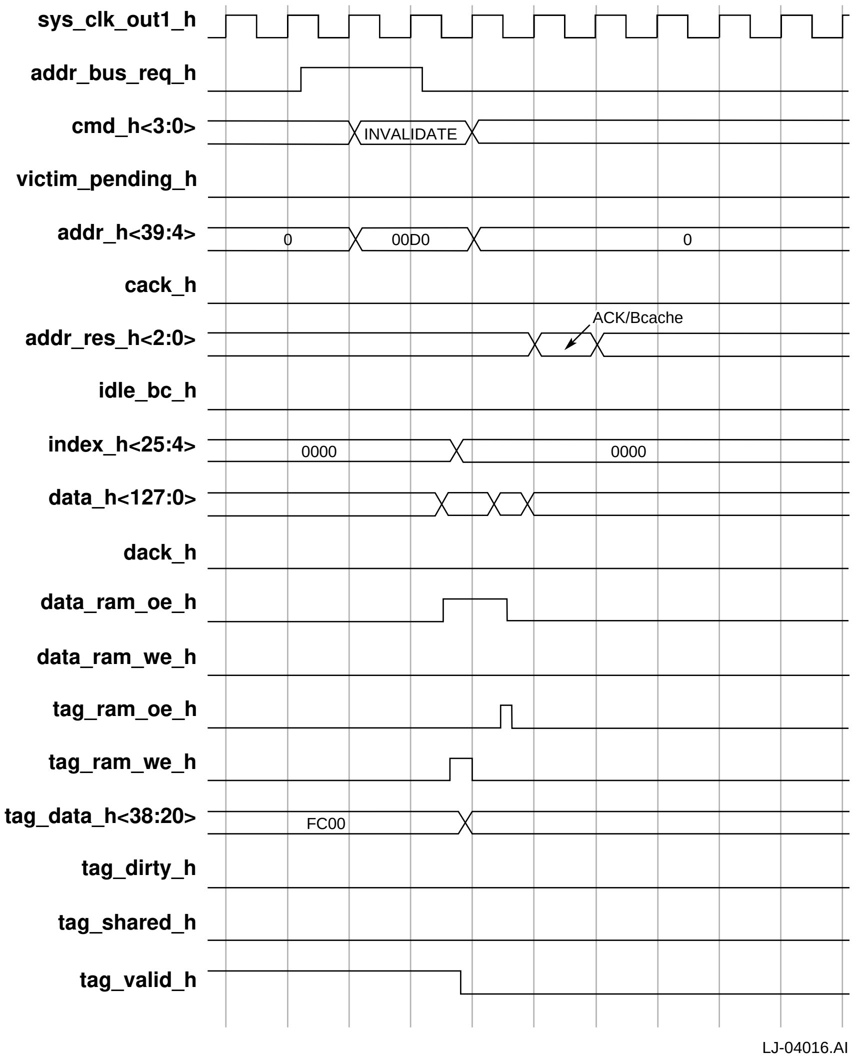
Figure 4–25 READ DIRTY Timing Diagram (Scache Hit)



4.10.2.3 INVALIDATE

The INVALIDATE command can be used to remove a block from the cache system. Unlike the FLUSH command, any modified data will not be read. The Scache is probed and invalidated if the block is found. The Bcache is invalidated without probing. Figure 4–26 shows the timing of an INVALIDATE transaction.

Figure 4–26 INVALIDATE Timing Diagram—Bcache Hit

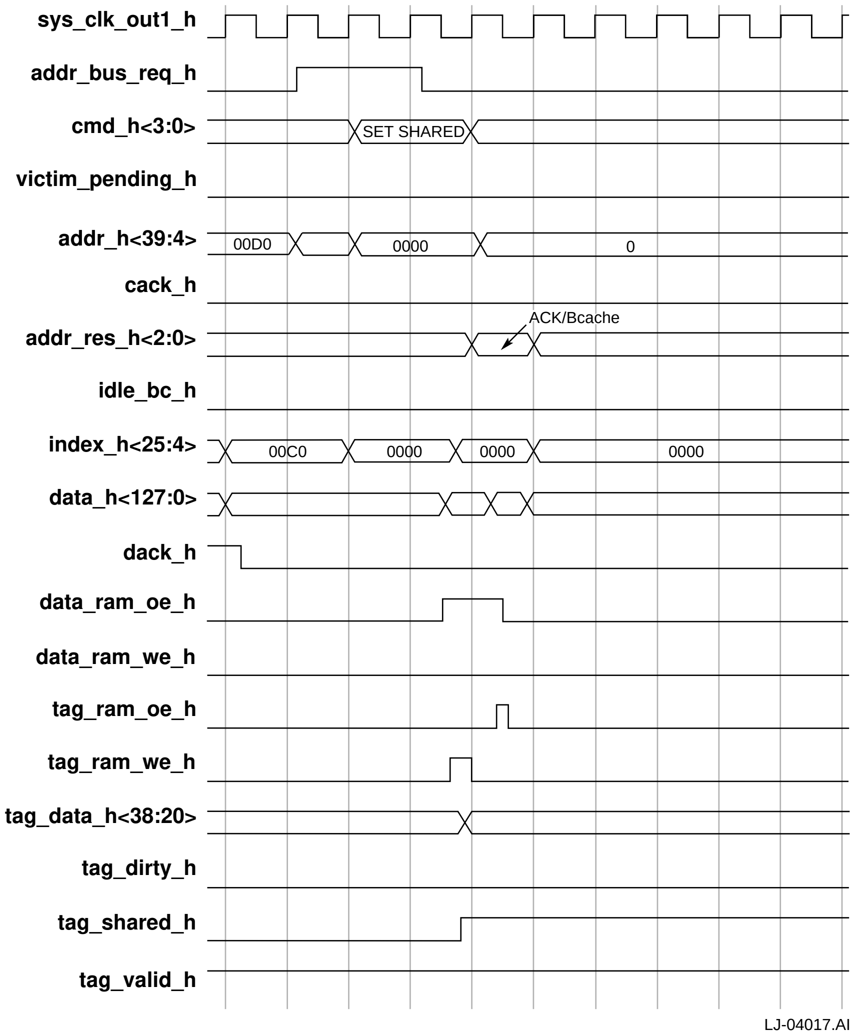


4.10 System-Initiated Transactions

4.10.2.4 SET SHARED

When the 21164 receives a SET SHARED command, it probes the Scache and changes the state of the block to SHARED if it is found. The 21164 ''assumes'' that the block is in the Bcache and writes the state of the tag to $\mathrm{SHARED}, \overline{\mathrm{DIRTY}}$. Figure 4–27 shows the timing of a SET SHARED command.

Figure 4–27 SET SHARED Timing Diagram



4.10.3 Flush-Based Cache Coherency Protocol Commands

All 21164-based systems that use the flush protocol are expected to use the READ and FLUSH commands defined in Table 4–14 to maintain cache coherency.

Table 4–14 System-Initiated Interface Commands (Flush Protocol)

Command	cmd_h <3:0>	Description
NOP	0000	The NOP command is driven by the owner of the **cmd_h<3:0>** bus when it has no tasks queued.
FLUSH	0001	Remove block from caches; return dirty data. The FLUSH command causes a block to be removed from the 21164 cache system. If the block is not found, the 21164 responds with NOACK. If the block is found and the block is clean, the 21164 responds with NOACK. The block is invalidated in the Dcache, Scache, and Bcache. If the block is found and is dirty, the 21164 responds with ACK/Scache or ACK/Bcache. If the data is found dirty in the Scache, it is driven at the interface in the same sysclk as the ACK/Scache. If the data is found dirty in the Bcache, the Bcache read starts on the same sysclk as ACK. The block is invalidated in the Dcache, Scache, and Bcache.
READ	0100	Read a block. The READ command probes the Scache and Bcache to see if the requested block is present. If the block is present, the 21164 responds with ACK/Scache or ACK/Bcache. If the data is in Scache, the data is driven on the **data_h<127:0>** bus in the same sysclk as the ACK. If the data is in the Bcache, a Bcache read operation begins in the same sysclk as the ACK. If the block is not present in either cache, the 21164 responds with a NOACK on **addr_res_h<1:0>**.

4.10.3.1 Alpha 21164 Responses to Flush-Based Protocol Commands

The system responds to flush-based protocol commands on **addr_res_h<1:0>**, as shown in Table 4–15.

Table 4–15 Alpha 21164 Responses to Flush-Based Protocol Commands

READ and FLUSH Commands		
Bcache Status	**Scache Status**	**21164 Response**
No Bcache	Scache_Miss	NOACK
No Bcache	Scache_Hit,Not Dirty	NOACK
No Bcache	Scache_Hit,Dirty	ACK/Scache
Bcache_Miss	Scache_Miss	NOACK
Bcache_Hit	Scache_Hit,Dirty	ACK/Scache
Bcache_Hit, Not Dirty	Scache_Miss/Hit, Not Dirty	NOACK
Bcache_Hit,Dirty	Scache_Miss	ACK/Bcache

The signal **addr_res_h<2>** allows a system without a duplicate tag store to determine if a block is present in the Scache or lock register. The system logic can use this information to correctly assert **tag_shared_h** in a multiprocessor system.

The 21164 responds to the READ, FLUSH, READ DIRTY, SET SHARED, and READ DIRTY/INVALIDATE commands on **addr_res_h<2>**, as listed in Table 4–16.

Table 4–16 Alpha 21164 Responses on addr_res_h<2> to 21164 Commands

Scache	Lock Register	addr_res_h<2>
Miss	Miss	0
Miss	Hit	1
Hit	Miss	1
Hit	Hit	1

Table 4–17 presents the 21164 best-case response time to system commands in a flush protocol system.

4.10 System-Initiated Transactions

Table 4–17 Minimum 21164 Response Time to Flush Protocol Commands

Cache Status	Response	Number of sys_clk_out1_h,l Cycles
No Bcache	NOACK	8 CPU cycles rounded up to next **sys_clk_out1_h,l** cycles
No Bcache	ACK/Scache	12 CPU cycles rounded up to next **sys_clk_out1_h,l** cycles
Bcache	NOACK, ACK/Scache, ACK/Bcache	10 CPU cycles plus <BC_RD_SPD> rounded up to next **sys_clk_out1_h,l** cycles

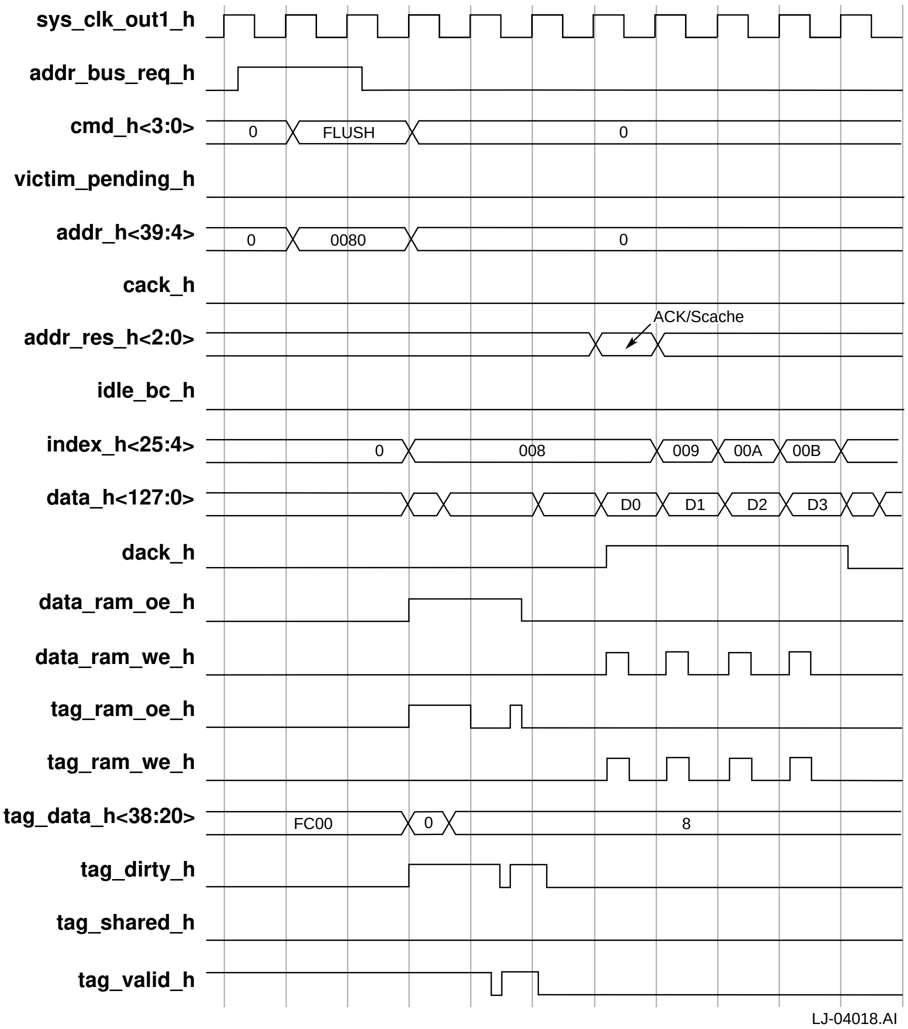
4.10.3.2 FLUSH

The FLUSH command is used to remove blocks from the 21164 cache system. Figure 4–28 shows the timing of a FLUSH transaction.

If the block is DIRTY, the 21164 will respond with an ACK and the system must read data from the cache, using **dack_h** to control the rate at which data is supplied, and write it to memory.

In the timing diagram shown in Figure 4–28, the cache block state changes from DIRTY, $\overline{\mathrm{SHARED}}$, VALID to $\overline{\mathrm{DIRTY}}$, $\overline{\mathrm{SHARED}}$, $\overline{\mathrm{VALID}}$. When the block state changes to $\overline{\mathrm{VALID}}$, the state of SHARED and DIRTY does not matter.

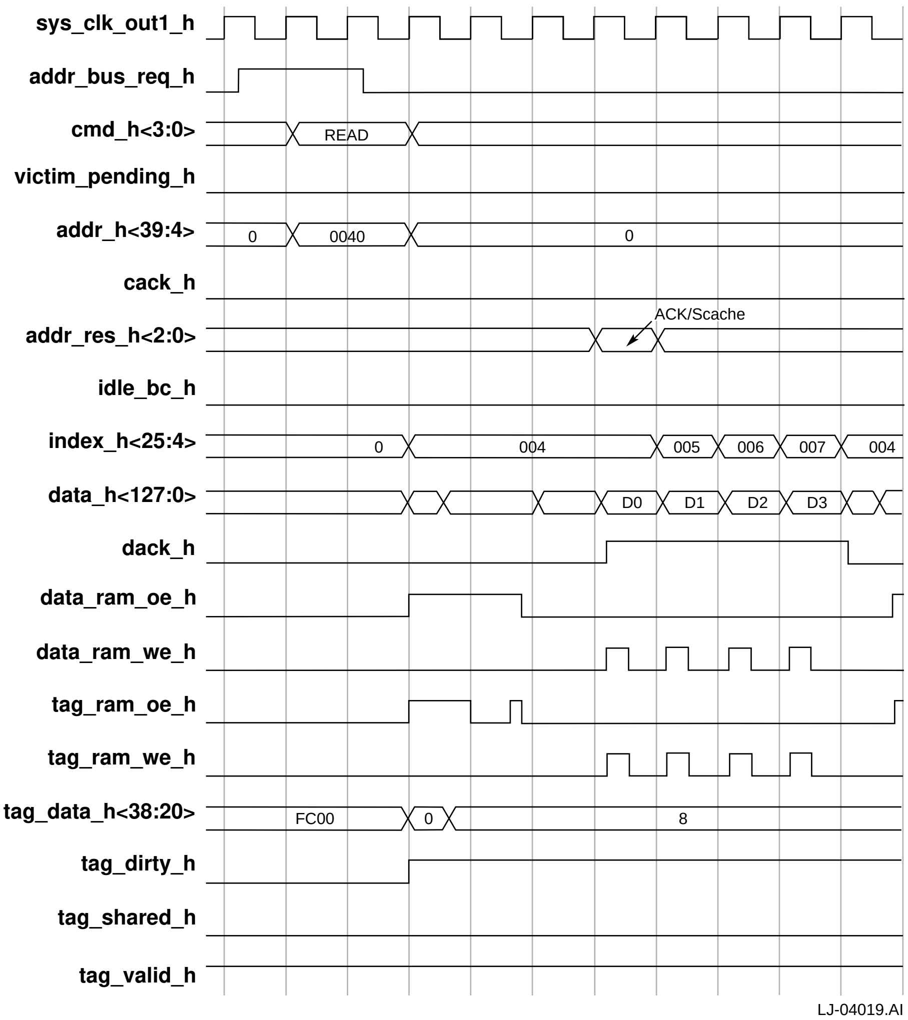
Figure 4–28 FLUSH Timing Diagram (Scache Hit)



4.10.3.3 READ

The READ command is used by the system to read DIRTY data from the 21164. The tag control status does not change. Figure 4–29 shows the timing and tag control status of a READ transaction.

Figure 4–29 READ Timing Diagram (Scache Hit)



4.11 Data Bus and Command/Address Bus Contention

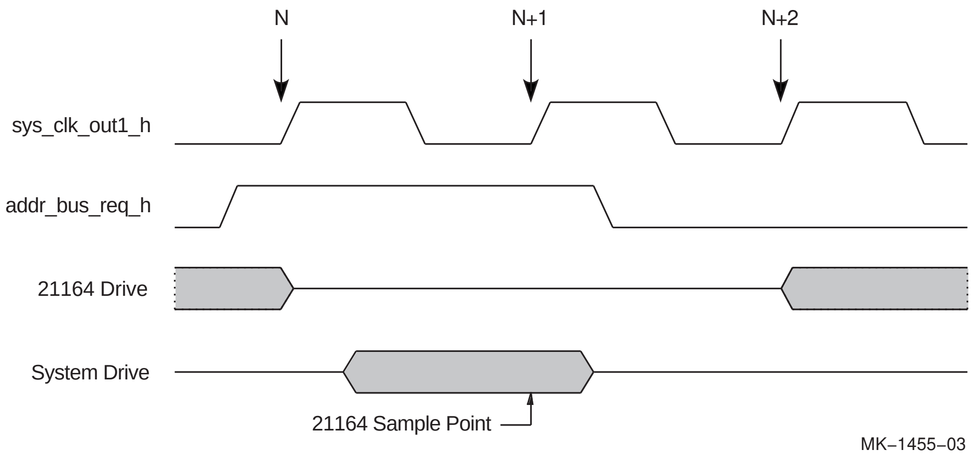
The data bus is composed of **data_h<127:0>** and **data_check_h<15:0>**. The command/address bus is composed of **cmd_h<3:0>**, **addr_h<39:4>**, and **addr_cmd_par_h**.

The following sections describe situations that have contention for use of the data bus or contention for use of the command/address bus.

4.11.1 Command/Address Bus

Figure 4–30 shows the 21164 and the system alternately driving the command/address bus. If signal **addr_bus_req_h** is asserted at the rising edge of sysclk N, the next cycle on the command/address bus belongs to the system. The 21164 turns off its drivers at the rising edge of sysclk N. While the system must turn on its drivers between sysclk N and sysclk N+1, it must ensure that the drivers do not turn on before the 21164 drivers turn off. The 21164 samples the state of the command/address bus at the end of sysclk N+1. If **addr_bus_req_h** remains asserted, the system should continue to drive the command/address bus.

Figure 4–30 Driving the Command/Address Bus



To pass control of the command/address bus back to the 21164, the system should turn off its drivers during a sysclk and deassert **addr_bus_req_h**. The 21164 does not sample the state of the bus if **addr_bus_req_h** is deasserted. The 21164 drives the command/address bus at the rising edge of sysclk N+2.

On every 21164 sample point, the **cmd_h<3:0>**, **addr_h<39:4>**, and **addr_cmd_par_h** signals must be valid, and the parity must be correct unless BC_CONTROL<DIS_SYS_PAR> is set. If DIS_SYS_PAR is clear, **addr_cmd_par_h** must be valid for the address and command, even when the address is irrelevant, because the system is driving a NOP on **cmd_h<3:0>**.

4.11.2 Read/Write Spacing—Data Bus Contention

The data bus, **data_h<127:0>**, can be driven by the 21164, the Bcache array, or the system.

In the case of private Bcache write operations followed by private Bcache read operations, the 21164 stops driving the data bus well in advance of the Bcache turning on.

For private Bcache read operations followed by private Bcache write operations, the 21164 inserts a programmable number of CPU cycles between the read and the write operation. This allows time for the Bcache drivers to turn off before the 21164 data drivers are turned on.

Note

This rule also applies to WRITE BLOCK, WRITE BLOCK LOCK, READ, READ DIRTY, READ DIRTY/INV, and FLUSH commands.

4.11.3 Using idle_bc_h and fill_h

The 21164 uses the **idle_bc_h** and **fill_h** signals to fill data into the Scache, the Bcache, or both. The system must assert the **idle_bc_h** signal early enough to ensure that the 21164 completes any Bcache transaction it might have started while waiting for the fill data.

Signal **fill_h** is asserted a fixed number of sysclk cycles before the start of a fill transaction.

At the end of the fill, the 21164 waits five CPU cycles before starting a read or write operation. This time should allow the system to turn off its drivers. If, in practice, this is not enough time, the system may assert **data_bus_req_h** to gain additional cycles.

Calculating Time to Assert idle_bc_h

The equations for calculating length of time to assert **idle_bc_h** are:

```
read_hit_idle  = 2 + (block_size/16) *  BC_RD_SPD +
                 tristate_ram_turn_off - 3 *  wave_pipelining;

read_miss_idle = 6 + BC_RD_SPD + Sysclk_ratio + tristate_RAM_turn_off;

write_idle     = 4 + (block_size/16) *  BC_WRT_SPD + tristate_21164_turn_off;
```

When using these equations, the turn-off times should be expressed as an integer number of CPU clock periods. Take the largest of the three times and then round up to the next sysclk boundary.

When determining the tristate turn-off times, if the system will not turn on its drivers for some number of nanoseconds after the 21164 starts driving Bcache **index_h<25:4>**; this time can be used to reduce the tristate_turn_off time.

For example if the sysclk ratio is 6 (the caches use a 64-byte block size), Bcache read/write speed is 5, with no wave pipelining, 2 cycles for tristate read, 0 cycles for tristate_write, then the equations would work out to:

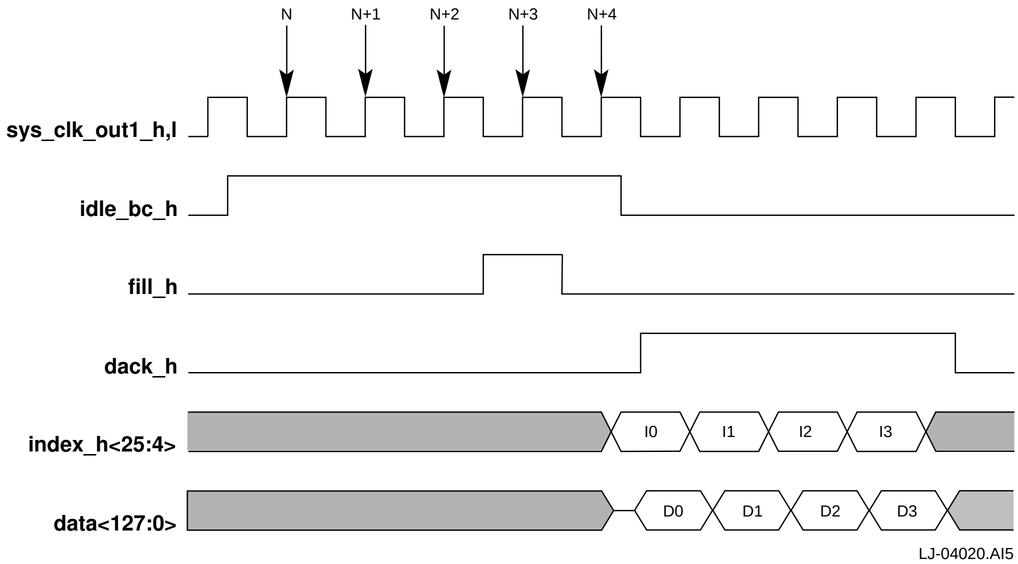
```
read_hit_idle  = 2 + (64/16) *  5 + 2 - 3 *  0 = 24

read_miss_idle = 6 + 5 + 6 + 2 = 19

write_idle     = 4 + (64/16) *  5 + 0 = 24

  Maximum of (24/6), (19/6), (24/6) = 4
```

In this example wave_pipelining = 0 makes only the partial product zero, not the entire equation.

If the 21164 samples **idle_bc_h** asserted at sysclk edge N, the earliest time that the system can allow the 21164 to sample **fill_h** asserted is at sysclk edge N+3. The 21164 drives **index_h<25:4>** to fill the Bcache on sysclk edge N+4.

Systems without a Bcache are not required to assert **idle_bc_h** to use the **data_bus_req_h** signal.

Figure 4–31 Example of Using idle_bc_h and fill_h



Minimum idle_bc_h time

If the system contains a Bcache, and the write ratio of the Bcache is greater than or equal to twice the sysclk ratio, then the minimum **idle_bc_h** assertion time is two sysclk cycles.

For example, if the Bcache write speed is 10, and the sysclk ratio is 4, then any assertion of **idle_bc_h** must be for two or more sysclk cycles.

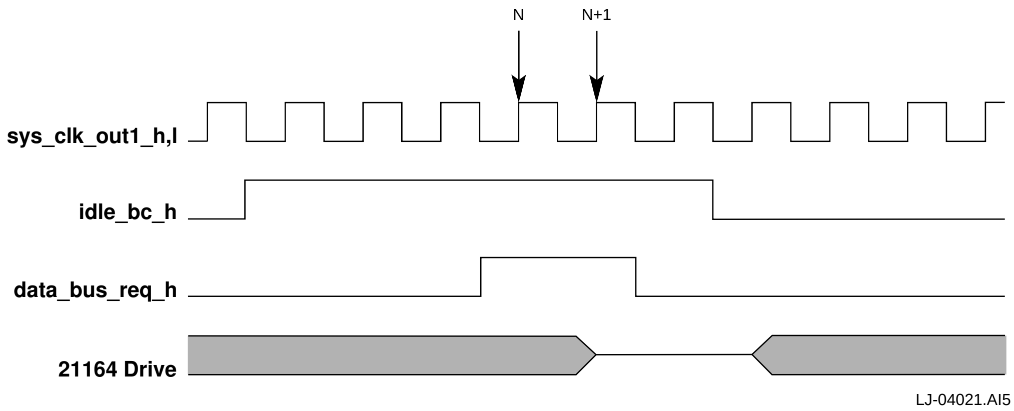
4.11.4 Using data_bus_req_h

The signal **data_bus_req_h** can be used along with the **idle_bc_h** signal to prevent the 21164 and the Bcache from driving the data bus. In general, the system should not need to use this feature but it may be useful if the system places other devices on the data bus.

To gain control of the data bus, the system must ensure that the Bcache is idle by asserting **idle_bc_h** for the required time. It can then assert **data_bus_req_h**. If **data_bus_req_h** is received asserted at the rising edge of sysclk N, the 21164 stops driving the bus on the rising edge of sysclk N+1.

To return the bus to the 21164, the system should deassert **data_bus_req_h** and then deassert **idle_bc_h** on the next sysclk.

Figure 4–32 Using data_bus_req_h



4.11.5 Tristate Overlap

The **addr_h<39:4>**, **cmd_h<3:0>**, **data_h<127:0>**, and **tag_data_h<38:20>** buses must be operated in such a way that no more than one driver may drive the bus at a time. This section describes particular cases where tristate overlap may be a problem that needs to be corrected using features described in previous sections.

The ''owner'' of each bus must drive the bus to some value for each cycle. Tristate drivers in the 21164 turn on and off very fast (in the 0.5-ns to 1.0-ns range). At the other end of the range, SRAM memory devices turn on and off slowly (in the 7.0-ns to 10.0-ns range). Generally, system drivers fall somewhere in the middle.

4.11.5.1 READ or WRITE to FILL

The time required to tristate the 21164 drivers at the end of a WRITE command, or the Bcache drivers at the end of a READ command is part of the **idle_bc_h** equation.

4.11.5.2 BCACHE VICTIM to FILL

The time to turn off the Bcache drivers at the end of a BCACHE VICTIM is fixed by the 21164 design. The system must allow for this time before starting a FILL.

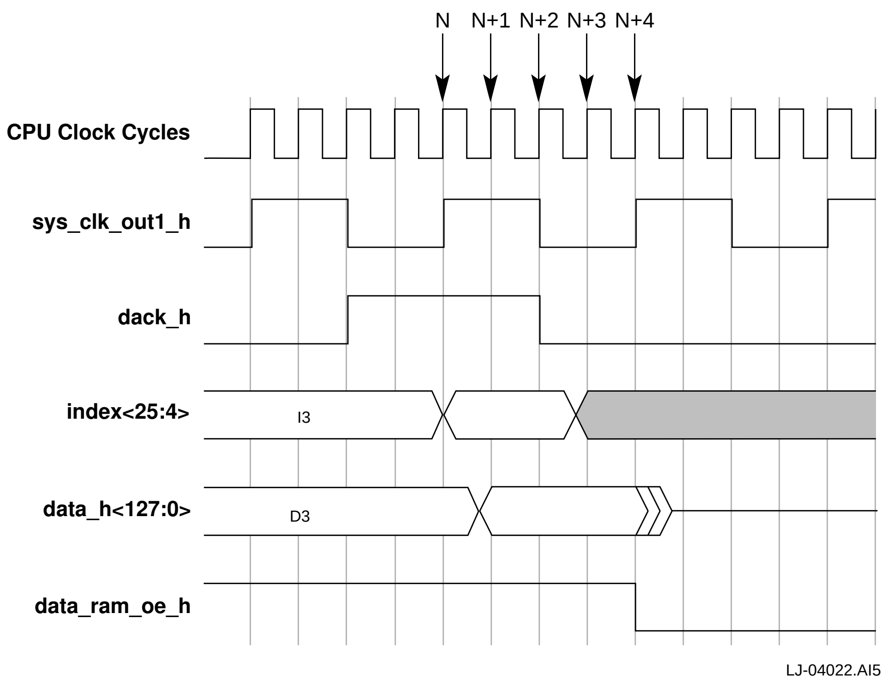
There are two READ MISS with victim cases to consider. In one case, the READ MISS operation will be completed first because the system logic contains a victim buffer. In the other case the READ MISS operation will be completed second because the system logic does not have a victim buffer.

4.11 Data Bus and Command/Address Bus Contention

READ MISS Completed First—Victim Buffer

The final **dack_h** will be sampled by the 21164 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, then **data_ram_oe_h** will deassert at the rising edge of CPU clock N+4.

Figure 4–33 READ MISS Completed First—Victim Buffer

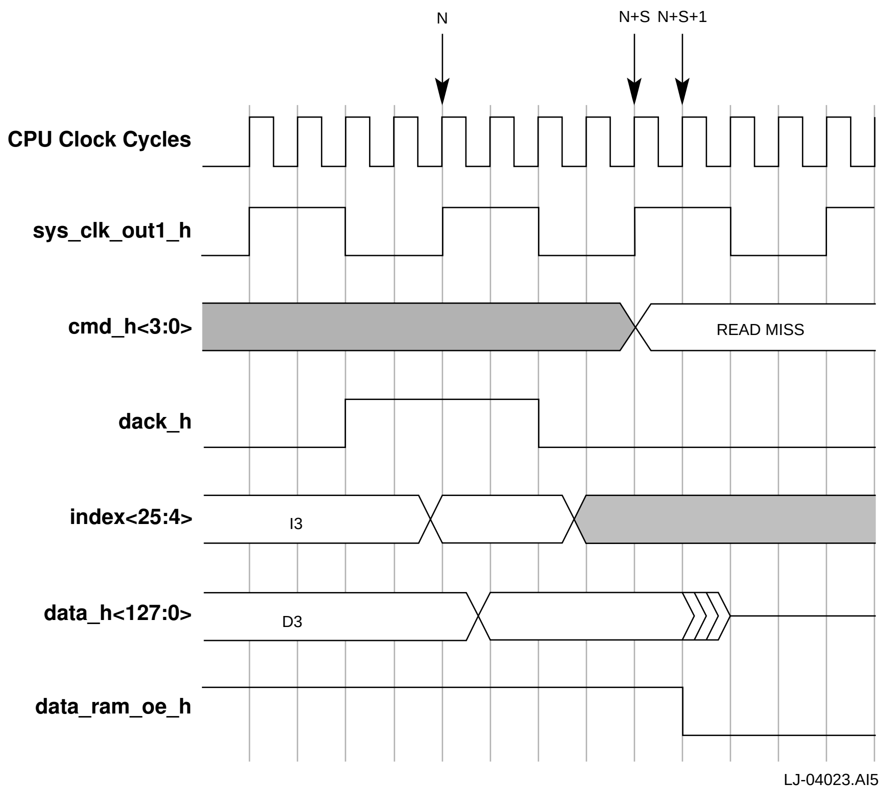


4.11 Data Bus and Command/Address Bus Contention

READ MISS Second—No Victim Buffer

The final **dack_h** will be sampled by 21164 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, then the READ MISS command will arrive on the next sysclk edge, and the **data_ram_oe_h** will deassert at the rising edge of CPU clock N+S+1, where S is the sysclk ratio. If the sysclk ratio is 3, it will take an extra sysclk to send the READ MISS command, so the **data_ram_oe_h** will deassert at N+2S+1.

Figure 4–34 READ MISS Second—No Victim Buffer

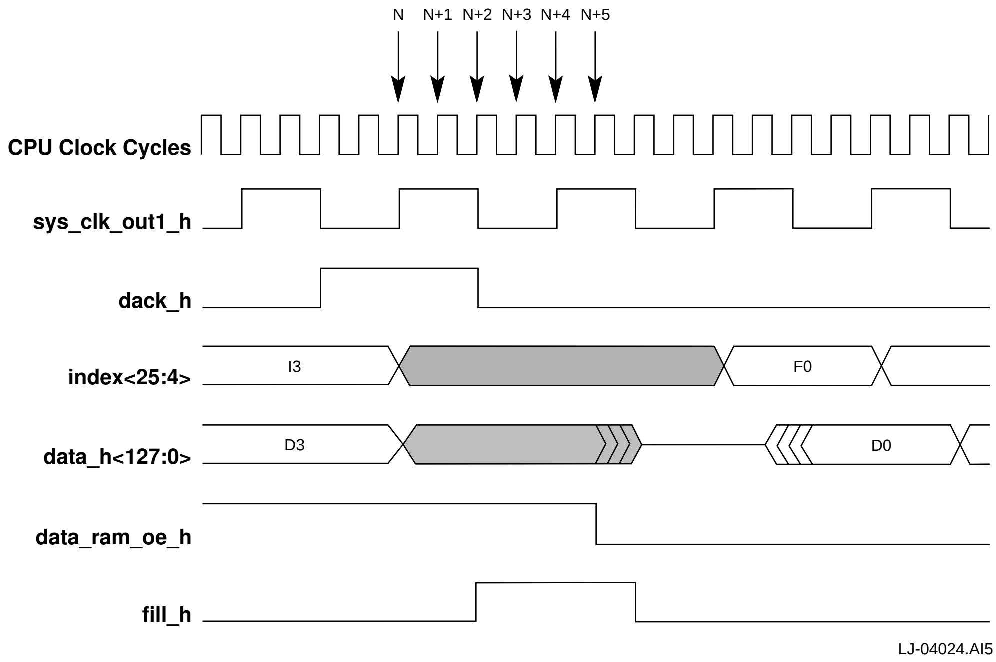


4.11.5.3 System Bcache Command to FILL

At the end of a system command that uses the Bcache, the system must provide enough time for the Bcache drivers to turn off before returning any fill data.

The final **dack_h** will be sampled by the 21164 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, **data_ram_oe_h** will deassert at the rising edge of CPU clock N+5.

Figure 4–35 System Command to FILL Example 1

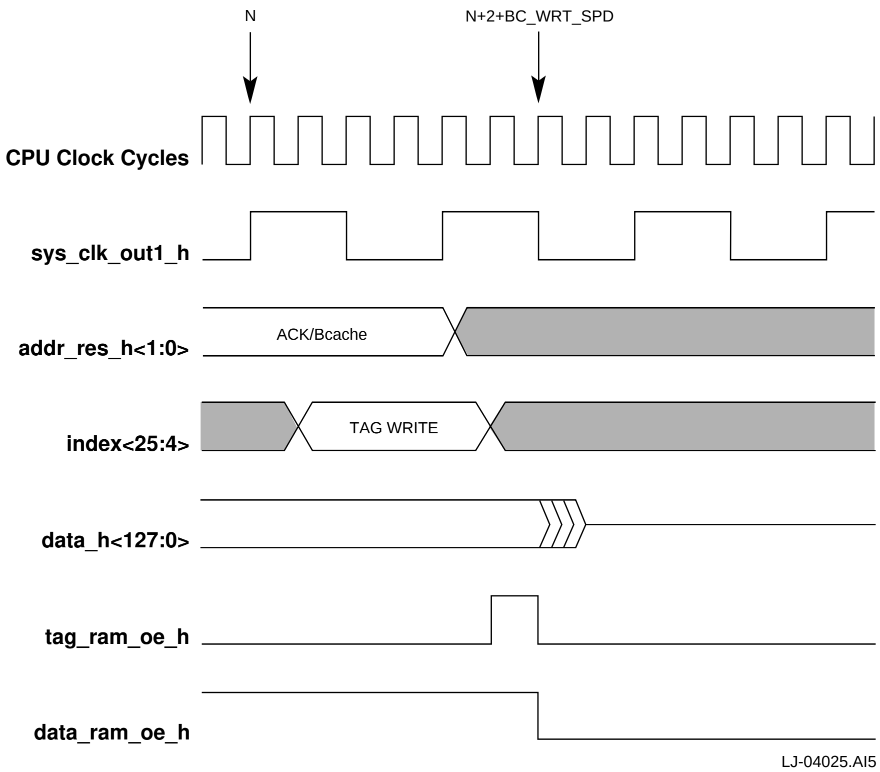


A side effect of this is the earliest assertion of **fill_h** after a system command. The system must allow time for **data_ram_oe_h** to turn off and the RAMs to stop driving the bus before the system drives the fill data.

If the system command was a SET SHARED or an INVALIDATE command, the system must allow time for the 21164 to complete the Bcache tag write operation and then for the drivers to turn off before driving the **tag_shared_h**, **tag_dirty_h**, and **tag_ctl_par_h** lines.

The 21164 begins the tag write operation one CPU cycle after the response is sent to the system. The write transaction will take BC_WRT_SPD cycles to complete. During the write transaction, **data_ram_oe_h** will be asserted but not **tag_ram_oe_h**. At the end of the write transaction, **tag_ram_oe_h** will pulse for one CPU cycle, then both will go off. Refer to Figure 4–36 if the response is driven at the rising edge of CPU clock N, then **data_ram_oe_h** will fall at N+2+BC_WRT_SPD, or N+6 for a 4-cycle write speed.

Figure 4–36 System Command to FILL Example 2

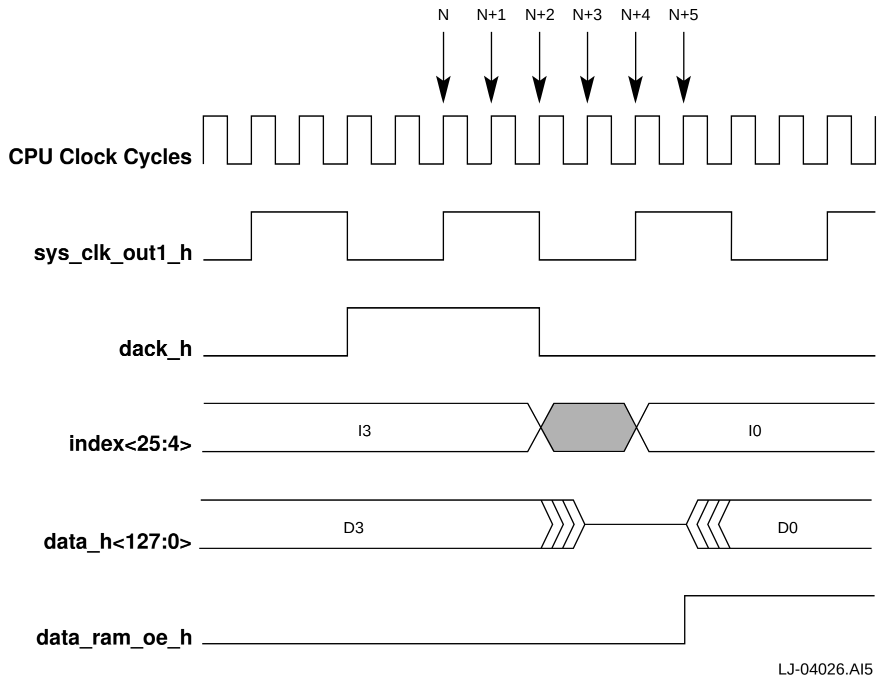


4.11.5.4 FILL to Private Read or Write Operation

At the end of the fill, the 21164 does not begin to drive the data bus until the fifth CPU cycle after the sysclk that loads the last **dack_h**. The 21164 does not assert **data_ram_oe_h** until the fifth cycle after the sysclk that loads the last **dack_h**.

Systems requiring more time to turn off their drivers must not send any more requests and must use **idle_bc_h** and **data_bus_req_h** at the end of the fill to stop 21164 requests.

Figure 4–37 FILL to Private Read or Write Operation



4.12 Alpha 21164 Interface Restrictions

This section lists restrictions on the use of 21164 interface features.

4.12.1 FILL Operations after Other Transactions

If the system has removed data from the 21164 with any of the system commands, or completed a WRITE_BLOCK, or removed a Bcache victim from the Bcache, and wants to follow any of these transactions with a FILL, then the earliest point the system can assert the **fill_h** signal is at the sysclk after the last assertion of **dack_h**. However, **fill_h** can be asserted at the sysclk with the last **dack_h** if the sysclk ratio is greater than 3.

FILL operations followed by FILL operations are special cases. FILL operations can be pipelined back-to-back so that 100% of the data bus bandwidth can be used.

4.12.2 Command Acknowledge for WRITE BLOCK Commands

When the 21164 requests a WRITE BLOCK or WRITE BLOCK LOCK operation, the system can acknowledge the data by asserting **dack_h** before asserting **cack_h**. The system must assert **cack_h** no later than the last assertion of **dack_h**.

4.12.3 Systems Without a Bcache

Systems without a Bcache must set a 64-byte block size.

If systems without a Bcache have an Scache duplicate tag store, they are required to maintain tags for the two blocks in the 21164 Scache victim buffer.

4.12.4 Fast Probes with No Bcache

If BC_CONTROL<BC_ENABLED>=0, then the 21164 processes system requests while other commands are being processed by the interface. The 21164 does not wait for the interface to become idle before processing system requests. This creates race conditions for the state of a cache block.

For example, if a certain block is being filled private-clean, and the system sends a SET SHARED command for the block, the SET SHARED command must be delayed until the fill completes and records the correct end state for the block, shared-clean. The system must avoid changing the state of a block that is in transit.

The restrictions are as follows:

- The system may not send a request to the 21164 for a block that has been filled until one sysclk after the last **dack_h** if the sysclk ratio is greater than 3.

- The system may not send a request to the 21164 for a block that has been filled until two sysclks after the last **dack_h** if the sysclk ratio is 3.
- The system may not send a request to the 21164 for a block that has completed a WRITE BLOCK command until one sysclk after the last **dack_h**.
- The system may not send a request to the 21164 for a block that has completed a SET DIRTY command until one sysclk after the **cack_h** for the SET DIRTY command.

If BC_CONTROL<BC_ENABLED>=1, all system requests are delayed to avoid race conditions.

4.12.5 WRITE BLOCK LOCK

A WRITE BLOCK LOCK transaction is caused by a store conditional instruction to I/O space. Two octawords of data are provided by the 21164, each requiring the system to assert **dack_h**. If the system asserts **dack_h** for the first octaword, and asserts **cack_h** and **cfail_h** together, the 21164 hangs.

If **dack_h**, **cack_h**, and **cfail_h** are asserted for the second INT16 of data, the write operation will be failed correctly.

If **cack_h** and **cfail_h** are asserted at any time without asserting **dack_h**, the write operation will be failed correctly.

4.13 Alpha 21164/System Race Conditions

When certain sequences of transactions occur on the interface between the 21164, the Bcache and the system race conditions may occur. The rules for use of the interface by the 21164 and the system are listed in Section 4.13.1.

Examples of race conditions to be avoided are described and illustrated in Section 4.13.2 through Section 4.13.6.

4.13.1 Rules for 21164 and System Use of External Interface

This section goes over the rules for determining the order in which 21164 and system requests are allowed by the Cbox BIU. In general, the order allowed is determined by use of **cmd_h<3:0>**, **idle_bc_h**, and **fill_h**.

1. If **idle_bc_h** is not asserted and there are no valid requests in the BIU command buffer, then the BIU is free to perform any 21164 request.
2. If a FILL transaction is pending, the BIU only produces another READ MISS command, with a possible BCACHE VICTIM command. The BIU will not attempt any other command.
3. The assertion of **idle_bc_h**, or the sending of a system command other than NOP to the 21164, causes the BIU to idle. If the BIU has a command loaded in the pad ring, it removes the command and replaces it with a NOP command. The state of **cmd_h<3:0>** is unpredictable until the idle condition ends.
4. The idle condition ends when the 21164 receives a deasserted **idle_bc_h**, and the 21164 has responded to all the system commands that were sent.
5. The system must not assert **cack_h** during the idle condition.
6. There is one exception to rules 3, 4, and 5. If **idle_bc_h** or a system command arrives while the 21164 is reading the Bcache, and that read transaction turns into a READ MISS transaction, and it does not produce a victim, then the 21164 loads the miss into the pad ring. The system may assert **cack_h** for this READ MISS request at any time.
7. If **cack_h** is asserted at the same time as **idle_bc_h** or a valid system request, **cack_h** wins and the command is taken by the system. Signal **cack_h** should not be asserted if **idle_bc_h** has been asserted or a valid system command is under way.
8. A READ MISS with a BCACHE VICTIM transaction is treated as an atomic pair. The command order, READ MISS then BCACHE VICTIM or BCACHE VICTIM then READ MISS, is programmable. Either way, if the first command is acknowledged with **cack_h**, then both commands must be

acknowledged with **cack_h** and all the data acknowledged with **dack_h**, before the 21164 responds to any other request.

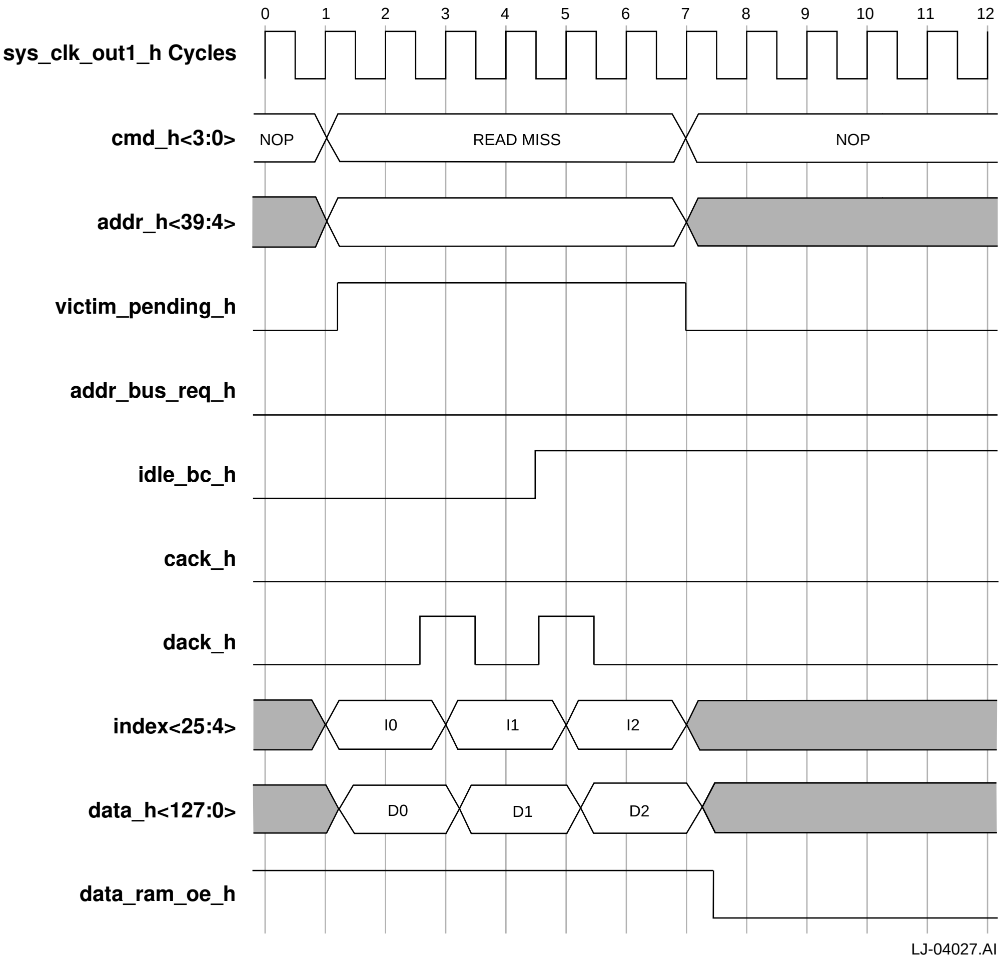
9. The **cack_h** acknowledgment for a WRITE BLOCK or BCACHE VICTIM transaction must be received by the 21164 with or before the last **dack_h** acknowledgment of the data. For WRITE BLOCK and BCACHE VICTIM transactions, it is possible to acknowledge all but the last data, and then decide to do something else.
10. For a READ MISS transaction, **cack_h** must be received with or before the last data acknowledgment (**dack_h**) for the requested FILL operation.
11. If a 21164 request is interrupted by an idle condition, the 21164 restarts the same command unless:
    a. A system request is received that changes the state of the block made by the original 21164 request.

       For example, if the 21164 is requesting a WRITE BLOCK and the system sends an INVALIDATE command to the same block, then the WRITE BLOCK command will not be restarted.
    b. If the system does not have a Bcache, and a WRITE BLOCK command to write an Scache victim back is interrupted, then the WRITE BLOCK command will not be restarted if a higher priority request arrives in the BIU.

4.13.2 READ MISS with Victim Example

In this example, the 21164 asserts a READ MISS command with a victim. The system asserts **dack_h** for two data cycles received from the Bcache and then asserts **idle_bc_h**. This causes the 21164 to remove the READ MISS command with victim pending. The 21164 reasserts the READ MISS and BCACHE VICTIM commands, if needed, at a later time.

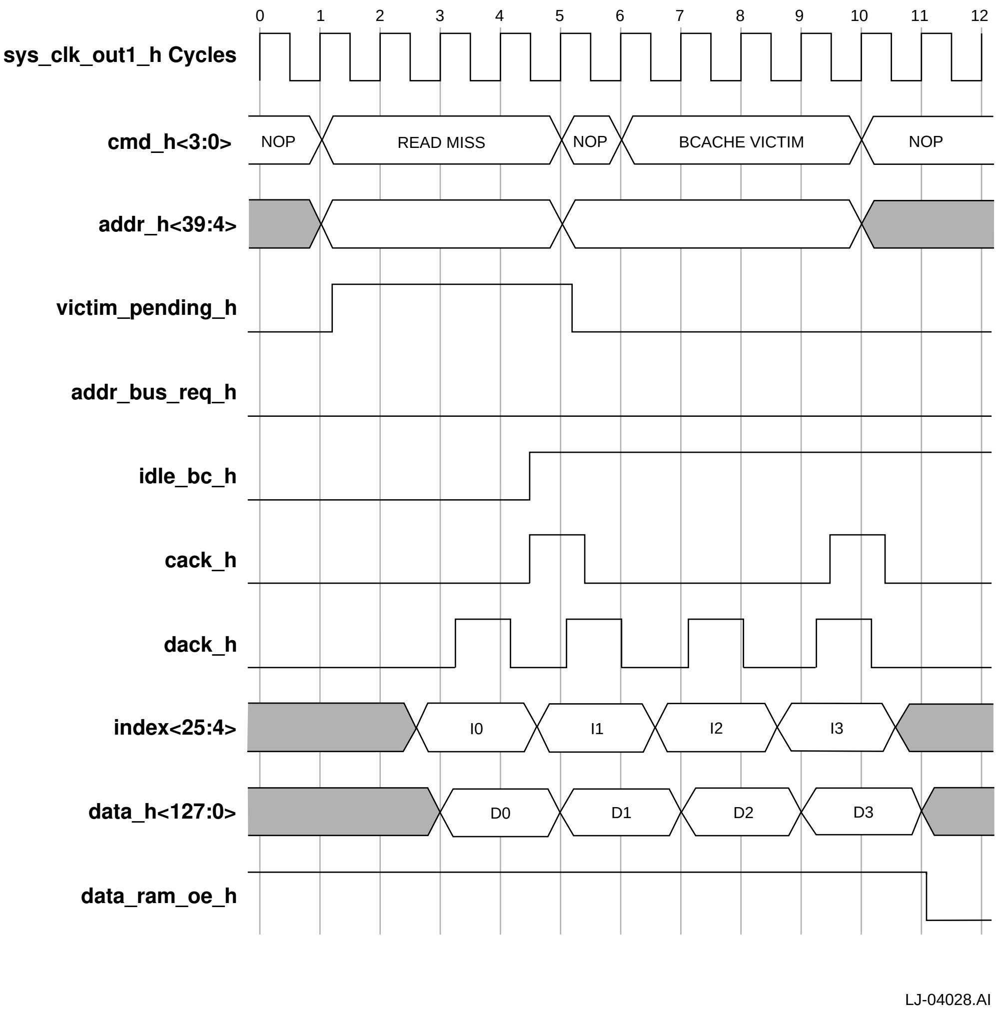
Figure 4–38 READ MISS with Victim Example



4.13.3 idle_bc_h and cack_h Race Example

In this example, **idle_bc_h** and **cack_h** are asserted in the same sysclk. The system takes the READ MISS and BCACHE VICTIM commands before doing anything else. The last **dack_h** meets the requirement that the **cack_h** arrive before or with the last **dack_h**.

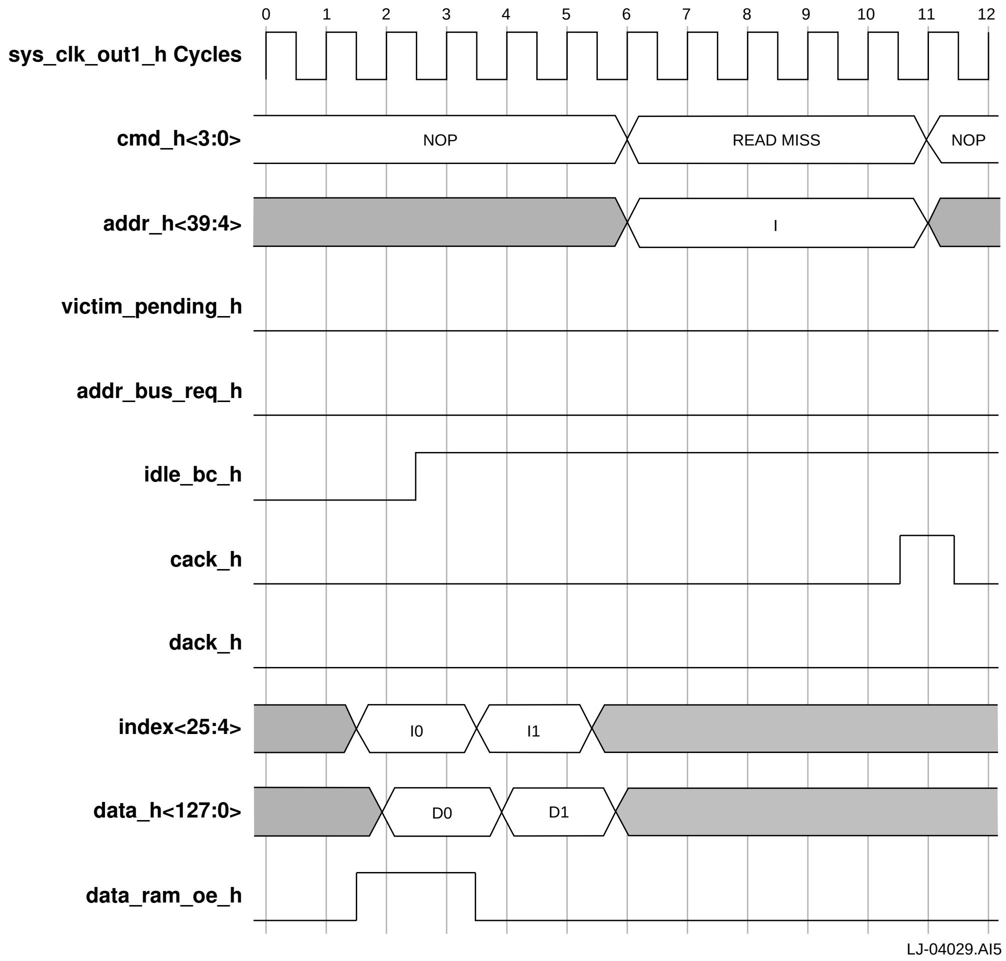
Figure 4–39 idle_bc_h and cack_h Race Example



4.13.4 READ MISS with idle_bc_h Asserted Example

In this example, the 21164 has started a Bcache read operation that misses. The signal **idle_bc_h** is asserted, but no victim was created, so the READ MISS request is loaded into the pad ring. The system then takes the request.

Figure 4–40 READ MISS with idle_bc_h Asserted Example

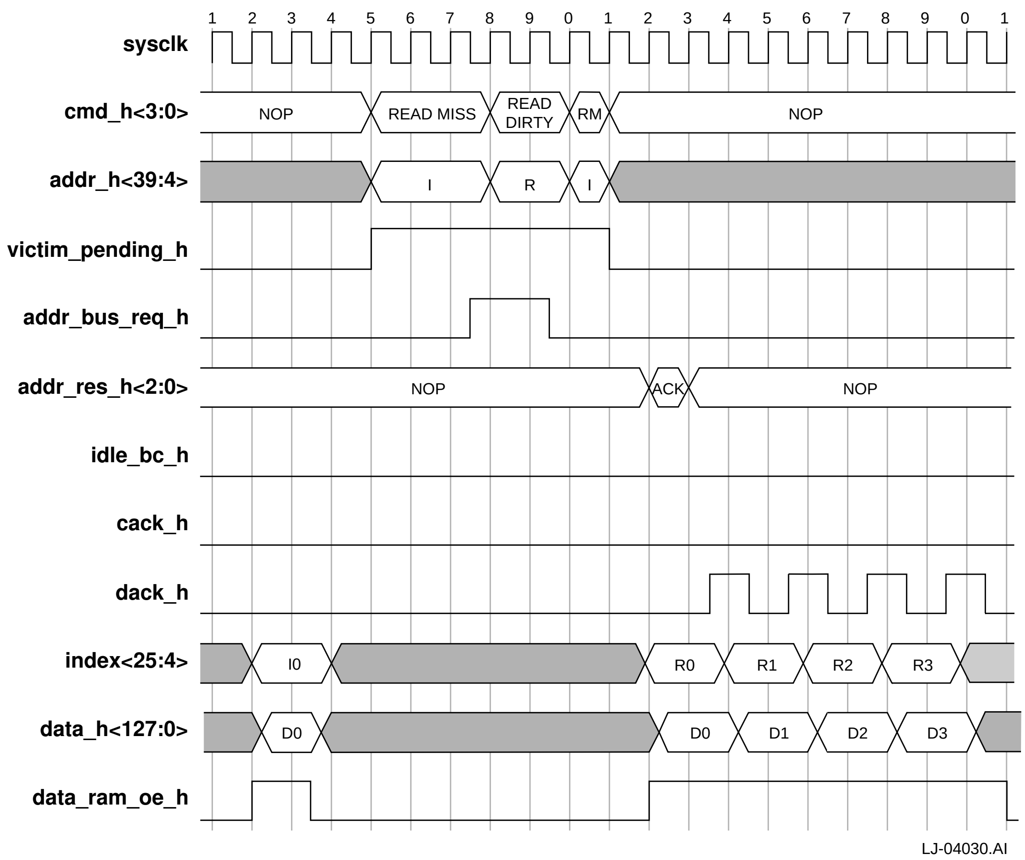


4.13.5 READ MISS with Victim Abort Example

In this example, the 21164 produces a READ MISS command with a victim and is waiting for the system to take it when the system takes the bus and requests a READ DIRTY transaction. The 21164 drives the READ MISS request for one more cycle after it gets command of the bus and then removes the request. The 21164 then responds to the READ DIRTY command and drives **index_h<25:4>** to read the Bcache. The 21164 restarting the Bcache read operation, requesting the read miss with victim, is not shown in the timing diagram. If the victim block was invalidated by the system request, the 21164 produces a clean READ MISS transaction.

4.13 Alpha 21164/System Race Conditions

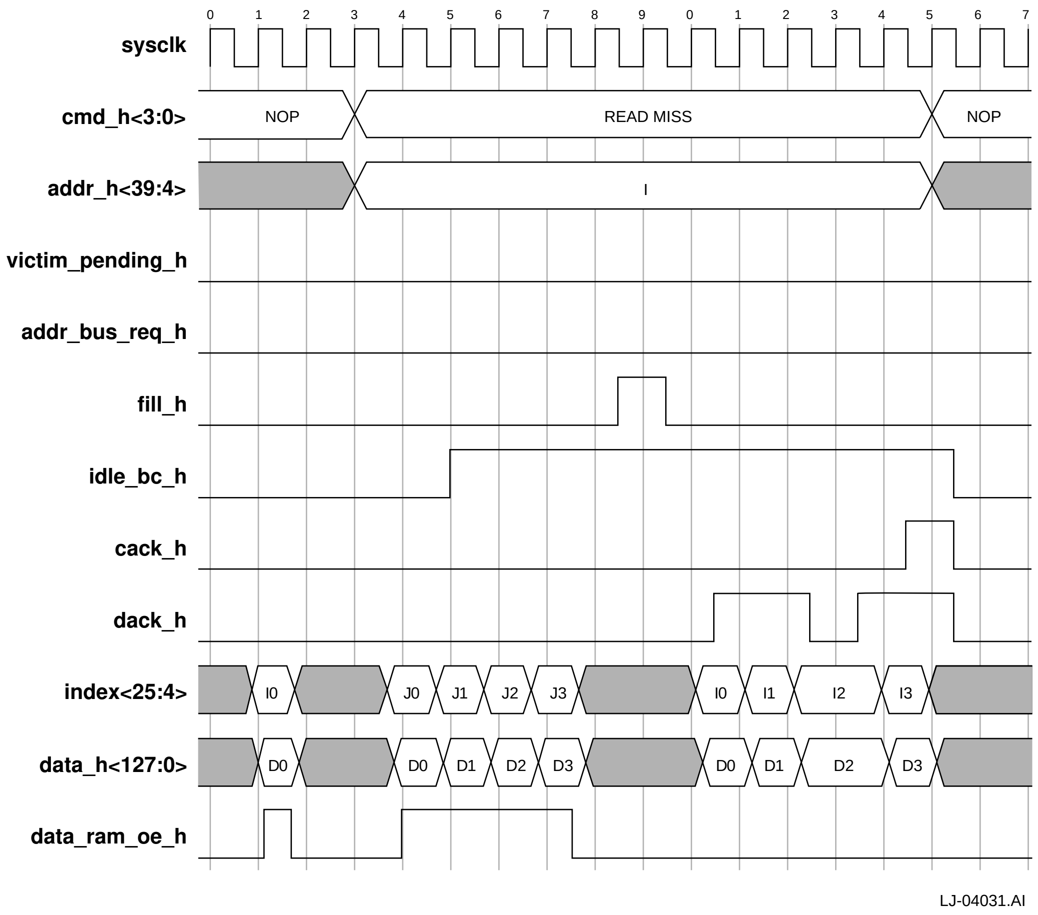
Figure 4–41 READ MISS with Victim Abort Example



4.13.6 Bcache Hit Under READ MISS Example

In this example, the 21164 produces a READ MISS transaction and requests a fill from the system. A Bcache hit to index j take places while waiting for the fill. The system then returns the requested data in two bursts, asserting **cack_h** at the same time as the last assertion of **dack_h**.

Figure 4–42 Bcache Hit Under READ MISS Example



4.14 Data Integrity, Bcache Errors, and Command/Address Errors

Mechanisms for ensuring that errors on data received by the 21164 from the Bcache, the system, or both are described in this section. Tag data and tag control errors are described. Command/address bus parity protection is also described.

4.14.1 Data ECC and Parity

The 21164 supports INT8 error correction code (ECC) for the external Bcache and memory system. ECC is generated by the CPU for each INT8 that is written into the Bcache. FILL data from the Bcache to the system is not checked for errors. The receiving node detects any ECC errors.

Uncorrected data from the Bcache or system is sent to the Dcache, and register files. If a correctable error is detected (single bit error) the machine traps and the fill is replayed with corrected data.

Double bit errors are detected. If the system indicates that the data should not be checked, then no checking or correcting is performed.

Each data bus cycle delivers one INT16 worth of data. ECC is calculated as ECC(data<063:000>) and ECC(data<127:064>). Figure 4–43 shows the code. Two IDT49C460 or AMD29C660 chips can be cascaded to produce this ECC code. A single IDT49C466 chip also supports this ECC code.

The code provides single bit correct, double bit detect, and all 1s and all 0s detect.

If the 21164 is in parity mode, it generates byte parity and places it on **data_check_h<15:0>** for write operations. Parity is checked for read operations. Parity for **data_h<7:0>** is driven on signal **data_check_h<0>** and so on.

4.14 Data Integrity, Bcache Errors, and Command/Address Errors

Figure 4–43 ECC Code

```
                11 1111 1111 2222 2222 2233 3333 3333 4444 4444 4455 5555 5555 6666 cccc cccc
     0123 4567 8901 2345 6789 0123 4567 8901 2345 6789 0123 4567 8901 2345 6789 0123 0123 4567

CB0  .111 .1.. 11.1 ..1. .111 .1.. 11.1 ..1. 1... 1.11 ..1. 11.1 1... 1.11 ..1. 11.1 1... ....
CB1  111. 1.1. 1.1. 1... 111. 1.1. 1.1. 1... 111. 1.1. 1.1. 1... 111. 1.1. 1.1. 1... .1.. ....
CB2  1..1 1..1 .11. .1.1 1..1 1..1 .11. .1.1 1..1 1..1 .11. .1.1 1..1 1..1 .11. .1.1 ..1. ....
CB3  11.. .111 ...1 11.. 11.. .111 ...1 11.. 11.. .111 ...1 11.. 11.. .111 ...1 11.. ...1 ....

CB4  ..11 1111 .... ..11 ..11 1111 .... ..11 ..11 1111 .... ..11 ..11 1111 .... ..11 .... 1...
CB5  .... .... 1111 1111 .... .... 1111 1111 .... .... 1111 1111 .... .... 1111 1111 .... .1..
CB6  1111 1111 .... .... .... .... 1111 1111 1111 1111 .... .... .... .... 1111 1111 .... ..1.
CB7  1111 1111 .... .... .... .... 1111 1111 .... .... 1111 1111 1111 1111 .... .... .... ...1
```

CB2 and CB3 are calculated for CDD parity (an odd number of 1s counting the CB).

CB0, CB1, CB4, CB5, CB6, and CB7 are calculated for EVEN parity (an even number of 1s counting the CB).

LJ-03461-TI0

The correspondence of data check bits to CB*n* is shown in Table 4–18.

Table 4–18 Data Check Bit Correspondence to CB*n*

	data_check_h	
CBn	**Upper 64 bits**	**Lower 64 bits**
CB0	<8>	<0>
CB1	<9>	<1>
CB2	<10>	<2>
CB3	<11>	<3>
CB4	<12>	<4>
CB5	<13>	<5>
CB6	<14>	<6>
CB7	<15>	<7>

For x4 RAMs, the following bit arrangement detects nibble errors:

```
CB0   CB1   CB5   CB6
CB2    D0    D4    D5
CB3   CB4    D7    D8
CB7    D2    D3   D11
 D1    D6   D10   D13
 D9   D14   D18   D21
D12   D16   D17   D22
D15   D19   D20   D23
D24   D25   D27   D30
D26   D28   D29   D31
D32   D34   D35   D37
D33   D36   D38   D40
D39   D41   D43   D46
D42   D44   D45   D47
D48   D50   D51   D53
D49   D52   D54   D56
D55   D57   D59   D62
D58   D60   D61   D63
```

4.14.2 Force Correction

Setting BC_CTL<4> (CORR_FILL_DAT), forces the 21164 to route fill data from the Bcache or memory through error correction logic before being driven to the Scache or Dcache. If the error is correctable, it is transparent to the 21164.

4.14.3 Bcache Tag Data Parity

The signal line **tag_data_par_h** is used to maintain parity over **tag_data_h<38:20>**. A Bcache tag data parity error is usually not recoverable.

A Bcache hit is determined based on the tag alone, not the tag parity bit. The Cbox records the Bcache probe address and the tag value read from the Bcache. A tag data parity error causes a trap to privileged architecture library code (PALcode), which handles the error condition.

4.14.4 Bcache Tag Control Parity

The signal **tag_ctl_par_h** is used to maintain parity over **tag_shared_h**, **tag_valid_h**, and **tag_dirty_h**. A Bcache tag control parity error is usually not recoverable.

A Bcache victim is processed according to the tag control status alone, not the tag control parity bit. The Cbox records the Bcache probe address and the tag control value read from the Bcache. A tag control parity error causes a trap to PALcode, which handles the error condition.

4.14.5 Address and Command Parity

The signal line **addr_cmd_par_h** is used to maintain odd parity over **addr_h<39:4>** and **cmd_h<3:0>**. These signals are driven by the 21164 or by the system, using the protocol described in Section 4.11.1.

4.14.6 Fill Error

The signal **fill_error_h** is asserted by the system to notify the 21164 that a fill error has occurred.

Systems in which a fill error timeout is not expected, such as a small system with fixed access time, it is likely that the 21164 internal Ibox timeout logic would detect a stall if the system fails to complete a fill transaction.

Systems in which a fill error timeout could occur should contain logic to detect fill timeouts and cleanly terminate the transaction with the 21164.

To properly terminate a fill in an error case, the **fill_error_h** line is asserted for one cycle and the normal fill sequence involving lines **fill_h**, **fill_id_h**, and **dack_h** is generated by the system.

Asserting **fill_error_h** forces a trap to the PALcode at the MCHK entry point but has no other effect.

4.14.7 Forcing 21164 Reset

Assertion of **cfail_h** in a sysclk cycle in which **cack_h** is deasserted causes the 21164 to execute a partial internal reset and then trap to the MCHK entry point in PALcode. The current command, if any, and all pending fills, and all pending system commands are cleared. The 21164 will complete its partial reset in 128 CPU cycles, then begin execution of the machine check PALcode flow. The system should not send a request to the 21164 during this time.

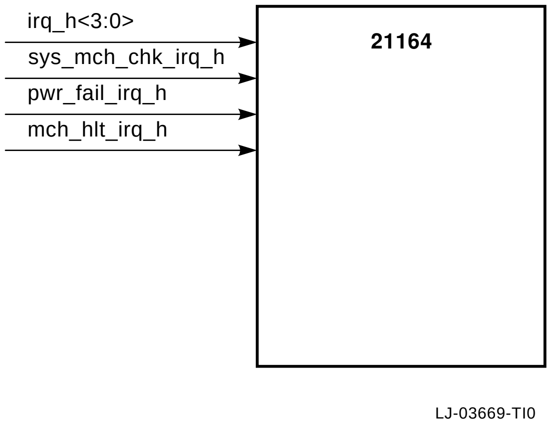
This mechanism is used by the 21164 to restore itself and the system to a consistent state after command or address parity error or a timeout error. Refer also to Section 8.1.18.

4.15 Interrupts

The 21164 has seven interrupt signals that have different uses during initialization and normal operation.

Figure 4–44 shows the 21164 interrupt signals.

Figure 4–44 Alpha 21164 Interrupt Signals



4.15.1 Interrupt Signals During Initialization

The 21164 interrupt signals work in tandem with the **sys_reset_l** signal to set the values for clock ratios and clock delays. During initialization, the 21164 reads system clock configuration parameters from the interrupt pins. Section 4.2.2 and Section 4.2.3 describe how the interrupt signals are used to set system clock values when the system is initialized.

4.15.2 Interrupt Signals During Normal Operation

During normal operation, interrupt signals indicate interrupt requests from external devices such as the realtime clock and I/O controllers.

4.15.3 Interrupt Priority Level

Table 4–19 shows which interrupts are enabled for a given interrupt priority level (IPL). An interrupt is enabled if the current IPL is less than the target IPL of the interrupt.

Table 4–19 Interrupt Priority Level Effect

Interrupt Source	Target IPL	Source
Software Interrupt Request 1	1	Internal
Software Interrupt Request 2	2	Internal
Software Interrupt Request 3	3	Internal
Software Interrupt Request 4	4	Internal
Software Interrupt Request 5	5	Internal
Software Interrupt Request 6	6	Internal
Software Interrupt Request 7	7	Internal
Software Interrupt Request 8	8	Internal
Software Interrupt Request 9	9	Internal
Software Interrupt Request 10	10	Internal
Software Interrupt Request 11	11	Internal
Software Interrupt Request 12	12	Internal
Software Interrupt Request 13	13	Internal
Software Interrupt Request 14	14	Internal
Software Interrupt Request 15	15	Internal
Asynchronous system trap ATR pending (for current or more privileged mode)	2	Internal
Performance counter interrupt	29	Internal
Powerfail interrupt[1]	30	**pwr_fail_irq_h**
System machine check interrupt[1], internally detected correctable error interrupt pending	31	**sys_mch_chk_irq_h** and internal
External interrupt 20[1]	20[2]	**irq_h<0>**
External interrupt 21[1]	21[2]	**irq_h<1>**
External interrupt 22[1]	22[2]	**irq_h<2>**
External interrupt 23[1]	23[2]	**irq_h<3>**

[1]These interrupts are from external sources. In some cases, the system environment provides the logic-OR of multiple interrupt sources at the same IPL to a particular pin.

[2]The external interrupts 20–23 are separately maskable by setting the appropriate bits in the ICSR register.

(continued on next page)

4.15 Interrupts

Table 4–19 (Cont.) Interrupt Priority Level Effect

Interrupt Source	Target IPL	Source
Halt[1]	Masked only by executing in PALmode.	**mch_hlt_irq_h**
Serial line interrupt	Masked only by executing in PALmode.	Internal

[1]These interrupts are from external sources. In some cases, the system environment provides the logic-OR of multiple interrupt sources at the same IPL to a particular pin.

When the processor receives an interrupt request and that request is enabled, an interrupt is reported or delivered to the exception logic if the processor is not currently executing PALcode. Before vectoring to the interrupt service PAL dispatch address, the pipeline is completely drained to the point that instructions issued before entering the PALcode cannot trap (implied TRAPB).

The restart address is saved in the exception address (EXC_ADDR) IPR and the processor enters PALmode. The cause of the interrupt can be determined by examining the state of the INTID and ISR registers.

Hardware interrupt requests are level sensitive and therefore may be removed before an interrupt is serviced. PALcode must verify that the interrupt actually indicated in INTID is to be serviced at an IPL higher that the current IPL. If it is not, PALcode should ignore the spurious interrupt.

5 Internal Processor Registers

This chapter describes the 21164 microprocessor internal processor registers (IPRs). It is organized as follows:

- Instruction fetch/decode unit and branch unit (Ibox) IPRs
- Memory address translation unit (Mbox) IPRs
- Cache control and bus interface unit (Cbox) IPRs
- PAL storage registers
- Restrictions

Ibox, Mbox, data cache (Dcache), and PALtemp IPRs are accessible to PALcode by means of the HW_MTPR and HW_MFPR instructions. Table 5–1 lists the IPR numbers for these instructions.

Cbox, second-level cache (Scache), and backup cache (Bcache) IPRs are accessible in the physical address region FF FFF0 0000 to FF FFFF FFFF. Table 5–25 summarizes the Cbox, Scache, and Bcache IPRs. Table 5–38 lists restrictions on the IPRs.

Note

Unless explicitly stated, IPRs are not cleared or set by hardware on chip or timeout reset.

Table 5–1 Ibox, Mbox, Dcache, and PALtemp IPR Encodings

IPR Mnemonic	Access	Index$_{16}$	Ibox Slots to Pipe
Ibox IPRs			
ISR	R	100	E1
ITB_TAG	W	101	E1
ITB_PTE	R/W	102	E1
ITB_ASN	R/W	103	E1
ITB_PTE_TEMP	R	104	E1
ITB_IA	W	105	E1
ITB_IAP	W	106	E1
ITB_IS	W	107	E1
SIRR	R/W	108	E1
ASTRR	R/W	109	E1
ASTER	R/W	10A	E1
EXC_ADDR	R/W	10B	E1
EXC_SUM	R/W0C	10C	E1
EXC_MASK	R	10D	E1
PAL_BASE	R/W	10E	E1
ICM	R/W	10F	E1
IPLR	R/W	110	E1
INTID	R	111	E1
IFAULT_VA_FORM	R	112	E1
IVPTBR	R/W	113	E1
HWINT_CLR	W	115	E1
SL_XMIT	W	116	E1
SL_RCV	R	117	E1
ICSR	R/W	118	E1
IC_FLUSH_CTL	W	119	E1
ICPERR_STAT	R/W1C	11A	E1
PMCTR	R/W	11C	E1

(continued on next page)

Table 5–1 (Cont.) Ibox, Mbox, Dcache, and PALtemp IPR Encodings

IPR Mnemonic	Access	$Index_{16}$	Ibox Slots to Pipe
PALtemp IPRs			
PALtemp0	R/W	140	E1
PALtemp1	R/W	141	E1
PALtemp2	R/W	142	E1
PALtemp3	R/W	143	E1
PALtemp4	R/W	144	E1
PALtemp5	R/W	145	E1
PALtemp6	R/W	146	E1
PALtemp7	R/W	147	E1
PALtemp8	R/W	148	E1
PALtemp9	R/W	149	E1
PALtemp10	R/W	14A	E1
PALtemp11	R/W	14B	E1
PALtemp12	R/W	14C	E1
PALtemp13	R/W	14D	E1
PALtemp14	R/W	14E	E1
PALtemp15	R/W	14F	E1
PALtemp16	R/W	150	E1
PALtemp17	R/W	151	E1
PALtemp18	R/W	152	E1
PALtemp19	R/W	153	E1
PALtemp20	R/W	154	E1
PALtemp21	R/W	155	E1
PALtemp22	R/W	156	E1
PALtemp23	R/W	157	E1
Mbox IPRs			
DTB_ASN	W	200	E0
DTB_CM	W	201	E0

(continued on next page)

Table 5–1 (Cont.) Ibox, Mbox, Dcache, and PALtemp IPR Encodings

IPR Mnemonic	Access	$Index_{16}$	Ibox Slots to Pipe
DTB_TAG	W	202	E0
DTB_PTE	R/W	203	E0
DTB_PTE_TEMP	R	204	E0
MM_STAT	R	205	E0
VA	R	206	E0
VA_FORM	R	207	E0
MVPTBR	W	208	E0
DTB_IAP	W	209	E0
DTB_IA	W	20A	E0
DTB_IS	W	20B	E0
ALT_MODE	W	20C	E0
CC	W	20D	E0
CC_CTL	W	20E	E0
MCSR	R/W	20F	E0
DC_FLUSH	W	210	E0
DC_PERR_STAT	R/W1C	212	E0
DC_TEST_CTL	R/W	213	E0
DC_TEST_TAG	R/W	214	E0
DC_TEST_TAG_TEMP	R/W	215	E0
DC_MODE	R/W	216	E0
MAF_MODE	R/W	217	E0

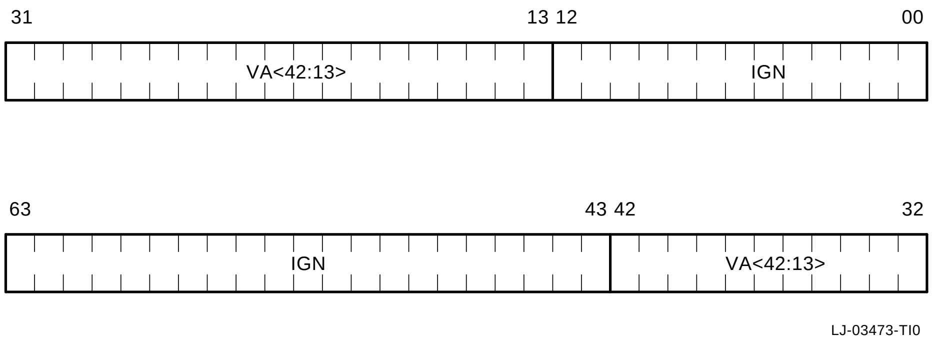
5.1 Instruction Fetch/Decode Unit and Branch Unit (Ibox) IPRs

The Ibox internal processor registers (IPRs) are described in Section 5.1.1 through Section 5.1.27.

5.1.1 Istream Translation Buffer Tag Register (ITB_TAG)

ITB_TAG is a write-only register written by hardware on an ITBMISS/IACCVIO, with the tag field of the faulting virtual address. To ensure the integrity of the instruction translation buffer (ITB), the TAG and page table entry (PTE) fields of an ITB entry are updated simultaneously by a write operation to the ITB_PTE register. This write operation causes the contents of the ITB_TAG register to be written into the tag field of the ITB location, which is determined by a not-last-used replacement algorithm. The PTE field is obtained from the HW_MTPR ITB_PTE instruction. Figure 5–1 shows the ITB_TAG register format.

Figure 5–1 Istream Translation Buffer Tag Register (ITB_TAG)



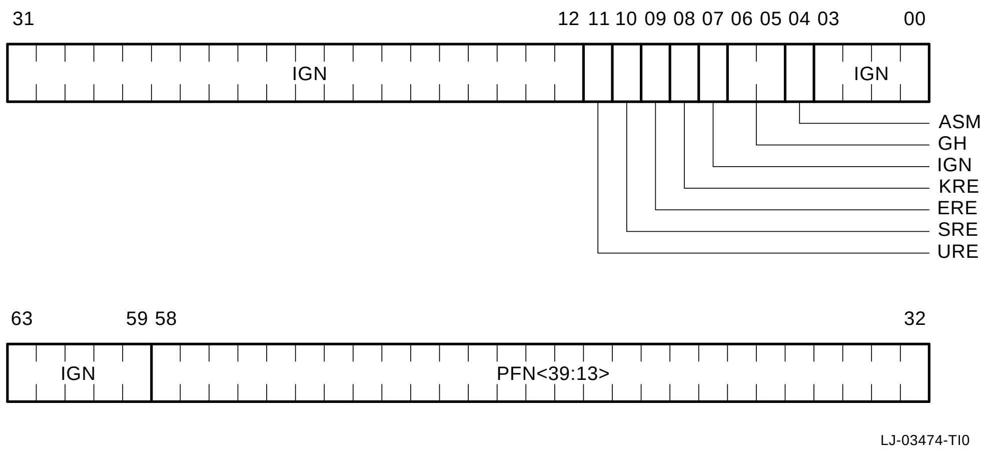
5.1.2 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register

ITB_PTE is a read/write register.

Write Format

A write operation to this register writes both the PTE and TAG fields of an ITB location determined by a not-last-used replacement algorithm. The TAG and PTE fields are updated simultaneously to ensure the integrity of the ITB. A write operation to the ITB_PTE register increments the not-last-used (NLU) pointer, which allows for writing the entire set of ITB PTE and TAG entries. If the HW_MTPR ITB_PTE instruction falls in the shadow of a trapping instruction, the NLU pointer may be incremented multiple times. The TAG field of the ITB location is determined by the contents of the ITB_TAG register. The PTE field is provided by the HW_MTPR ITB_PTE instruction. Write operations to this register use the memory format bits, as described in the *Alpha Architecture Reference Manual*. Figure 5–2 shows the ITB_PTE register write format.

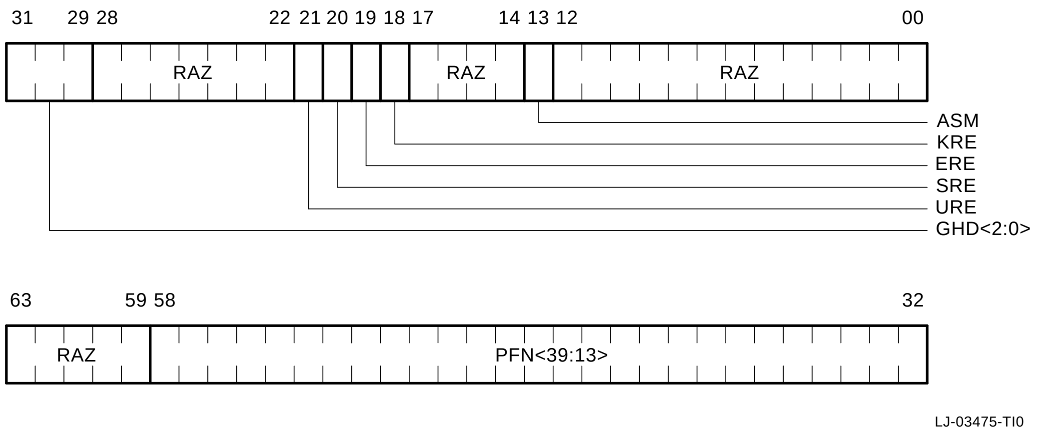
Figure 5–2 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Write Format



Read Format

A read of the ITB_PTE requires two instructions. A read of the ITB_PTE register returns the PTE pointed to by the NLU pointer to the ITB_PTE_TEMP register and increments the NLU pointer. If the HW_MFPR ITB_PTE instruction falls in the shadow of a trapping instruction, the NLU pointer may be incremented multiple times. A zero value is returned to the integer register file. A second read of the ITB_PTE_TEMP register returns the PTE to the general purpose integer register file (IRF). Figure 5–3 shows the ITB_PTE register read format.

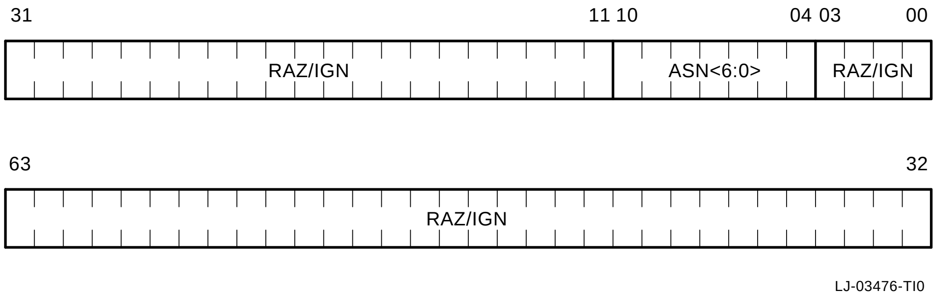
Figure 5–3 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Read Format



5.1.3 Instruction Translation Buffer Address Space Number (ITB_ASN) Register

ITB_ASN is a read/write register that contains the address space number (ASN) of the current process. Figure 5–4 shows the ITB_ASN register format.

Figure 5–4 Instruction Translation Buffer Address Space Number (ITB_ASN) Register



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

ITB_PTE_TEMP is a read-only holding register for ITB_PTE read data. A read of the ITB_PTE register returns data to this register. A second read of the ITB_PTE_TEMP register returns data to the general purpose integer register file (IRF). Figure 5–3 shows the ITB_PTE register format.

Table 5–2 shows the GHD settings for the ITB_PTE_TEMP register.

Table 5–2 Granularity Hint Bits in ITB_PTE_TEMP Read Format

Name	Extent	Type	Description
GHD	<29>	RO	Set if granularity hint equals 01, 10, or 11.
GHD	<30>	RO	Set if granularity hint equals 10 or 11.
GHD	<31>	RO	Set if granularity hint equals 11.

5.1.5 Instruction Translation Buffer Invalidate All Process (ITB_IAP) Register

ITB_IAP is a write-only register. Any write operation to this register invalidates all ITB entries that have an address space match (ASM) bit that equals zero.

5.1.6 Instruction Translation Buffer Invalidate All (ITB_IA) Register

ITB_IA is a write-only register. A write operation to this register invalidates all ITB entries, and resets the ITB not-last-used (NLU) pointer to its initial state. RESET PALcode must execute an HW_MTPR ITB_IA instruction in order to initialize the NLU pointer.

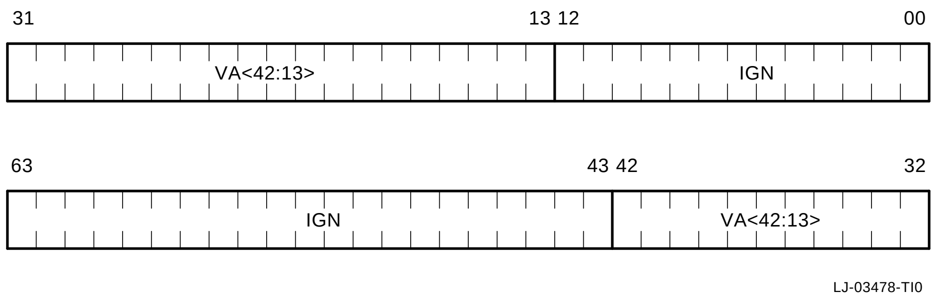
5.1.7 Instruction Translation Buffer IS (ITB_IS) Register

ITB_IS is a write-only register. Writing a virtual address to this register invalidates the ITB entry that meets either of the following criteria:

- An ITB entry whose virtual address (VA) field matches ITB_IS<42:13> and whose ASN field matches ITB_ASN<10:04>.
- An ITB entry whose VA field matches ITB_IS<42:13> and whose ASM bit is set.

Figure 5–5 shows the ITB_IS register format.

Figure 5–5 Instruction Translation Buffer IS (ITB_IS) Register



5.1.8 Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register

IFAULT_VA_FORM is a read-only register containing the formatted faulting virtual address on an ITBMISS/IACCVIO (except on IACCVIOs generated by sign-check errors). The formatted faulting address generated depends on whether NT superpage mapping is enabled through ICSR bit SPE<0>. Figure 5–6 shows the IFAULT_VA_FORM register format in non-NT mode.

Figure 5–6 Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register (NT_Mode=0)

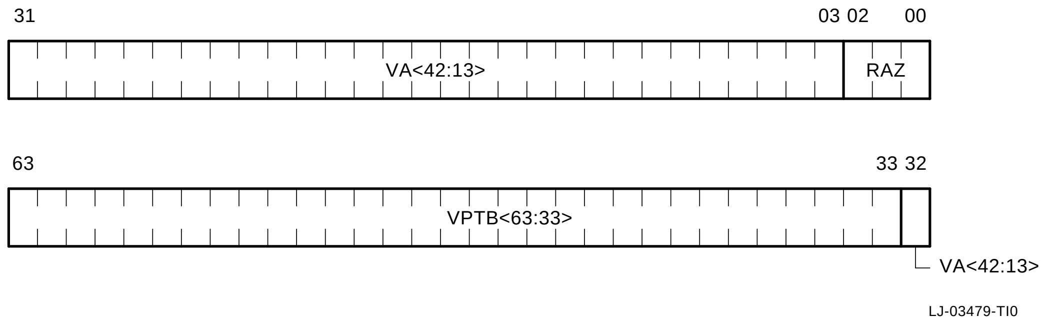
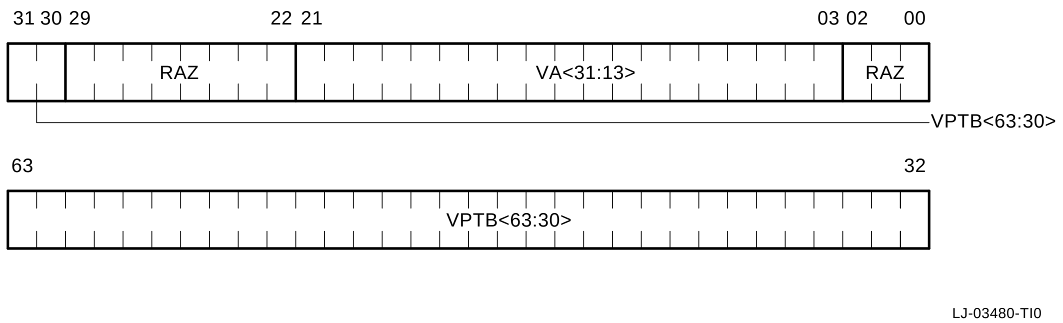


Figure 5–7 shows the IFAULT_VA_FORM register format in NT mode.

Figure 5–7 Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register (NT_Mode=1)



5.1.9 Virtual Page Table Base Register (IVPTBR)

IVPTBR is a read/write register. Bits <32:30> are UNDEFINED on a read of this register in non-NT mode. Figure 5–8 shows the IVPTBR format in non-NT mode.

Figure 5–8 Virtual Page Table Base Register (IVPTBR) (NT_Mode=0)

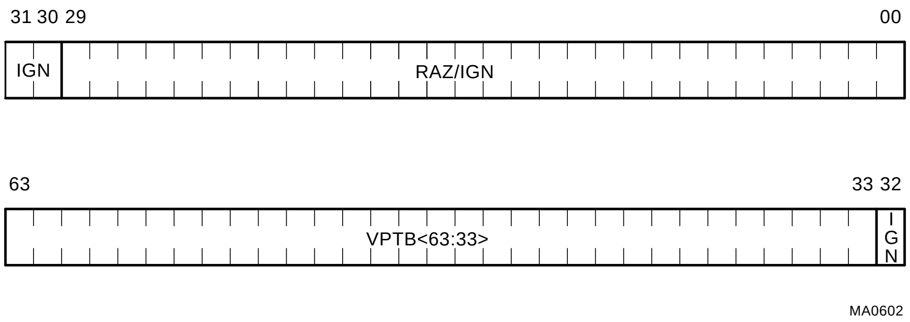
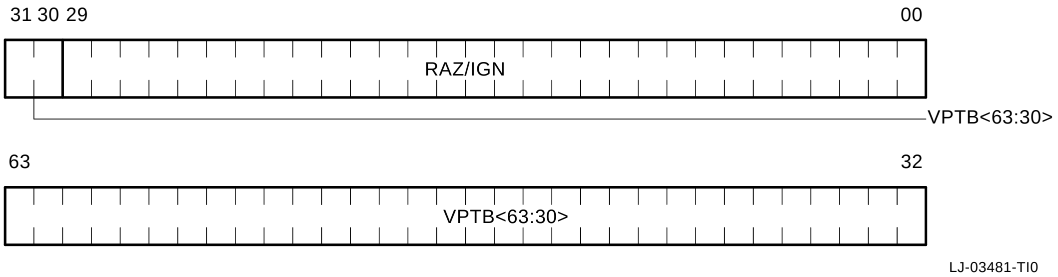


Figure 5–9 shows the IVPTBR format in NT mode.

Figure 5–9 Virtual Page Table Base Register (IVPTBR) (NT_Mode=1)



5.1.10 Icache Parity Error Status (ICPERR_STAT) Register

ICPERR_STAT is a read/write register. The Icache parity error status bits may be cleared by writing a 1 to the appropriate bits. Figure 5–10 and Table 5–3 describe the ICPERR_STAT register format.

Figure 5–10 Icache Parity Error Status (ICPERR_STAT) Register

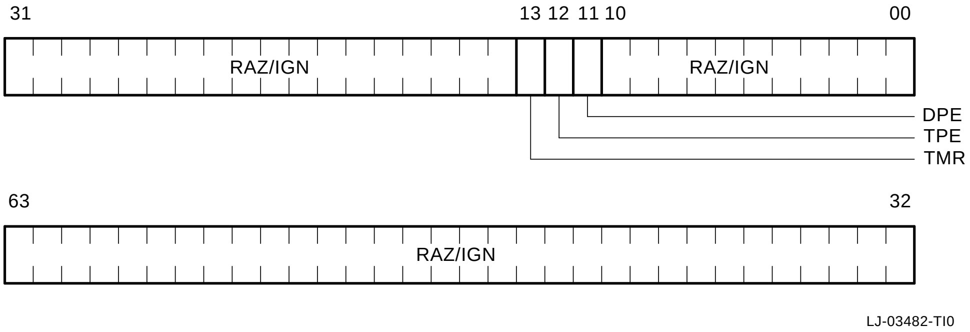


Table 5–3 Icache Parity Error Status Register Fields

Name	Extent	Type	Description
DPE	<11>	W1C	Data parity error
TPE	<12>	W1C	Tag parity error
TMR	<13>	W1C	Timeout reset error or **cfail_h**/no **cack_h** error

5.1.11 Icache Flush Control (IC_FLUSH_CTL) Register

IC_FLUSH_CTL is a write-only register. Writing any value to this register flushes the entire Icache.

5.1.12 Exception Address (EXC_ADDR) Register

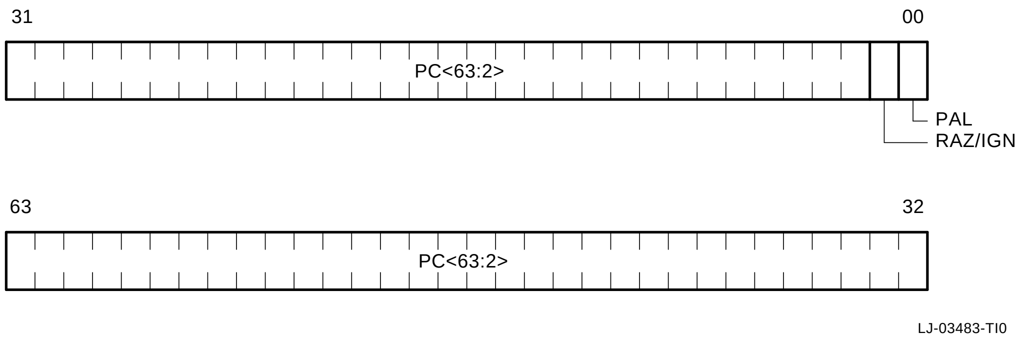
EXC_ADDR is a read/write register used to restart the system after exceptions or interrupts. The HW_REI instruction causes a return to the instruction pointed to by the EXC_ADDR register. This register can be written both by hardware and software. Hardware write operations occur as a result of exceptions/interrupts and CALL_PAL instructions. Hardware write operations that occur as a result of exceptions/interrupts take precedence over all other write operations.

In case of an exception/interrupt, hardware writes a program counter (PC) to this register. In case of precise exceptions, this is the PC value of the instruction that caused the exception. In case of imprecise exceptions/interrupts, this is the PC value of the next instruction that would have issued if the exception/interrupt was not reported.

In case of a CALL_PAL instruction, the PC value of the next instruction after the CALL_PAL is written to EXC_ADDR.

Bit <00> of this register is used to indicate PALmode. On a HW_REI instruction, the mode of the system is determined by bit <00> of EXC_ADDR. Figure 5–11 shows the EXC_ADDR register format.

Figure 5–11 Exception Address (EXC_ADDR) Register



5.1.13 Exception Summary (EXC_SUM) Register

EXC_SUM is a read/write register that records the different arithmetic traps that occur between EXC_SUM write operations. Any write operation to this register clears bits <16:10>. Figure 5–12 and Table 5–4 describe the EXC_SUM register format.

Figure 5–12 Exception Summary (EXC_SUM) Register

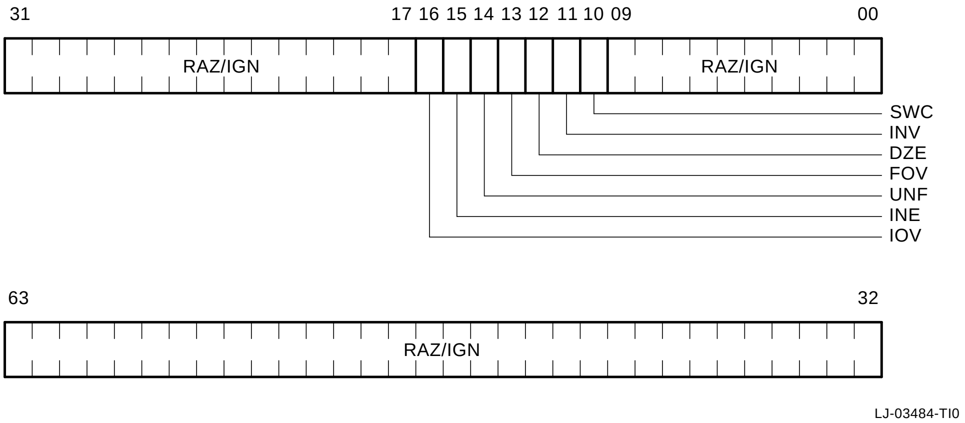


Table 5–4 Exception Summary Register Fields

Name	Extent	Type	Description
SWC	<10>	WA	Indicates software completion possible. This bit is set after a floating-point instruction containing the /S modifier completes with an arithmetic trap and if all previous floating-point instructions that trapped since the last HW_MTPR EXC_SUM instruction 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 an HW_MTPR instruction. The bit is always cleared upon any HW_MTPR write operation to the EXC_SUM register.

(continued on next page)

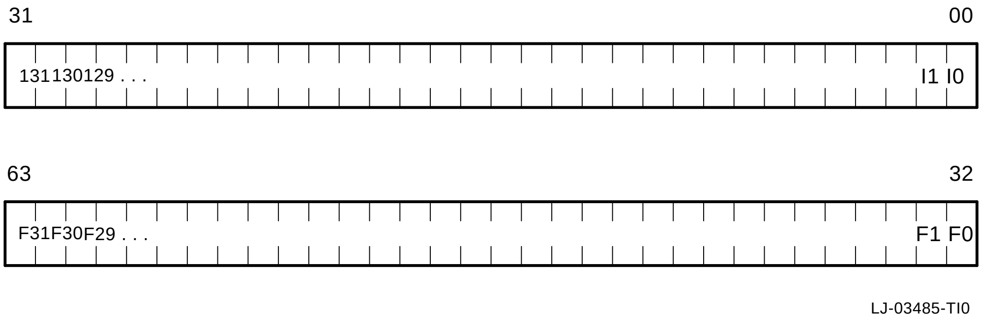
Table 5–4 (Cont.) Exception Summary Register Fields

Name	Extent	Type	Description
INV	<11>	WA	Indicates invalid operation.
DZE	<12>	WA	Indicates divide by zero.
FOV	<13>	WA	Indicates floating-point overflow.
UNF	<14>	WA	Indicates floating-point underflow.
INE	<15>	WA	Indicates floating inexact error.
IOV	<16>	WA	Indicates floating-point execution unit (Fbox) convert to integer overflow or integer arithmetic overflow.

5.1.14 Exception Mask (EXC_MASK) Register

EXC_MASK is a read/write register that records the destinations of instructions that have caused an arithmetic trap between EXC_MASK write operations. The destination is recorded as a single bit mask in the 64-bit IPR representing F0–F31 and I0–I31. A write operation to EXC_SUM clears the EXC_MASK register. Figure 5–13 shows the EXC_MASK register format.

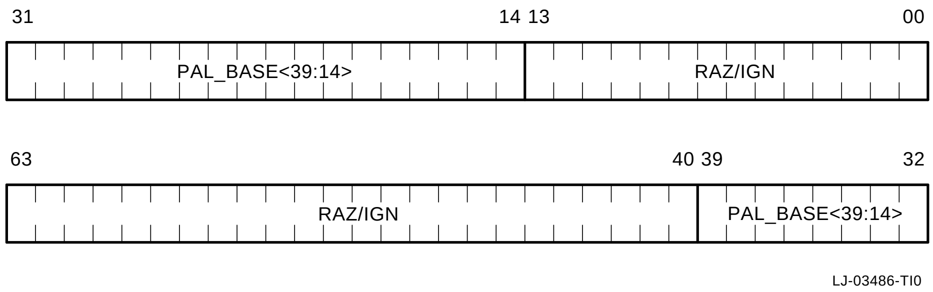
Figure 5–13 Exception Mask (EXC_MASK) Register



5.1.15 PAL Base Address (PAL_BASE) Register

PAL_BASE is a read/write register containing the base address for PALcode. The register is cleared by hardware on reset. Figure 5–14 shows the PAL_BASE register format.

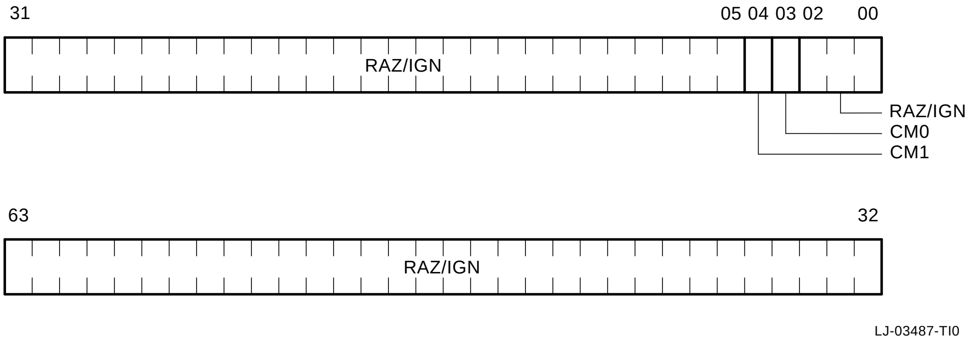
Figure 5–14 PAL Base Address (PAL_BASE) Register



5.1.16 Ibox Current Mode (ICM) Register

ICM is a read/write register containing the current mode bits of the architecturally defined processor status, as described in the *Alpha Architecture Reference Manual*. Figure 5–15 shows the ICM register format.

Figure 5–15 Ibox Current Mode (ICM) Register



5.1.17 Ibox Control and Status Register (ICSR)

ICSR is a read/write register containing Ibox-related control and status information. Figure 5–16 and Table 5–5 describe ICSR format.

Figure 5–16 Ibox Control and Status Register (ICSR)

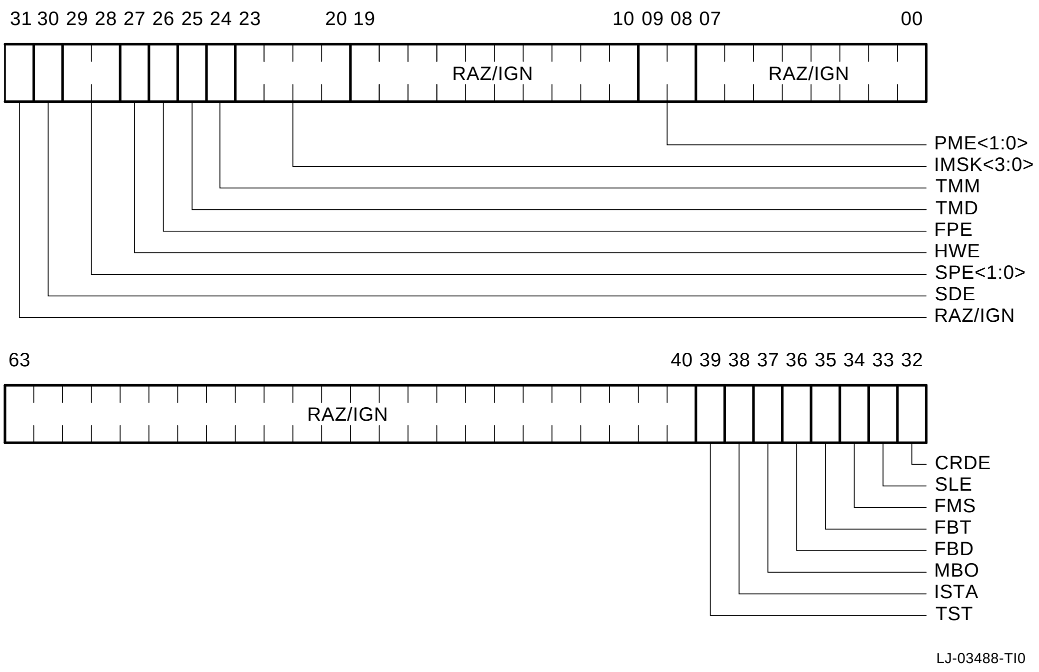


Table 5–5 Ibox Control and Status Register Fields

Name	Extent	Type	Description
PME<1:0>	<09:08>	RW,0	Performance counter master enable bits. If both PME<1> and PME<0> are clear, all performance counters in the PMCTR IPR are disabled. If either PME<1> or PME<0> are set, the counter is enabled according to the settings of the PMCTR CTL fields.
IMSK<3:0>	<23:20>	RW,0	If set, each IMSK<3:0> signal disables the corresponding IRQ_H<3:0> interrupt.
TMM	<24>	RW,0	If set, the timeout counter counts 5 thousand cycles before asserting timeout reset. If clear, the timeout counter counts 1 billion cycles before asserting timeout reset.
TMD	<25>	RW,0	If set, disables the Ibox timeout counter. Does not affect **cfail_h**/no **cack_h** error.
FPE	<26>	RW,0	If set, floating-point instructions may be issued. If clear, floating-point instructions cause FEN exceptions.
HWE	<27>	RW,0	If set, allows PALRES instructions to be issued in kernel mode.
SPE<1:0>	<29:28>	RW,0	If SPE<1> is set, it enables superpage mapping of Istream virtual address VA<39:13> directly to physical address PA<39:13> assuming VA<42:41> = 10. Virtual address bit VA<40> is ignored in this translation. Access is allowed only in kernel mode. If SPE<0> is set (NT mode), it enables superpage mapping of Istream virtual addresses VA<42:30> = $1FFE_{16}$ directly to physical address PA<39:30> = 0_{16}. VA<30:13> is mapped directly to PA<30:13>. Access is allowed only in kernel mode.
SDE	<30>	RW,0	If set, enables PAL shadow registers.
CRDE	<32>	RW,0	If set, enables correctable error interrupts.
SLE	<33>	RW,0	If set, enables serial line interrupts.
FMS	<34>	RW,0	If set, forces miss on Icache references. MBZ in normal operation.

(continued on next page)

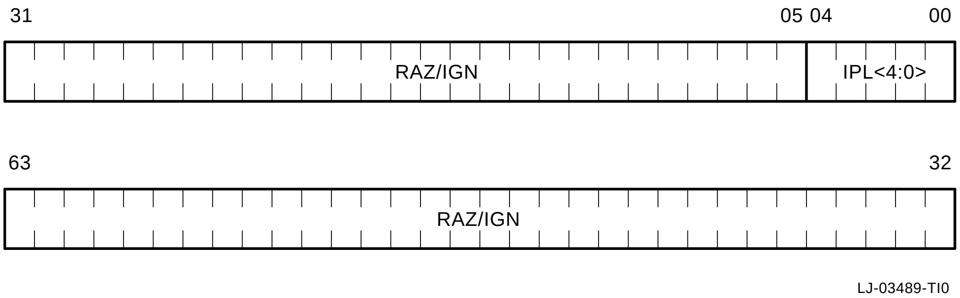
Table 5–5 (Cont.) Ibox Control and Status Register Fields

Name	Extent	Type	Description
FBT	<35>	RW,0	If set, forces bad Icache tag parity. MBZ in normal operation.
FBD	<36>	RW,0	If set, forces bad Icache data parity. MBZ in normal operation.
Reserved	<37>	RW,1	Reserved to Digital. Must be one.
ISTA	<38>	RO	Reading this bit indicates ICACHE BIST status. If set, ICACHE BIST was successful.
TST	<39>	RW,0	Writing a 1 to this bit asserts the **test_status_h<1>** signal.

5.1.18 Interrupt Priority Level Register (IPLR)

IPLR is a read/write register that is accessed by PALcode to set the value of the interrupt priority level (IPL). Whenever hardware detects an interrupt whose target IPL is greater than the value in IPLR<04:00>, an interrupt is taken. Figure 5–17 shows the IPLR register format. Refer to Table 4–19 for information on which interrupts are enabled for a given IPL.

Figure 5–17 Interrupt Priority Level Register (IPLR)



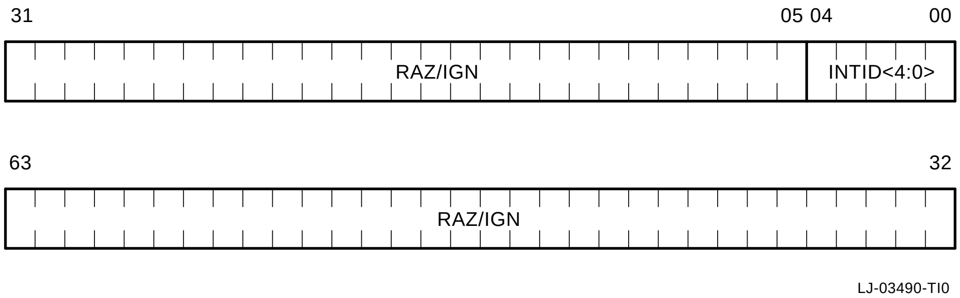
5.1.19 Interrupt ID (INTID) Register

INTID is a read-only register that is written by hardware with the target IPL of the highest priority pending interrupt. The hardware recognizes an interrupt if the IPL being read is greater than the IPL given by IPLR<04:00>.

Interrupt service routines may use the value of this register to determine the cause of the interrupt. PALcode, for the interrupt service, must ensure that the IPL in INTID is greater than the IPL specified by IPLR. This restriction is required because a level-sensitive hardware interrupt may disappear before the interrupt service routine is entered (passive release).

The contents of INTID are not correct on a HALT interrupt because this particular interrupt does not have a target IPL at which it can be masked. When a HALT interrupt occurs, INTID indicates the next highest priority pending interrupt. PALcode for interrupt service must check the interrupt summary register (ISR) to determine if a HALT interrupt has occurred. Figure 5–18 shows the INTID register format.

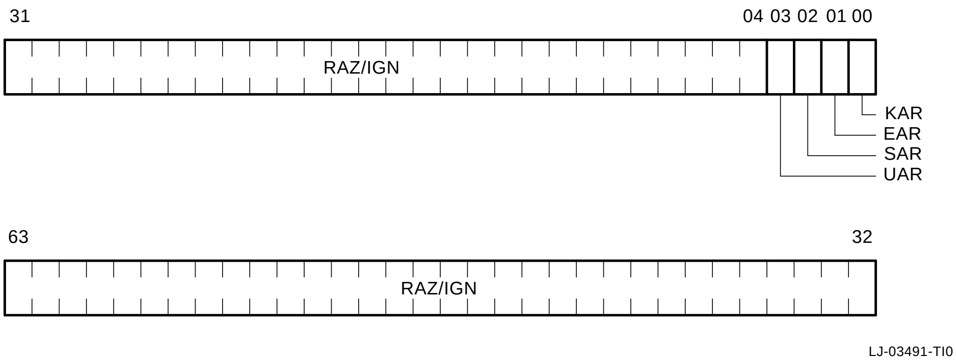
Figure 5–18 Interrupt ID (INTID) Register



5.1.20 Asynchronous System Trap Request Register (ASTRR)

ASTRR is a read/write register containing bits to request asynchronous system trap (AST) interrupts in each of the four processor modes (U,S,E,K). In order to generate an AST interrupt, the corresponding enable bit in the ASTER must be set and the current processor mode given in the ICM<04:03> should be equal to or higher than the mode associated with the AST request. Figure 5–19 shows the ASTRR format.

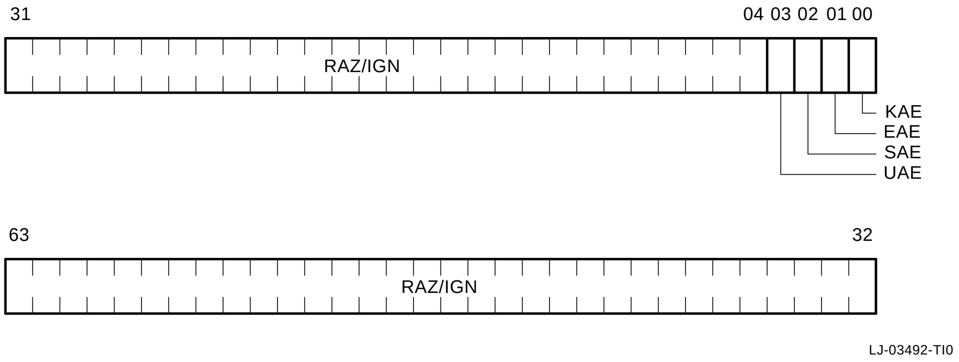
Figure 5–19 Asynchronous System Trap Request Register (ASTRR)



5.1.21 Asynchronous System Trap Enable Register (ASTER)

ASTER is a read/write register containing bits to enable corresponding asynchronous system trap (AST) interrupt requests. Figure 5–20 shows the ASTER format.

Figure 5–20 Asynchronous System Trap Enable Register (ASTER)



5.1.22 Software Interrupt Request Register (SIRR)

SIRR is a read/write register used to control software interrupt requests. A software request for a particular IPL may be requested by setting the appropriate bit in SIRR<15:01>. Figure 5–21 and Table 5–6 describe the SIRR format.

Figure 5–21 Software Interrupt Request Register (SIRR)

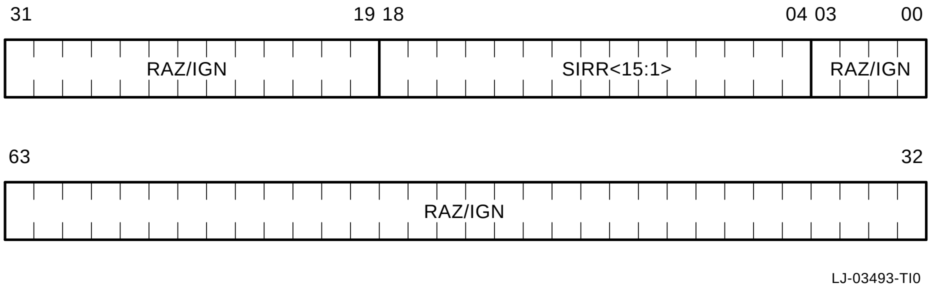


Table 5–6 Software Interrupt Request Register Fields

Name	Extent	Type	Description
SIRR<15:1>	<18:04>	RW	Request software interrupts.

5.1.23 Hardware Interrupt Clear (HWINT_CLR) Register

HWINT_CLR is a write-only register used to clear edge-sensitive hardware interrupt requests. Figure 5–22 and Table 5–7 describe the HWINT_CLR register format.

Figure 5–22 Hardware Interrupt Clear (HWINT_CLR) Register

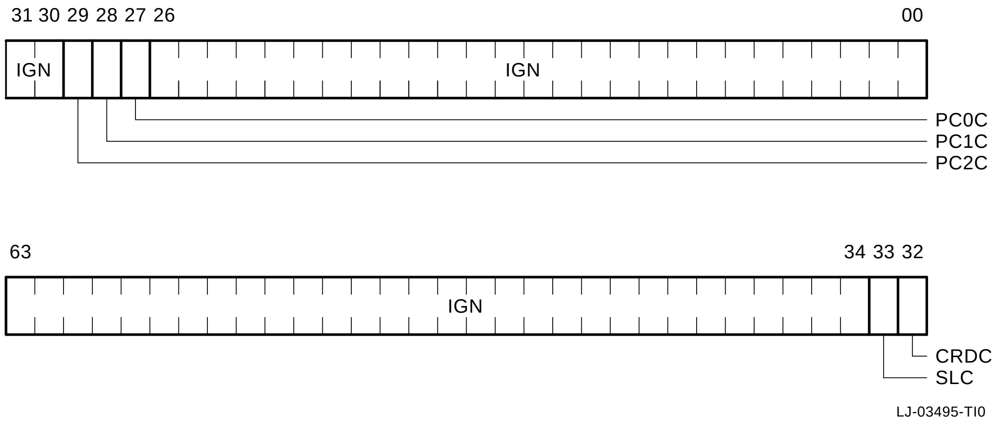


Table 5–7 Hardware Interrupt Clear Register Fields

Name	Extent	Type	Description
PC0C	<27>	W1C	Clears performance counter 0 interrupt requests.
PC1C	<28>	W1C	Clears performance counter 1 interrupt requests.
PC2C	<29>	W1C	Clears performance counter 2 interrupt requests.
CRDC	<32>	W1C	Clears correctable read data interrupt requests.
SLC	<33>	W1C	Clears serial line interrupt requests.

5.1.24 Interrupt Summary Register (ISR)

ISR is a read-only register containing information about all pending hardware, software, and asynchronous system trap (AST) interrupt requests. Figure 5–23 and Table 5–8 describe the ISR format. Refer to Table 4–19 for a description of which interrupts are enabled for a given interrupt priority level (IPL).

Figure 5–23 Interrupt Summary Register (ISR)

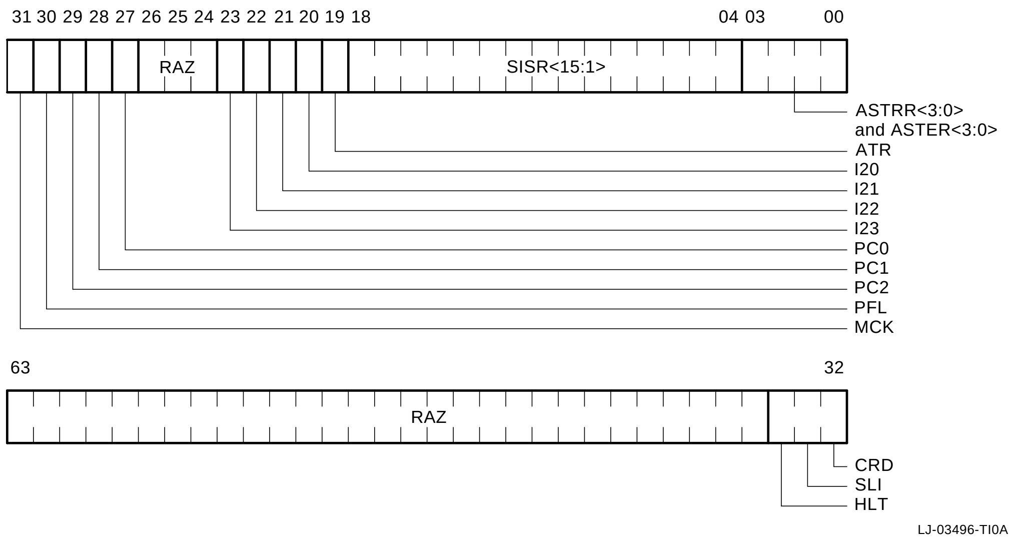


Table 5–8 Interrupt Summary Register Fields

Name	Extent	Type	Description
ASTRR<3:0> and ASTER<3:0>	<03:00>	RO	Boolean AND of ASTRR<USEK> with ASTER<USEK> used to indicate enabled AST requests.
SISR<15:1>	<18:04>	RO,0	Software interrupt requests 15 through 1 corresponding to IPL 15 through 1.
ATR	<19>	RO	Set if any AST request and corresponding enable bit is set and if the processor mode is equal to or higher than the AST request mode.
I20	<20>	RO	External hardware interrupt—**irq_h<0>**.
I21	<21>	RO	External hardware interrupt—**irq_h<1>**.
I22	<22>	RO	External hardware interrupt—**irq_h<2>**.
I23	<23>	RO	External hardware interrupt—**irq_h<3>**.
PC0	<27>	RO	External hardware interrupt—performance counter 0 (IPL 29).
PC1	<28>	RO	External hardware interrupt—performance counter 1 (IPL 29).
PC2	<29>	RO	External hardware interrupt—performance counter 2 (IPL 29).
PFL	<30>	RO	External hardware interrupt—power failure (IPL 30).
MCK	<31>	RO	External hardware interrupt—system machine check (IPL 31).
CRD	<32>	RO	Correctable ECC errors (IPL 31).
SLI	<33>	RO	Serial line interrupt.
HLT	<34>	RO	External hardware interrupt—halt.

5.1.25 Serial Line Transmit (SL_XMIT) Register

SL_XMIT is a write-only register used to transmit bit-serial data out of the microprocessor chip under the control of a software timing loop. The value of the TMT bit is transmitted offchip on the **srom_clk_h** signal. In normal operation mode (not in debugging mode), the **srom_clk_h** signal serves both the serial line transmission and the Icache serial ROM interface (see Section 7.5). Figure 5–24 and Table 5–9 describe the SL_XMIT register format.

Figure 5–24 Serial Line Transmit (SL_XMIT) Register

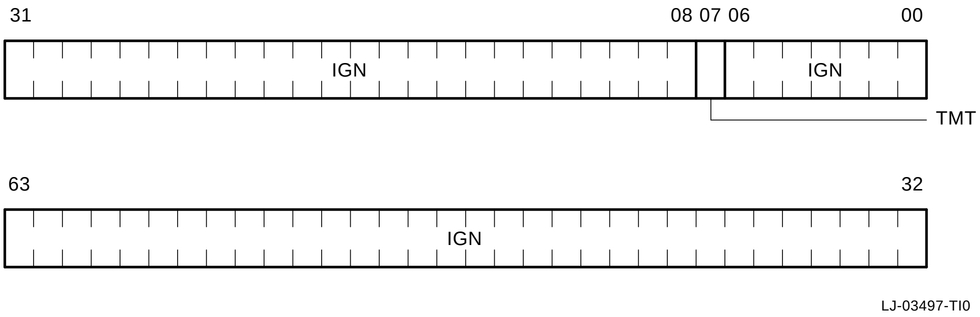


Table 5–9 Serial Line Transmit Register Fields

Name	Extent	Type	Description
TMT	<07>	WO,1	Serial line transmit data

5.1.26 Serial Line Receive (SL_RCV) Register

SL_RCV is a read-only register used to receive bit-serial data under the control of a software timing loop. The RCV bit in the SL_RCV register is functionally connected to the **srom_data_h** signal. A serial line interrupt is requested whenever a transition is detected on the **srom_data_h** signal and the SLE bit in the ICSR is set. During normal operations (not in test mode), the **srom_data_h** signal serves both the serial line reception and the Icache serial ROM (SROM) interface (see Section 7.5). Figure 5–25 and Table 5–10 describe the SL_RCV register format.

Figure 5–25 Serial Line Receive (SL_RCV) Register

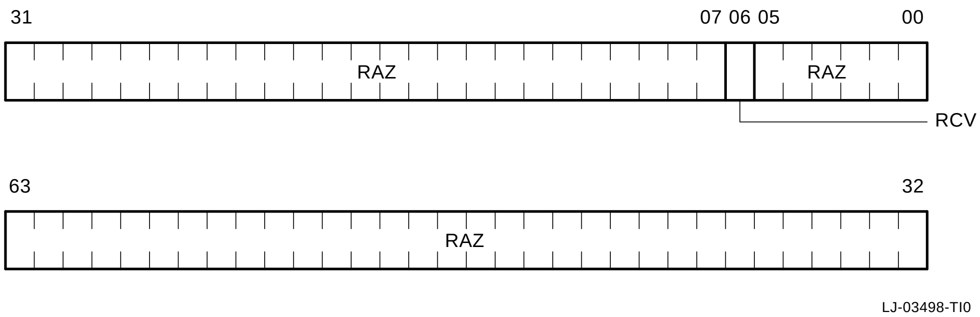


Table 5–10 Serial Line Receive Register Fields

Name	Extent	Type	Description
RCV	<06>	RO	Serial line receive data

5.1.27 Performance Counter (PMCTR) Register

PMCTR is a read/write register that controls the three onchip performance counters. Figure 5–26 and Table 5–11 describe the PMCTR format. Performance counter interrupt requests are summarized in Section 5.1.24. Cbox inputs to the counter select options are described in Table 5–31. Section 2.8 describes the performance measurement support features.

Note

The arrangement of the select option tables is not meant to imply any restrictions on permitted combinations of selections. The only cases in which the selection for one counter influences another's count is SEL1=8 (SEL 2=2, 3, other).

Figure 5–26 Performance Counter (PMCTR) Register

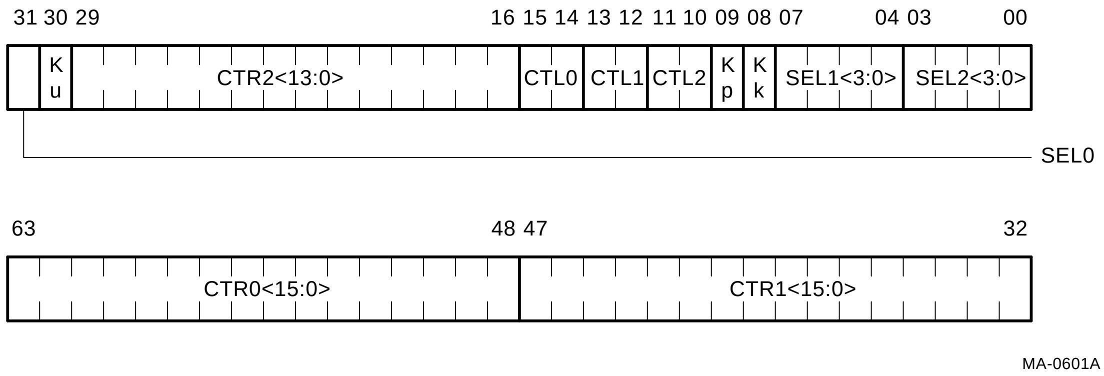


Table 5–11 Performance Counter Register Fields

Name	Extent	Type	Description
CTR0<15:0>	<63:48>	RW	A 16-bit counter of events selected by SEL0 and enabled by CTL0<1:0>.
CTR1<15:0>	<47:32>	RW	A 16-bit counter.
SEL0	<31>	RW	Counter0 Select—refer to Table 5–12.
Ku	<30>	RW	Kill user mode—disables all counters in user mode (refer to Table 5–13).
CTR2<13:0>	<29:16>	RW	14-bit counter
CTL0<1:0>	<15:14>	RW,0	CTR0 counter control: 00 counter disable, interrupt disable 01 counter enable, interrupt disable 10 counter enable, interrupt at count 65536 (Refer to Section 5.1.23 and Section 5.1.24.) 11 counter enable, interrupt at count 256
CTL1<1:0>	<13:12>	RW,0	CTR1 counter control: 00 counter disable,interrupt disable 01 counter enable, interrupt disable 10 counter enable, interrupt at count 65536 11 counter enable, interrupt at count 256
CTL2<1:0>	<11:10>	RW,0	CTR2 counter control: 00 counter disable,interrupt disable 01 counter enable, interrupt disable 10 counter enable, interrupt at count 16384 11 counter enable, interrupt at count 256
Kp	<09>	RW	Kill PALmode—disables all counters in PALmode (refer to Table 5–13).
Kk	<08>	RW	Kill kernel, executive, supervisor mode—disables all counters in kernel, executive, and supervisor modes (refer to Table 5–13). Ku=1, Kp=1, and Kk=1 enables counters in executive and supervisor modes only.
SEL1<3:0>	<07:04>	RW	Counter1 Select—refer to Table 5–12.
SEL2<3:0>	<03:00>	RW	Counter2 Select—refer to Table 5–12.

Table 5–12 shows the PMCTR counter select options.

Table 5–12 PMCTR Counter Select Options

Counter0 SEL0<0>	Counter1 SEL1<3:0>	Counter2 SEL2<3:0>
0:Cycles	0x0: nonissue cycles Valid instruction in S3 but none issued.	0x0: long(>15 cycle) stalls
	0x1: split-issue cycles Some, but not all, instructions at S3 issued.	0x1: reserved
	0x2: pipe-dry cycles No valid instruction at S3.	
	0x3: replay trap A replay trap occurred.	
	0x4: single-issue cycles Exactly one instruction issued.	
	0x5: dual-issue cycles Exactly two instructions issued.	
	0x6: triple-issue cycles Exactly three instructions issued.	
	0x7: quad-issue cycles Exactly four instructions issued.	
1:Instructions	0x8: jsr-ret if sel2=PC-M Instruction issued if sel2 is PC-M.	0x2: PC-mispredicts
	0x8: cond-branch if sel2=BR-M Instruction issued if sel2 is BR-M	0x3: BR-mispredicts
	0x8: all flow-change instructions if sel2=! (PC-M or BR-M)	
	0x9: IntOps issued	0x4: Icache/RFB misses
	0xA: FPOps issued	0x5: ITB misses
	0xB: loads issued	0x6: Dcache LD misses
	0xC: stores issued	0x7: DTB misses
	0xD: Icache issued	0x8: LDs merged in MAF

(continued on next page)

Table 5–12 (Cont.) PMCTR Counter Select Options

Counter0 SEL0<0>	Counter1 SEL1<3:0>	Counter2 SEL2<3:0>
	0xE: Dcache accesses	0x9: LDU replay traps
		0xA:WB/MAF full replay traps
		0xB: external **perf_mon_h** input. This counts in CPU cycles, but input is sampled in sysclk cycles. The external status **perf_mon_h** is sampled once per system clock and held through the system clock period. This means that ''sysclock ratio'' counts occur for each system clock cycle in which the status is true.
		0xC: CPU cycles
		0xD: MB stall cycles
		0xE: LD*x*L instructions issued
	0xF: pick CBOX input 1	0xF: pick CBOX input 2

Table 5–13 Measurement Mode Control

	Kill Bit Settings		
Measurement Mode Desired	**Ku**	**Kp**	**Kk**
Program	0	0	0
PAL only	1	0	1
OS only (kernel, executive, supervisor)	1	1	0
User only	0	1	1
All except PAL	0	1	0
OS + PAL (not user)	1	0	0
User + PAL (not kernel, executive, and supervisor)	0	0	1
Executive and supervisor only[1]	1	1	1

[1]In this instance, Kk means kill kernel only. The combination Ku=1, Kp=1, and Kk=1 is used to gather events for the executive and supervisor modes only.

Note

Both the user and the operating system can make PAL subroutine calls that put the machine in PALmode. The ‘‘OS only,’’ ‘‘user only,’’ and ‘‘executive and supervisor only’’ modes do not measure the events during the PAL subroutine calls made by the OS or user. The ‘‘OS + PAL’’ and ‘‘user + PAL’’ modes should be used carefully. ‘‘OS + PAL’’ mode measures the events during the PAL calls made by the user, whereas ‘‘user + PAL’’ mode measures the events during the PAL calls made by the OS.

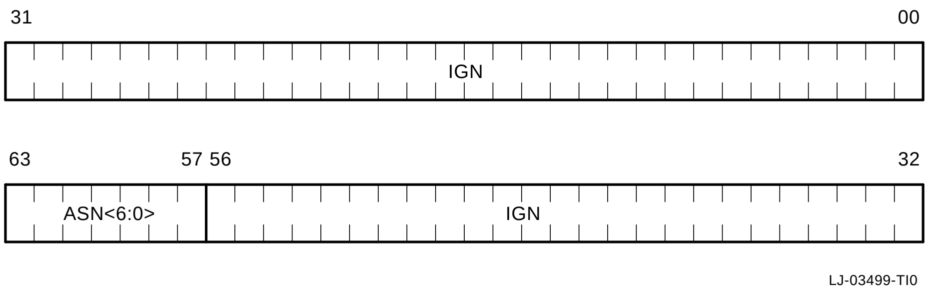
5.2 Memory Address Translation Unit (Mbox) IPRs

The Mbox internal processor registers (IPRs) are described in Section 5.2.1 through Section 5.2.23.

5.2.1 Dstream Translation Buffer Address Space Number (DTB_ASN) Register

DTB_ASN is a write-only register that must be written with an exact duplicate of the ITB_ASN register ASN field. Figure 5–27 shows the DTB_ASN register format.

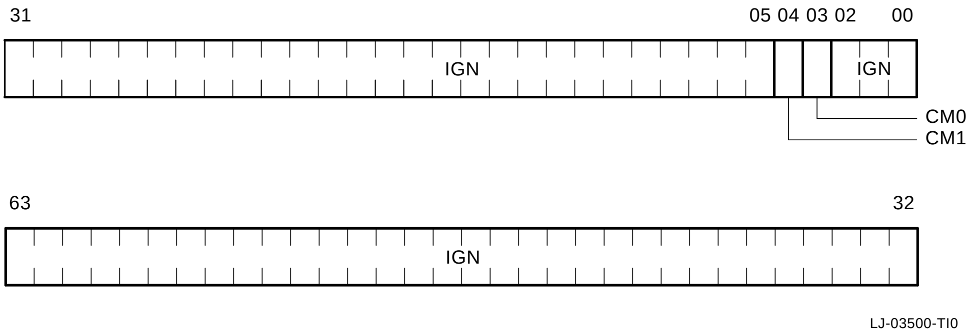
Figure 5–27 Dstream Translation Buffer Address Space Number (DTB_ASN) Register



5.2.2 Dstream Translation Buffer Current Mode (DTB_CM) Register

DTB_CM is a write-only register that must be written with an exact duplicate of the Ibox current mode (ICM) register CM field. These bits indicate the current mode of the machine, as described in the *Alpha Architecture Reference Manual*. Figure 5–28 shows the DTB_CM register format.

Figure 5–28 Dstream Translation Buffer Current Mode (DTB_CM) Register

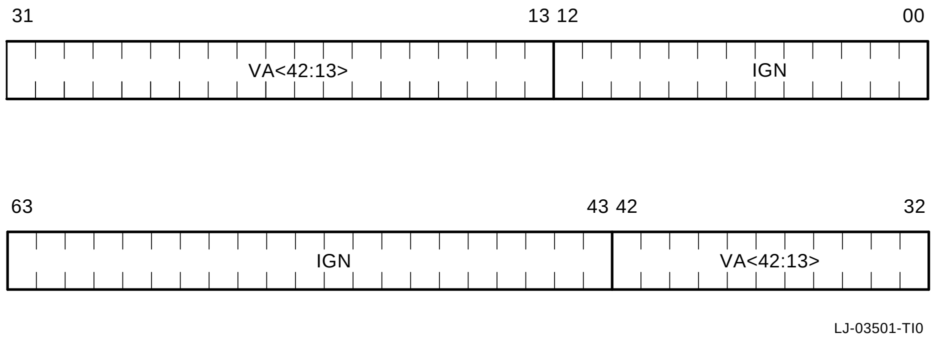


5.2.3 Dstream Translation Buffer Tag (DTB_TAG) Register

DTB_TAG is a write-only register that writes the DTB tag and the contents of the DTB_PTE register to the DTB. To ensure the integrity of the DTBs, the DTB's PTE array is updated simultaneously from the internal DTB_PTE register when the DTB_TAG register is written.

The entry to be written is chosen at the time of the DTB_TAG write operation by a not-last-used replacement algorithm implemented in hardware. A write operation to the DTB_TAG register increments the translation buffer (TB) entry pointer of the DTB, which allows writing the entire set of DTB PTE and TAG entries. The TB entry pointer is initialized to entry zero and the TB valid bits are cleared on chip reset but not on timeout reset. Figure 5–29 shows the DTB_TAG register format.

Figure 5–29 Dstream Translation Buffer Tag (DTB_TAG) Register



5.2.4 Dstream Translation Buffer Page Table Entry (DTB_PTE) Register

DTB_PTE is a read/write register representing the 64-entry DTB page table entries (PTEs). The entry to be written is chosen by a not-last-used replacement algorithm implemented in hardware. Write operations to DTB_PTE use the memory format bit positions, as described in the *Alpha Architecture Reference Manual*, with the exception that some fields are ignored. In particular, the page frame number (PFN) valid bit is not stored in the DTB.

To ensure the integrity of the DTB, the PTE is actually written to a temporary register and is not transferred to the DTB until the DTB_TAG register is written. As a result, writing the DTB_PTE and then reading without an intervening DTB_TAG write operation does not return the data previously written to the DTB_PTE register.

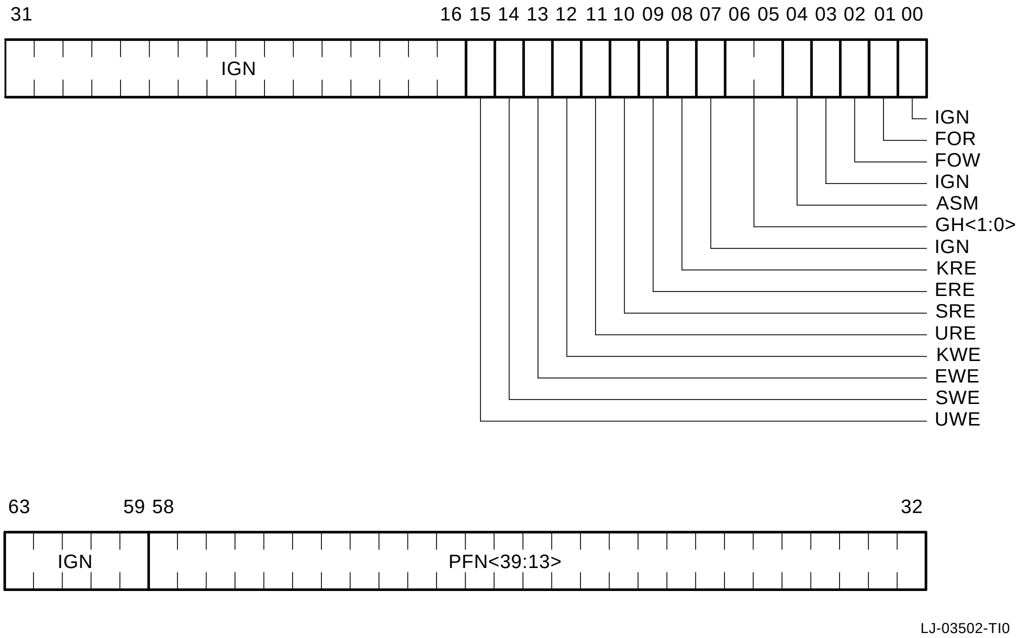
Read operations of the DTB_PTE require two instructions. First, a read from the DTB_PTE sends the PTE data to the DTB_PTE_TEMP register. A zero value is returned to the integer register file (IRF) on a DTB_PTE read operation. A second instruction reading from the DTB_PTE_TEMP register returns the PTE entry to the register file. Reading the DTB_PTE register increments the TB entry pointer of the DTB, which allows reading the entire set of DTB PTE entries. Figure 5–30 shows the DTB_PTE register format.

Note

The *Alpha Architecture Reference Manual* provides descriptions of the fields of the PTE.

5.2 Memory Address Translation Unit (Mbox) IPRs

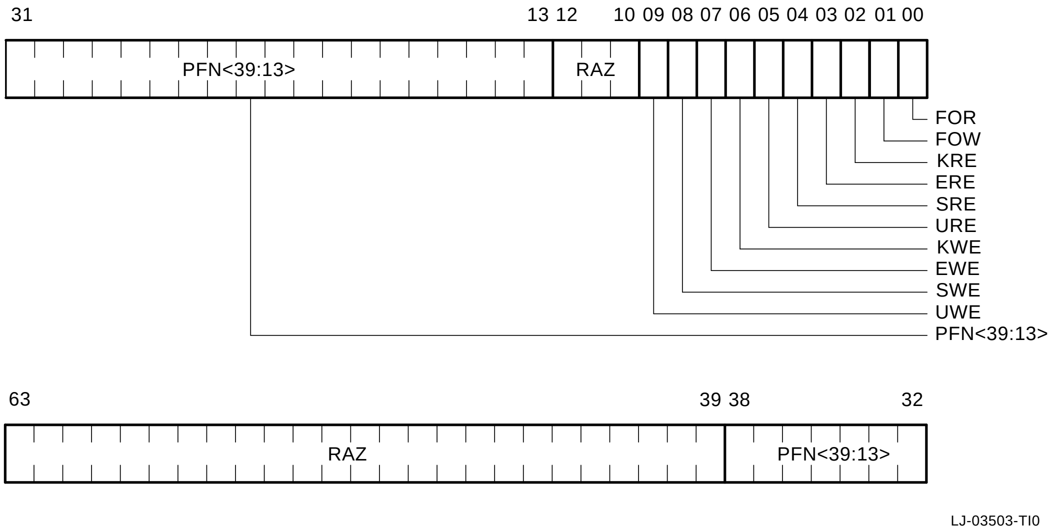
Figure 5–30 Dstream Translation Buffer Page Table Entry (DTB_PTE) Register—Write Format



5.2.5 Dstream Translation Buffer Page Table Entry Temporary (DTB_PTE_TEMP) Register

DTB_PTE_TEMP is a read-only holding register used for DTB_PTE data. Read operations of the DTB_PTE require two instructions to return the PTE data to the register file. The first reads the DTB_PTE register to the DTB_PTE_TEMP register and returns zero to the register file. The second returns the DTB_PTE_TEMP register to the integer register file (IRF). Figure 5–31 shows the DTB_PTE_TEMP register format.

Figure 5–31 Dstream Translation Buffer Page Table Entry Temporary (DTB_PTE_TEMP) Register



5.2.6 Dstream Memory Management Fault Status (MM_STAT) Register

MM_STAT is a read-only register that stores information on Dstream faults and Dcache parity errors. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. The MM_STAT bits are only modified by hardware when the register is not locked and a memory management error, DTB miss, or Dcache parity error occurs. The MM_STAT register is not unlocked or cleared on reset. Figure 5–32 and Table 5–14 describe the MM_STAT register format.

Figure 5–32 Dstream Memory Management Fault Status (MM_STAT) Register

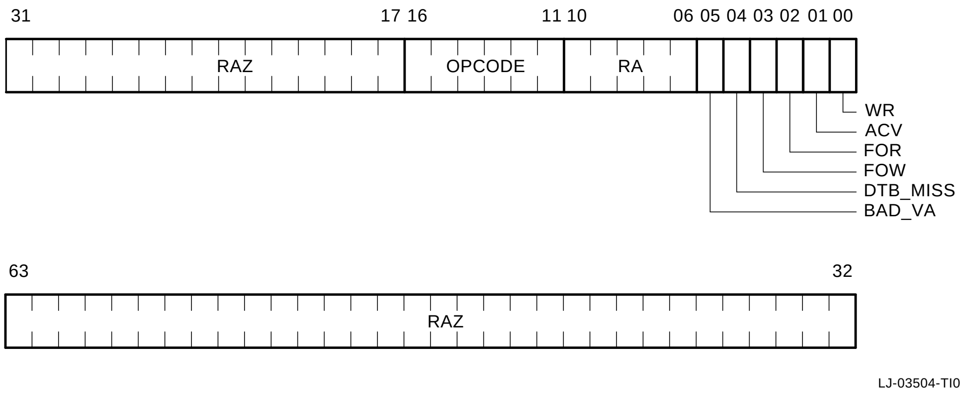


Table 5–14 Dstream Memory Management Fault Status Register Fields

Name	Extent	Type	Description
WR	<00>	RO	Set if reference that caused error was a write operation.
ACV	<01>	RO	Set if reference caused an access violation. Includes bad virtual address.
FOR	<02>	RO	Set if reference was a read operation and the PTE FOR bit was set.
FOW	<03>	RO	Set if reference was a write operation and the PTE FOW bit was set.
DTB_MISS	<04>	RO	Set if reference resulted in a DTB miss.
BAD_VA	<05>	RO	Set if reference had a bad virtual address.

(continued on next page)

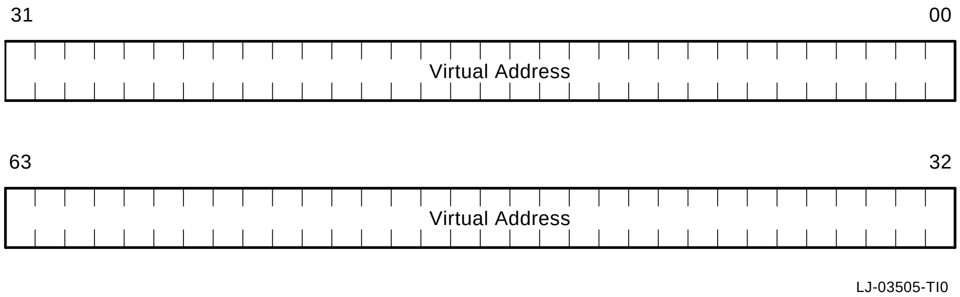
Table 5–14 (Cont.) Dstream Memory Management Fault Status Register Fields

Name	Extent	Type	Description
RA	<10:06>	RO	RA field of the faulting instruction.
OPCODE	<16:11>	RO	Opcode field of the faulting instruction.

5.2.7 Faulting Virtual Address (VA) Register

VA is a read-only register. When Dstream faults, DTB misses, or Dcache parity errors occur, the effective virtual address associated with the fault, miss, or error is latched in the VA register. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. The VA register is not unlocked on reset. Figure 5–33 shows the VA register format.

Figure 5–33 Faulting Virtual Address (VA) Register



5.2.8 Formatted Virtual Address (VA_FORM) Register

VA_FORM is a read-only register containing the virtual page table entry (PTE) address calculated as a function of the faulting virtual address and the virtual page table base (VA and MVPTBR registers). This is done as a performance enhancement to the Dstream TBmiss PAL flow.

The virtual address is formatted as a 32-bit PTE when the NT_Mode bit (MCSR<01>) is set (see Figure 5–34). VA_FORM is locked on any Dstream fault, DTB miss, or Dcache parity error. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. The VA_FORM register is not unlocked on reset. Figure 5–35 shows the VA_FORM register format when MCSR<01> is clear.

Figure 5–34 Formatted Virtual Address (VA_FORM) Register (NT_Mode=1)

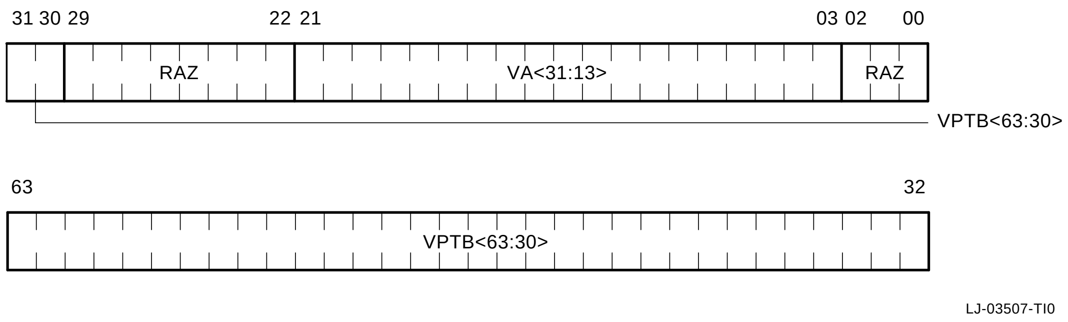
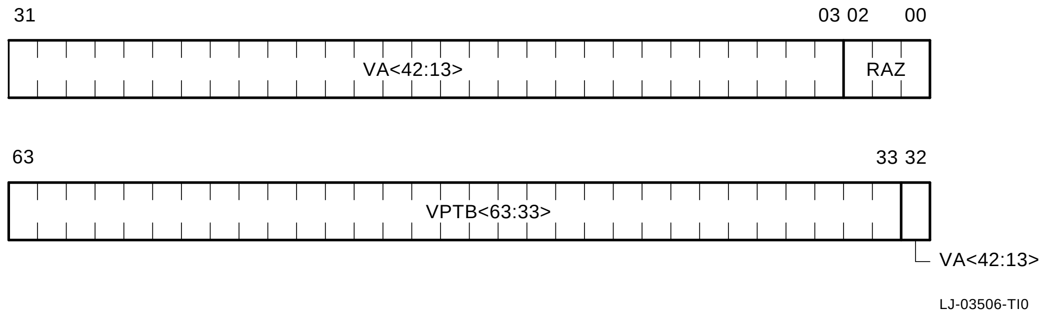


Figure 5–35 Formatted Virtual Address (VA_FORM) Register (NT_Mode=0)



5.2 Memory Address Translation Unit (Mbox) IPRs

Table 5–15 describes the VA_FORM register fields.

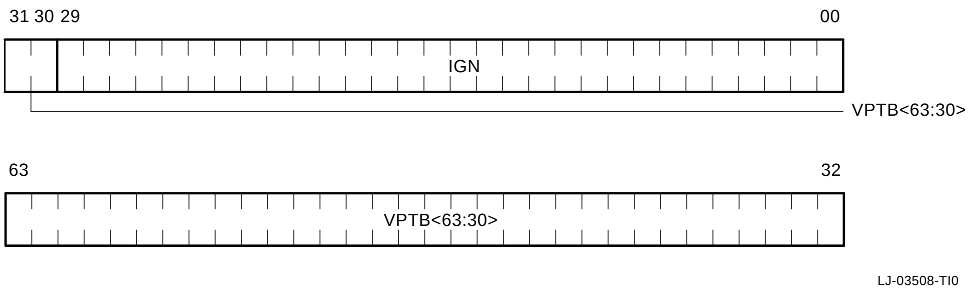
Table 5–15 Formatted Virtual Address Register Fields

Name	Extent	Type	Description
NT_Mode=0			
VPTB	<63:33>	RO	Virtual page table base address as stored in MVPTBR
VA<42:13>	<32:03>	RO	Subset of the original faulting virtual address
NT_Mode=1			
VPTB	<63:30>	RO	Virtual page table base address as stored in MVPTBR
VA<31:13>	<21:03>	RO	Subset of the original faulting virtual address

5.2.9 Mbox Virtual Page Table Base Register (MVPTBR)

MVPTBR is a write-only register containing the virtual address of the base of the page table structure. It is stored in the Mbox to be used in calculating the VA_FORM value for the Dstream TBmiss PAL flow. Unlike the VA register, the MVPTBR is not locked against further updates when a Dstream fault, DTB Miss, or Dcache parity error occurs. Figure 5–36 shows the MVPTBR format.

Figure 5–36 Mbox Virtual Page Table Base Register (MVPTBR)



5.2.10 Dcache Parity Error Status (DC_PERR_STAT) Register

DC_PERR_STAT is a read/write register that locks and stores Dcache parity error status. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. If a Dcache parity error is detected while the Dcache parity error status register is unlocked, the error status is loaded into DC_PERR_STAT<05:02>. The LOCK bit is set and the register is locked against further updates (except for the SEO bit) until software writes a 1 to clear the LOCK bit.

The SEO bit is set when a Dcache parity error occurs while the Dcache parity error status register is locked. Once the SEO bit is set, it is locked against further updates until the software writes a 1 to DC_PERR_STAT<00> to unlock and clear the bit. The SEO bit is not set when Dcache parity errors are detected on both pipes within the same cycle. In this particular situation, the pipe0/pipe1 Dcache parity error status bits indicate the existence of a second parity error. The DC_PERR_STAT register is not unlocked or cleared on reset.

Figure 5–37 and Table 5–16 describe the DC_PERR_STAT register format.

Figure 5–37 Dcache Parity Error Status (DC_PERR_STAT) Register

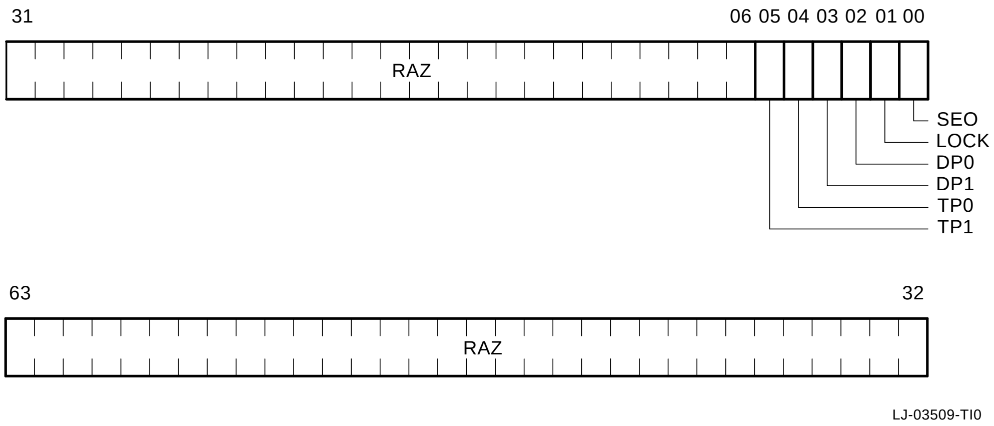


Table 5–16 Dcache Parity Error Status Register Fields

Name	Extent	Type	Description
SEO	<00>	W1C	Set if second Dcache parity error occurred in a cycle after the register was locked. The SEO bit is not set as a result of a second parity error that occurs within the same cycle as the first.
LOCK	<01>	W1C	Set if parity error detected in Dcache. Bits <05:02> are locked against further updates when this bit is set. Bits <05:02> are cleared when the LOCK bit is cleared.
DP0	<02>	RO	Set on data parity error in Dcache bank 0.
DP1	<03>	RO	Set on data parity error in Dcache bank 1.
TP0	<04>	RO	Set on tag parity error in Dcache bank 0.
TP1	<05>	RO	Set on tag parity error in Dcache bank 1.

5.2.11 Dstream Translation Buffer Invalidate All Process (DTB_IAP) Register

DTB_IAP is a write-only register. Any write operation to this register invalidates all data translation buffer (DTB) entries in which the address space match (ASM) bit is equal to zero.

5.2.12 Dstream Translation Buffer Invalidate All (DTB_IA) Register

DTB_IA is a write-only register. Any write operation to this register invalidates all 64 DTB entries, and resets the DTB not-last-used (NLU) pointer to its initial state.

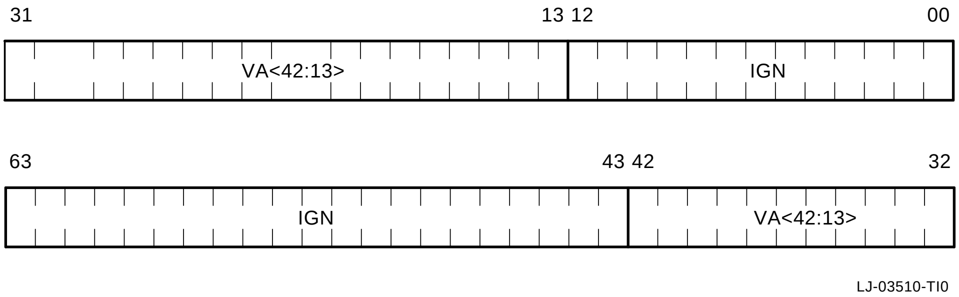
5.2.13 Dstream Translation Buffer Invalidate Single (DTB_IS) Register

DTB_IS is a write-only register. Writing a virtual address to this register invalidates the DTB entry that meets either of the following criteria:

- A DTB entry whose VA field matches DTB_IS<42:13> and whose ASN field matches DTB_ASN<63:57>.
- A DTB entry whose VA field matches DTB_IS<42:13> and whose ASM bit is set.

Figure 5–38 shows the DTB_IS register format.

Figure 5–38 Dstream Translation Buffer Invalidate Single (DTB_IS) Register



Note

The DTB_IS register is written before the normal Ibox trap point. The DTB invalidate single operation is aborted by the Ibox only for the following trap conditions:

- ITB miss
- PC mispredict
- When the HW_MTPR DTB_IS is executed in user mode

5.2.14 Mbox Control Register (MCSR)

MCSR is a read/write register that controls features and records status in the Mbox. This register is cleared on chip reset but not on timeout reset. Figure 5–39 and Table 5–17 describe the MCSR format.

Figure 5–39 Mbox Control Register (MCSR)

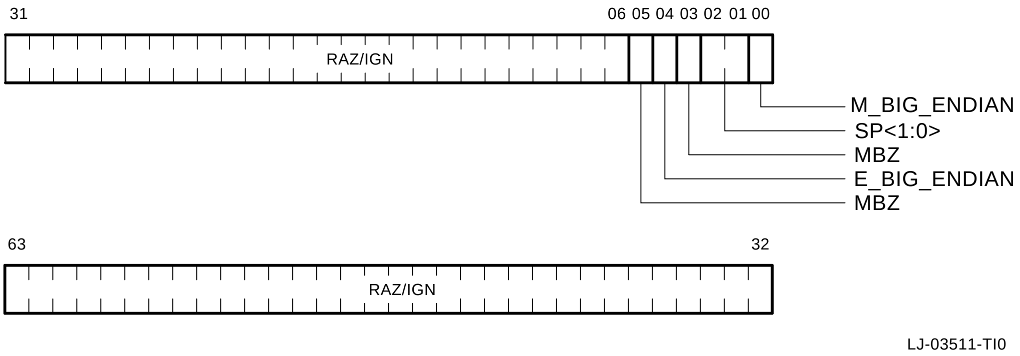


Table 5–17 Mbox Control Register Fields

Name	Extent	Type	Description
M_BIG_ ENDIAN	<00>	RW,0	Mbox Big Endian mode enable. When set, bit 2 of the physical address is inverted for all longword Dstream references.
SP<1:0>	<02:01>	RW,0	Superpage mode enables. **Note:** Superpage access is only allowed in kernel mode. SP<1> enables superpage mapping when VA<42:41> = 2. In this mode, virtual addresses VA<39:13> are mapped directly to physical addresses PA<39:13>. Virtual address bit VA<40> is ignored in this translation. SP<0> enables one-to-one superpage mapping of Dstream virtual addresses with VA<42:30> = $1FFE_{16}$. In this mode, virtual addresses VA<29:13> are mapped directly to physical addresses PA<29:13>, with bits <39:30> of physical address set to 0. SP<0> is the NT_Mode bit that is used to control virtual address formatting on a read operation from the VA_FORM register.
Reserved	<03>	RW,0	Reserved to Digital. Must be zero (MBZ).
E_BIG_ ENDIAN	<04>	RW,0	Ebox Big Endian mode enable. This bit is sent to the Ebox to enable Big Endian support for the EXT*xx*, MSK*xx* and INS*xx* byte instructions. This bit causes the shift amount to be inverted (one's-complemented) prior to the shifter operation.
Reserved	<05>	RW,0	Reserved to Digital. Must be zero (MBZ).

5.2.15 Dcache Mode (DC_MODE) Register

DC_MODE is a read/write register that controls diagnostic and test modes in the Dcache. This register is cleared on chip reset but not on timeout reset. Figure 5–40 and Table 5–18 describe the DC_MODE register format.

Note

The following bit settings are required for normal operation:

DC_ENA = 1
DC_FHIT = 0
DC_BAD_PARITY = 0
DC_PERR_DISABLE = 0

Figure 5–40 Dcache Mode (DC_MODE) Register

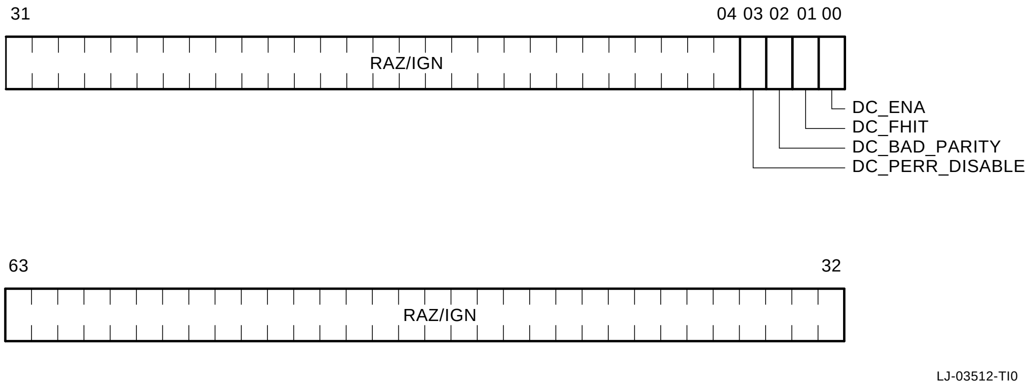


Table 5–18 Dcache Mode Register Fields

Name	Extent	Type	Description
DC_ENA	<00>	RW,0	Software Dcache enable. The DC_ENA bit enables the Dcache unless the Dcache has been disabled in hardware (DC_DOA is set). (The Dcache is enabled if DC_ENA=1 and DC_DOA=0). When clear, the Dcache command is not updated by ST or FILL operations, and all LD operations are forced to miss in the Dcache. Must be one (MBO) in normal operation.
DC_FHIT	<01>	RW,0	Dcache force hit. When set, the DC_FHIT bit forces all Dstream references to hit in the Dcache. Must be zero in normal operation.
DC_BAD_ PARITY	<02>	RW,0	When set, the DC_BAD_PARITY bit inverts the data parity inputs to the Dcache on integer stores. This has the effect of putting bad data parity into the Dcache on integer stores that hit in the Dcache. This bit has no effect on the tag parity written to the Dcache during FILL operations, or the data parity written to the Cbox write data buffer on integer store instructions. Floating-point store instructions should *not* be issued when this bit is set because it may result in bad parity being written to the Cbox write data buffer. Must be zero (MBZ) in normal operation.
DC_PERR_ DISABLE	<03>	RW,0	When set, the DC_PERR_DISABLE bit disables Dcache parity error reporting. When clear, this bit enables all Dcache tag and data parity errors. Parity error reporting is enabled during all other Dcache test modes unless this bit is explicitly set. Must be zero (MBZ) in normal operation.

5.2.16 Miss Address File Mode (MAF_MODE) Register

MAF_MODE is a read/write register that controls diagnostic and test modes in the Mbox miss address file (MAF). This register is cleared on chip reset. MAF_MODE<05> is also cleared on timeout reset. Figure 5–41 and Table 5–19 describe the MAF_MODE register format.

Note

The following bit settings are required for normal operation:

DREAD_NOMERGE = 0
WB_FLUSH_ALWAYS = 0
WB_NOMERGE = 0
MAF_ARB_DISABLE = 0
WB_CNT_DISABLE = 0

Figure 5–41 Miss Address File Mode (MAF_MODE) Register

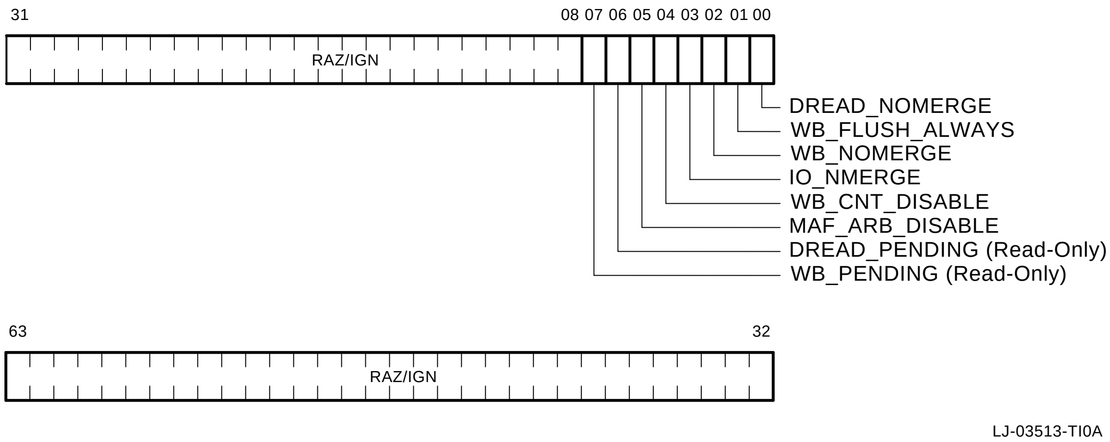


Table 5–19 Miss Address File Mode Register Fields

Name	Extent	Type	Description
DREAD_ NOMERGE	<00>	RW,0	Miss address file (MAF) DREAD Merge Disable. When set, this bit disables all merging in the DREAD portion of the MAF. Any load instruction that is issued when DREAD_NOMERGE is set is forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if DREAD_NOMERGE is cleared). Must be zero (MBZ) in normal operation.
WB_FLUSH_ ALWAYS	<01>	RW,0	When set, this bit forces the write buffer to flush whenever there is a valid WB entry. Must be zero (MBZ) in normal operation.
WB_ NOMERGE	<02>	RW,0	When set, this bit disables all merging in the write buffer. Any store instruction that is issued when WB_ NOMERGE is set is forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if WB_NOMERGE is cleared). Must be zero (MBZ) in normal operation.
IO_NMERGE	<03>	RW,0	When set, this bit prevents loads from I/O space (address bit <39>=1) from merging in the MAF. Should be zero (SBZ) in typical operation.
WB_CNT_ DISABLE	<04>	RW,0	When set, this bit disables the 64-cycle WB counter in the MAF arbiter. The top entry of the WB arbitrates at low priority only when a LD*x*_L instruction is issued or a second WB entry is made. Must be zero (MBZ) in normal operation.
MAF_ARB_ DISABLE	<05>	RW,0	When set, this bit disables all DREAD and WB requests in the MAF arbiter. WB_Reissue, Replay, Iref and MB requests are not blocked from arbitrating for the Scache. This bit is cleared on both timeout and chip reset. Must be zero (MBZ) in normal operation.
DREAD_ PENDING	<06>	R,0	Indicates the status of the MAF DREAD file. When set, there are one or more outstanding DREAD requests in the MAF file. When clear, there are no outstanding DREAD requests.
WB_ PENDING	<07>	R,0	This bit indicates the status of the MAF WB file. When set, there are one or more outstanding WB requests in the MAF file. When clear, there are no outstanding WB requests.

5.2.17 Dcache Flush (DC_FLUSH) Register

DC_FLUSH is a write-only register. A write operation to this register clears all the valid bits in both banks of the Dcache.

5.2.18 Alternate Mode (ALT_MODE) Register

ALT_MODE is a write-only register that specifies the alternate processor mode used by some HW_LD and HW_ST instructions. Figure 5–42 and Table 5–20 describe the ALT_MODE register format.

Figure 5–42 Alternate Mode (ALT_MODE) Register

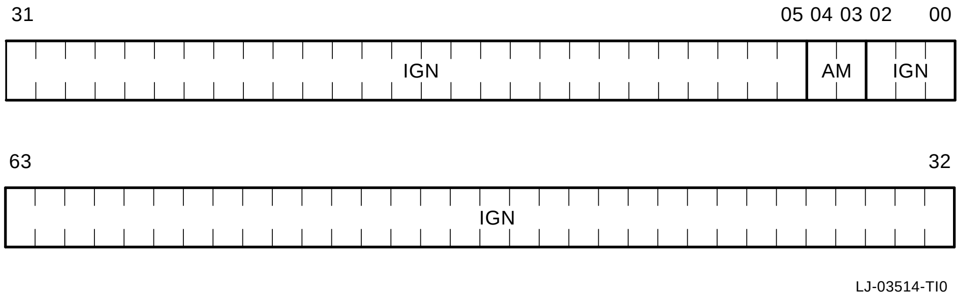


Table 5–20 Alternate Mode Register Settings

ALT_MODE<04:03>	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

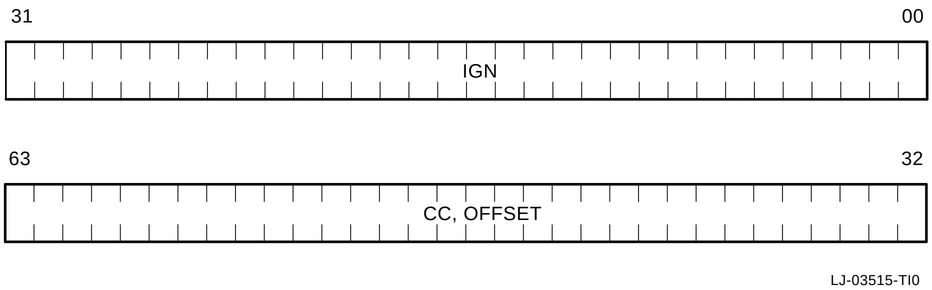
5.2.19 Cycle Counter (CC) Register

CC is a read/write register. The 21164 supports it as described in the *Alpha Architecture Reference Manual*. The low half of the counter, when enabled, increments once each CPU cycle. The upper half of the CC register is the counter offset. An HW_MTPR instruction writes CC<63:32>. Bits <31:00> are unchanged. CC_CTL<32> is used to enable or disable the cycle counter. The CC<31:00> is written to CC_CTL by an HW_MTPR instruction.

The CC register is read by the RPCC instruction as defined in the *Alpha Architecture Reference Manual*. The RPCC instruction returns a 64-bit value. The cycle counter is enabled to increment only three cycles after the MTPR CC_CTL (with CC_CTL<32> set) instruction is issued. This means that an RPCC instruction issued four cycles after an HW_MTPR CC_CTL instruction that enables the counter reads a value that is one greater than the initial count.

The CC register is disabled on chip reset. Figure 5–43 shows the CC register format.

Figure 5–43 Cycle Counter (CC) Register



5.2.20 Cycle Counter Control (CC_CTL) Register

CC_CTL is a write-only register that writes the low 32 bits of the cycle counter to enable or disable the counter. Bits CC<31:04> are written with the value in CC_CTL<31:04> on a HW_MTPR instruction to the CC_CTL register. Bits CC<03:00> are written with zero. Bits CC<63:32> are not changed. If CC_CTL<32> is set, then the counter is enabled; otherwise, the counter is disabled. Figure 5–44 and Table 5–21 describe the CC_CTL register format.

Figure 5–44 Cycle Counter Control (CC_CTL) Register

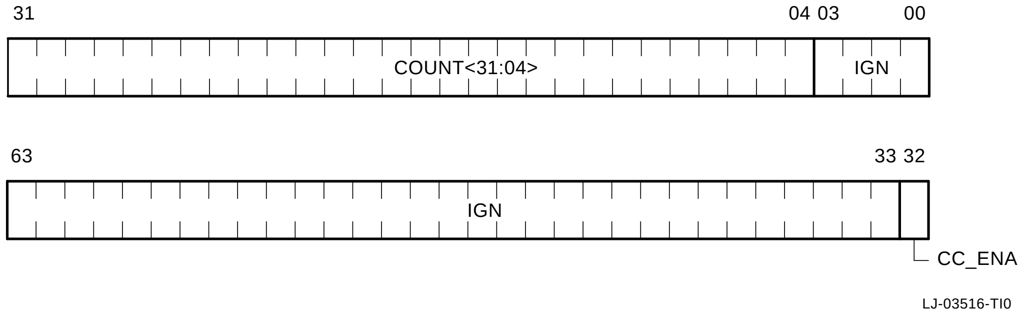


Table 5–21 Cycle Counter Control Register Fields

Name	Extent	Type	Description
COUNT<31:04>	<31:04>	WO	Cycle count. This value is loaded into CC<31:04>.
CC_ENA	<32>	WO	Cycle Counter enable. When set, this bit enables the CC register to begin incrementing 3 cycles later. An RPCC issued 4 cycles after CC_CTL<32> is written ‘‘sees’’ the initial count incremented by 1.

5.2.21 Dcache Test Tag Control (DC_TEST_CTL) Register

DC_TEST_CTL is a read/write register used exclusively for testing and diagnostics. An address written to this register is used to index into the Dcache array when reading or writing to the DC_TEST_TAG register. Figure 5–45 and Table 5–22 describe the DC_TEST_CTL register format. Section 5.2.22 describes how this register is used.

Figure 5–45 Dcache Test Tag Control (DC_TEST_CTL) Register

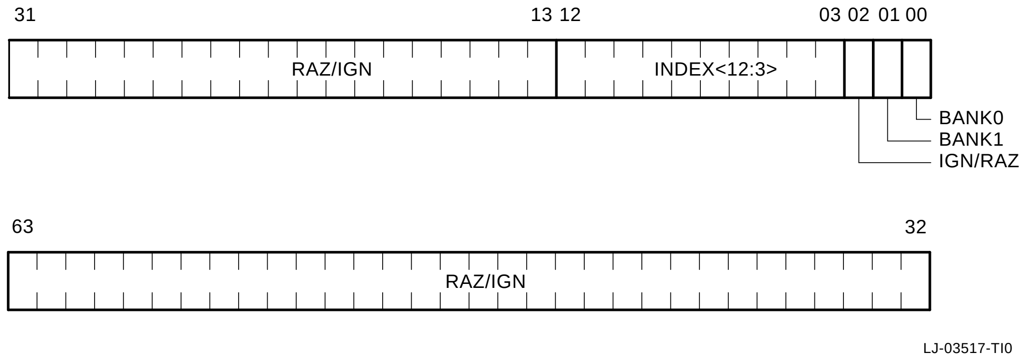


Table 5–22 Dcache Test Tag Control Register Fields

Name	Extent	Type	Description
BANK0	<00>	RW	Dcache Bank0 enable. When set, reads from DC_TEST_TAG return the tag from Dcache bank0, writes to DC_TEST_TAG write to Dcache bank0. When clear, reads from DC_TEST_TAG return the tag from Dcache bank1.
BANK1	<01>	RW	Dcache Bank1 enable. When set, writes to DC_TEST_TAG write to Dcache bank1. This bit has no effect on reads.
INDEX<12:3>	<12:03>	RW	Dcache tag index. This field is used on reads from and writes to the DC_TEST_TAG register to index into the Dcache tag array.

5.2.22 Dcache Test Tag (DC_TEST_TAG) Register

DC_TEST_TAG is a read/write register used exclusively for testing and diagnostics. When DC_TEST_TAG is read, the value in the DC_TEST_CTL register is used to index into the Dcache. The value in the tag, tag parity, valid and data parity bits for that index are read out of the Dcache and loaded into the DC_TEST_TAG_TEMP register. A zero value is returned to the integer register file (IRF). If BANK0 is set, the read operation is from Dcache bank0. Otherwise, the read operation is from Dcache bank1.

When DC_TEST_TAG is written, the value written to DC_TEST_TAG is written to the Dcache index referenced by the value in the DC_TEST_CTL register. The tag, tag parity, and valid bits are affected by this write operation. Data parity bits are not affected by this write operation (use DC_MODE<02> and force hit modes). If BANK0 is set, the write operation is to Dcache bank0. If BANK1 is set, the write operation is to Dcache bank1. If both are set, both banks are written.

Figure 5–46 and Table 5–23 describe the DC_TEST_TAG register format.

Figure 5–46 Dcache Test Tag (DC_TEST_TAG) Register

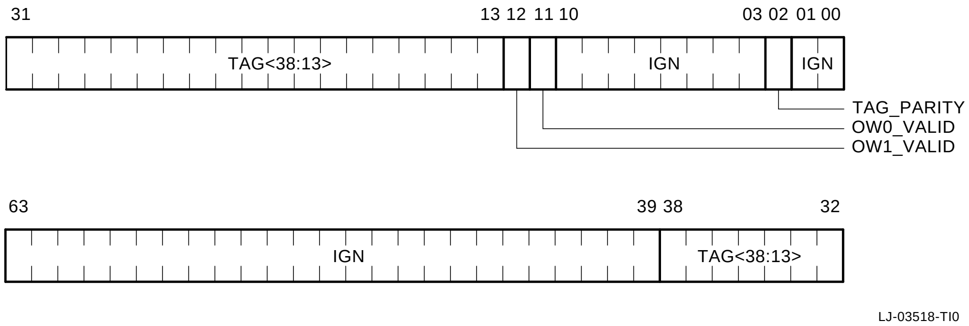


Table 5–23 Dcache Test Tag Register Fields

Name	Extent	Type	Description
TAG_PARITY	<02>	WO	Tag parity. This bit refers to the Dcache tag parity bit that covers tag bits 38 through 13 (valid bits not covered).
OW0_VALID	<11>	WO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low-order octaword within a Dcache 32-byte block.
OW1_VALID	<12>	WO	Octaword valid bit 1. This bit refers to the Dcache valid bit for the high-order octaword within a Dcache 32-byte block.
TAG<38:13>	<38:13>	WO	TAG<38:13>. These bits refer to the tag field in the Dcache array. **Note:** Bit 39 is not stored in the array.

5.2.23 Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register

DC_TEST_TAG_TEMP is a read-only register used exclusively for testing and diagnostics.

Reading the Dcache tag array requires a two-step read process:

1. The first read operation from DC_TEST_TAG reads the tag array and data parity bits and loads them into the DC_TEST_TAG_TEMP register. An UNDEFINED value is returned to the integer register file (IRF).
2. The second read operation of the DC_TEST_TAG_TEMP register returns the Dcache test data to the integer register file (IRF).

Figure 5–47 and Table 5–24 describe the DC_TEST_TAG_TEMP register format.

Figure 5–47 Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register

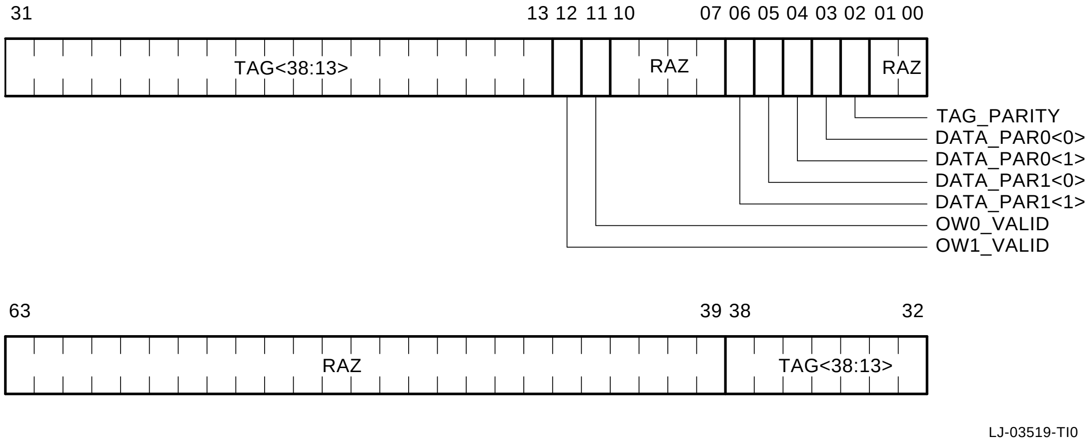


Table 5–24 Dcache Test Tag Temporary Register Fields

Name	Extent	Type	Description
TAG_PARITY	<02>	RO	Tag parity. This bit refers to the Dcache tag parity bit that covers tag bits 38 through 13 (valid bits not covered).
DATA_PAR0<0>	<03>	RO	Data parity. This bit refers to the Bank0 Dcache data parity bit that covers the lower longword of data indexed by DC_TEST_CTL<12:03>.
DATA_PAR0<1>	<04>	RO	Data parity. This bit refers to the Bank0 Dcache data parity bit that covers the upper longword of data indexed by DC_TEST_CTL<12:03>.
DATA_PAR1<0>	<05>	RO	Data parity. This bit refers to the Bank1 Dcache data parity bit that covers the lower longword of data indexed by DC_TEST_CTL<12:03>.
DATA_PAR1<1>	<06>	RO	Data parity. This bit refers to the Bank1 Dcache data parity bit that covers the upper longword of data indexed by DC_TEST_CTL<12:03>.
OW0_VALID	<11>	RO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low-order octaword within a Dcache 32-byte block.
OW1_VALID	<12>	RO	Octaword valid bit 1. This bit refers to the Dcache valid bit for the high-order octaword within a Dcache 32-byte block.
TAG<38:13>	<38:13>	RO	TAG<38:13>. These bits refer to the tag field in the Dcache array. **Note:** Bit 39 is not stored in the array.

5.3 External Interface Control (Cbox) IPRs

Table 5–25 lists specific IPRs for controlling Scache, Bcache, system configuration, and logging error information. These IPRs cannot be read or written from the system. They are placed in the 1 MB region of 21164-specific I/O address space ranging from FF FFF0 0000 to FF FFFF FFFF. Any read or write operation to an undefined IPR in this address space produces UNDEFINED behavior. The operating system should not map any address in this region as writable in any mode.

The Cbox internal processor registers are described in Section 5.3.1 through Section 5.3.9.

Table 5–25 Cbox Internal Processor Register Descriptions

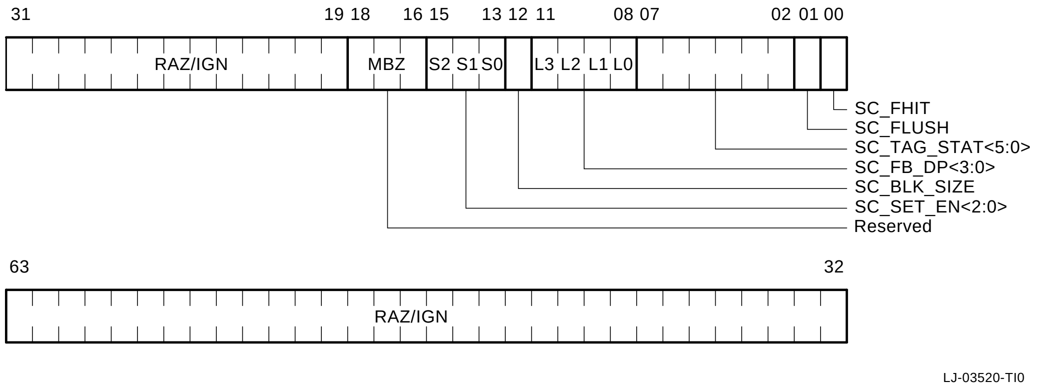
Register	Address	Type[1]	Description
SC_CTL	FF FFF0 00A8	RW	Controls Scache behavior.
SC_STAT	FF FFF0 00E8	R	Logs Scache-related errors.
SC_ADDR	FF FFF0 0188	R	Contains the address for Scache-related errors.
BC_CONTROL	FF FFF0 0128	W	Controls Bcache/system interface and Bcache testing.
BC_CONFIG	FF FFF0 01C8	W	Contains Bcache configuration parameters.
BC_TAG_ADDR	FF FFF0 0108	R	Contains tag and control bits for FILLs from Bcache.
EI_STAT	FF FFF0 0168	R	Logs Bcache/system-related errors.
EI_ADDR	FF FFF0 0148	R	Contains the address for Bcache/system-related errors.
FILL_SYN	FF FFF0 0068	R	Contains fill syndrome or parity bits for FILLs from Bcache or main memory.

[1]BC_CONTROL<01> must be 0 when reading any IPR in this table.

5.3.1 Scache Control (SC_CTL) Register (FF FFF0 00A8)

SC_CTL is a read/write register that controls Scache activity. Figure 5–48 and Table 5–26 describe the SC_CTL register format. The bits in this register are initialized to the value indicated in Table 5–26 on reset, but not on timeout reset.

Figure 5–48 Scache Control (SC_CTL) Register



5.3 External Interface Control (Cbox) IPRs

Table 5–26 Scache Control Register Fields

<table>
<tr><th>Field</th><th>Extent</th><th>Type</th><th>Description</th></tr>
<tr><td>SC_FHIT</td><td><00></td><td>RW,0</td><td>When set, this bit forces cacheable load and store instructions to hit in the Scache, irrespective of the tag status bits. Noncacheable references are not forced to hit in the Scache and will be driven offchip. In this mode, only one Scache set may be enabled. The Scache tag and data parity checking are disabled.

For store instructions, the value of the tag status and parity bits are specified by the SC_TAG_STAT<5:0> field. The tag is written with the address provided to the Scache with the store instruction.</td></tr>
<tr><td>SC_FLUSH</td><td><01></td><td>RW,0</td><td>All the Scache tag valid bits are cleared every time this bit field is written to 1.</td></tr>
<tr><td>SC_TAG_ STAT<5:0></td><td><07:02></td><td>RW,0</td><td>This field is used only in the SC_FHIT mode to write any combination of tag status and parity bits in the Scache. The parity bit can be used to write bad tag parity. The correct value of tag parity is even.

The following bits must be zero for normal operation:
<table>
<tr><th>Scache Tag Status<5:0></th><th>Description</th></tr>
<tr><td>SC_TAG_ STAT<5:2></td><td>Tag parity, valid, shared, dirty; bits 7, 6, 5, and 4 respectively</td></tr>
<tr><td>SC_TAG_ STAT<1:0></td><td>Octaword modified bits</td></tr>
</table></td></tr>
</table>

(continued on next page)

Table 5–26 (Cont.) Scache Control Register Fields

Field	Extent	Type	Description
SC_FB_DP<3:0>	<11:08>	RW,0	Force bad parity—This field is used to write bad data parity for the selected longwords within the octaword when writing the Scache. If any one of these bits is set to one, then the corresponding longword's computed parity value is inverted when writing the Scache. For Scache write transactions, the Cbox allocates two consecutive cycles to write up to two octawords based on the longword valid bits received from the Mbox. Therefore, the same longword parity control bits are used for writing both octawords. For example, SC_FB_DP<0> corresponds to LW0 and LW4. This bit field must be zero during normal operation.
SC_BLK_SIZE	<12>	RW,1	This bit selects the Scache and Bcache block size to be either 64 bytes or 32 bytes. The Scache and Bcache always have identical block sizes. All the Bcache and main memory FILLs or write transactions are of the selected block size. At power-up time, this bit is set and the default block size is 64 bytes. When clear, the block size is 32 bytes. This bit must be set to the desired value to reflect the correct Scache/Bcache block size before the 21164 does the first cacheable read or write transaction from Bcache or system.
SC_SET_EN<2:0>	<15:13>	RW,7	This field is used to enable the Scache sets. Only *one* or *all three* sets may be enabled at a time. Enabling any combination of *two* sets at a time results in UNPREDICTABLE behavior. One of the Scache sets must always be enabled irrespective of the Bcache.
Reserved	<18:16>	RW,0	Reserved to Digital. Must be zero (MBZ).

5.3.2 Scache Status (SC_STAT) Register (FF FFF0 00E8)

SC_STAT is a read-only register. It is not cleared or unlocked by reset. Any PALcode read of this register unlocks SC_ADDR and SC_STAT and clears SC_STAT.

If an Scache tag or data parity error is detected during an Scache lookup, the SC_STAT register is locked against further updates from subsequent transactions. Figure 5–49 and Table 5–27 describe the SC_STAT register format.

Figure 5–49 Scache Status (SC_STAT) Register

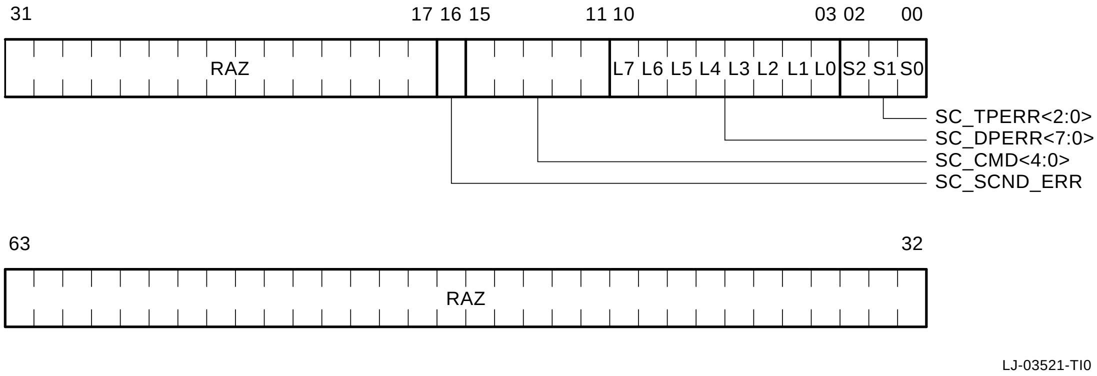


Table 5–27 Scache Status Register Fields

Field	Extent	Type	Description
SC_TPERR<2:0>	<02:00>	RO	When set, these bits indicate that an Scache tag lookup resulted in a tag parity error and identify the set that had the tag parity error.
SC_DPERR<7:0>	<10:03>	RO	When set, these bits indicate that an Scache read transaction resulted in a data parity error and indicate which longword within the two octawords had the data parity error. These bits are loaded if any longword within two octawords read from the Scache during lookup had a data parity error. If SC_FHIT (SC_CTL<00>) is set, this field is used for loading the longword parity bits read out from the Scache.
SC_CMD<4:0>	<15:11>	RO	This field indicates the Scache transaction that resulted in a Scache tag or data parity error. This field is written at the time the actual Scache error bit is written. The Scache transaction may be DREAD, IREAD, or WRITE command from the Mbox, Scache victim command, or the system command being serviced. Refer to Table 5–28 for field encoding.
SC_SCND_ERR	<16>	RO	When set, this bit indicates that an Scache transaction resulted in a parity error while the SC_TPERR or SC_DPERR bit was already set from the earlier transaction. This bit is not set for two errors in different octawords of the same transaction.

5.3 External Interface Control (Cbox) IPRs

Table 5–28 SC_CMD Field Descriptions

SC_CMD<4:3> Source	SC_CMD<2:0> Encoding	Description
1x	110	Set shared from system
	101	Read dirty from system
	100	Invalidate from system
	001	Scache victim
00	001	Scache IREAD
01	001	Scache DREAD
	011	Scache DWRITE

5.3.3 Scache Address (SC_ADDR) Register (FF FFF0 0188)

SC_ADDR is a read-only register. It is not cleared or unlocked by reset. The address is loaded into this register every time the Scache is accessed if one of the error bits in the SC_STAT register is not set. If an Scache tag or data parity error is detected, then this register is locked preventing further updates. This register is unlocked whenever SC_STAT is read.

For Scache read transactions, address bits <39:04> are valid to identify the address being driven to the Scache. Address bit <04> identifies which octaword was accessed first. For each Scache lookup, there is one tag access and two data access cycles. If there is a hit, two octawords are read out in consecutive CPU cycles. Tag parity error is detected only while reading the first octaword. However, data parity error can be detected on either of the two octawords. SC_ADDR<39> is always zero.

If SC_CTL<00> is set (force hit mode), SC_ADDR is used for storing the Scache tag and status bits. For each tag in the Scache, there are unique valid, shared, and dirty bits for a 32-byte subblock, and modify bits for each octaword (16 bytes). There is a single tag and a parity bit for two consecutive 32-byte subblocks. In force hit mode, only reads and probes load tag and status into the SC_ADDR register. In this mode, tag and data parity checking are disabled and the SC_ADDR and SC_STAT registers are not locked on an error.

In force hit mode, to write the Scache and read back the same block and corresponding tag status bits, a minimum of 5-cycle spacing is required between the Scache write and read of the SC_ADDR or SC_STAT.

Figure 5–50 and Table 5–29 describe the SC_ADDR register format.

5.3 External Interface Control (Cbox) IPRs

Figure 5–50 Scache Address (SC_ADDR) Register

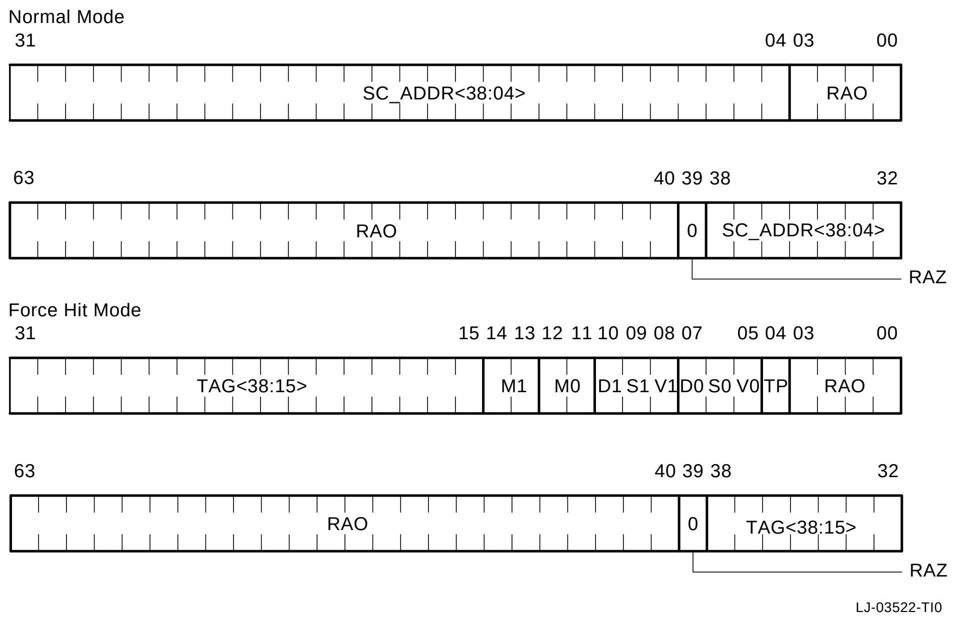


Table 5–29 Scache Address Register Fields

Name	Extent	Type	Description
Normal Mode			
SC_ADDR<38:04>	<38:04>	RO	Scache address.
Force Hit Mode			
TP	<04>	RO	Scache tag parity bit.
V0	<05>	RO	Subblock0 tag valid bit.
S0	<06>	RO	Subblock0 tag shared bit.
D0	<07>	RO	Subblock0 tag dirty bit.
V1	<08>	RO	Subblock1 tag valid bit.
S1	<09>	RO	Subblock1 tag shared bit.
D1	<10>	RO	Subblock1 tag dirty bit.
M0	<12,11>	RO	Octawords modified for subblock0.
M1	<14,13>	RO	Octawords modified for subblock1.
TAG<38:15>	<38:15>	RO	Scache tag.

5.3.4 Bcache Control (BC_CONTROL) Register (FF FFF0 0128)

BC_CONTROL is a write-only register. It is used to enable and control the external Bcache. Figure 5–51 and Table 5–30 describe the BC_CONTROL register format. The bits in this register are initialized to the value indicated in Table 7–2 on reset, but not on timeout reset.

Figure 5–51 Bcache Control (BC_CONTROL) Register

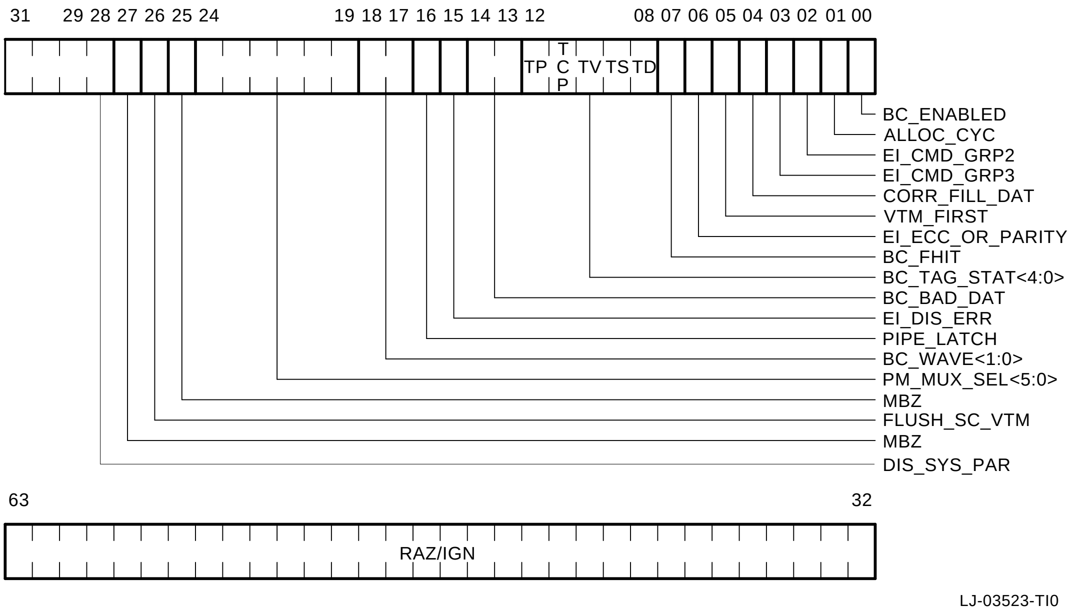


Table 5–30 Bcache Control Register Fields

Field	Extent	Type	Description
BC_ENABLED[1]	<00>	WO,0	When set, the external Bcache is enabled. When clear, the Bcache is disabled. When the Bcache is disabled, the BIU does not perform external cache read or write transactions.
ALLOC_CYC	<01>	WO,0	When set, the issue unit does not allocate a cycle for noncacheable fill data. When clear, the instruction issue unit allocates a cycle for returning noncacheable fill data to be written to the Dcache. In either case, a cycle is always allocated for cacheable integer fill data. If this bit is clear, the latency for all noncacheable read operations increases by 1 CPU cycle. **Note:** This bit *must* be clear before reading any Cbox IPR. It can be set when reading all other IPRs and noncacheable LDs.
EI_CMD_GRP2	<02>	WO,0	When set, the optional commands, LOCK and SET DIRTY are driven to the 21164 external interface command pins to be acknowledged by the system interface. When clear, the SET DIRTY command is not driven to the command pins. It is UNPREDICTABLE if the LOCK command is driven to the pins. However, the system should never CACK the LOCK command if this bit is clear.
EI_CMD_GRP3	<03>	WO,0	When set, the MB command is driven to the 21164 external interface command pins to be acknowledged by the system interface. When clear, the MB command is not driven to the command pins.
CORR_FILL_DAT	<04>	WO,1	Correct fill data from Bcache or main memory, in ECC mode. When set, fill data from Bcache or main memory first goes through error correction logic before being driven to the Scache or Dcache. If the error is correctable, it is transparent to the system. When clear, fill data from Bcache or main memory is driven directly to the Dcache before an ECC error is detected. If the error is correctable, corrected data is returned again, Dcache is invalidated, and an error trap is taken. This bit should be clear during normal operation.

[1]When clear, the read speed (BC_RD_SPD<3:0>) and the write speed (BC_WR_SPD<3:0>) must be equal to the sysclk to CPU clock ratio.

(continued on next page)

Table 5–30 (Cont.) Bcache Control Register Fields

Field	Extent	Type	Description
VTM_FIRST	<05>	WO,1	This bit is set for systems without a victim buffer. On a Bcache miss, the 21164 first drives out the victimized block's address on the system address bus, followed by the read miss address and command. This bit is cleared for systems with a victim buffer. On a Bcache miss with victim, the 21164 first drives out the read miss followed by the victim address and command.
EI_ECC_OR_PARITY	<06>	WO,1	When set, the 21164 generates or expects quadword ECC on the data check pins. When clear, the 21164 generates or expects even-byte parity on the data check pins.
BC_FHIT	<07>	WO,0	Bcache force hit. When set, and the Bcache is enabled, all references in cached space are forced to hit in the Bcache. A FILL to the Scache is forced to be private. Software should turn off BC_CONTROL<02> to allow clean to private transitions without going to the system. For write transactions, the values of tag status and parity bits are specified by the BC_TAG_STAT field. Bcache tag and index are the address received by the BIU. The Bcache tag RAMs are written with the address minus the Bcache index. This bit must be zero during normal operation.
BC_TAG_STAT<4:0>	<12:08>	WO	This bit field is used only in BC_FHIT=1 mode to write any combination of tag status and parity bits in the Bcache. The parity bit can be used to write bad tag parity. These bits are UNDEFINED on reset. This bit field must be zero during normal operation. The field encoding is as follows:

(continued on next page)

Table 5–30 (Cont.) Bcache Control Register Fields

Field	Extent	Type	Description

Bcache Tag Status Bit	Description
BC_TAG_STAT<4>	Parity for Bcache tag
BC_TAG_STAT<3>	Parity for Bcache tag status bits
BC_TAG_STAT<2>	Bcache tag valid bit
BC_TAG_STAT<1>	Bcache tag shared bit
BC_TAG_STAT<0>	Bcache tag dirty bit

Field	Extent	Type	Description
BC_BAD_DAT	<14:13>	WO,0	When set, bits in this field can be used to write bad data with correctable or uncorrectable errors in ECC mode. When bit <13> is set, data bit <0> and <64> are inverted. When bit <14> is set, data bit <1> and <65> are inverted. When the same octaword is read from the Bcache, the 21164 detects a correctable/uncorrectable ECC error on both the quadwords based on the value of bits <14:13> used when writing. This bit field must be zero during normal operation.
EI_DIS_ERR	<15>	WO,1	When set, this bit causes the 21164 to ignore any ECC (parity) error on fill data received from the Bcache or main memory; or Bcache tag or control parity error. It also ignores a system command/address parity error. No machine check is taken when this bit is set.
PIPE_LATCH	<16>	WO,0	When set, this bit causes the 21164 to pipe the system control pins (**addr_bus_req_h**, **cack_h**, and **dack_h**) for one system clock. Refer to Chapter 9 for timing details.

(continued on next page)

Table 5–30 (Cont.) Bcache Control Register Fields

Field	Extent	Type	Description
BC_WAVE<1:0>	<18:17>	WO,0	The bits in this field determine the number of cycles of wave pipelining that should be used during private read transactions of the Bcache. Wave pipelining cannot be used in 32-byte block systems. To enable wave pipelining, BC_CONFIG<07:04> should be set to the latency of the Bcache read. BC_CONTROL<18:17> should be set to the number of cycles to subtract from BC_CONFIG<07:04> to obtain the Bcache repetition rate. For example, if BC_CONFIG<07:04>=7 and BC_CONTROL<18:17>=2, it takes seven cycles for valid data to arrive at the interface pins, but a new read will start every five cycles. The read repetition rate must be greater than 3. For example, it is not permitted to set BC_CONFIG<07:04>=5 and BC_CONTROL<18:17>=2. The value of BC_CONTROL<18:17> should be added to the normal value of BC_CONFIG<14:12> to increase the time between read and write transactions. This prevents a write transaction from starting before the last data of a read transaction is received.
PM_MUX_ SEL<5:0>	<24:19>	WO,0	The bits in this field are used for selecting the BIU parameters to be driven to the two performance monitoring counters in the Ibox. Refer to Table 5–31 for the field encoding.
Reserved	<25>	WO,0	Reserved—MBZ.
FLUSH_SC_VTM	<26>	WO,0	Flush Scache victim buffer. For systems without a Bcache, when this bit is clear, the 21164 flushes the onchip victim buffer if it has to write-back any entry from the victim buffer. When this bit is set, the 21164 writes only one entry back from the victim buffer as needed. This tends to cause read and write operations to be batched rather than interleaved. For systems with a Bcache, this bit must always be clear. At power-up, this bit is initialized to a value of 0.
Reserved	<27>	WO,0	Reserved—MBZ.

(continued on next page)

Table 5–30 (Cont.) Bcache Control Register Fields

Field	Extent	Type	Description
DIS_SYS_PAR	<28>	WO,0	When set, the 21164 does not check parity on the system command/address bus. However, correct parity will still be generated.

Table 5–31 describes the PM_MUX_SEL fields.

Table 5–31 PM_MUX_SEL Register Fields

PM_MUX_SEL<21:19>	Counter 1
0x0	Scache accesses
0x1	Scache read operations
0x2	Scache write operations
0x3	Scache victims
0x4	Undefined
0x5	Bcache accesses
0x6	Bcache victims
0x7	System command requests

PM_MUX_SEL<24:22>	Counter 2
0x0	Scache misses
0x1	Scache read misses
0x2	Scache write misses
0x3	Scache shared write operations
0x4	Scache write operations
0x5	Bcache misses
0x6	System invalidate operations
0x7	System read requests

5.3.5 Bcache Configuration (BC_CONFIG) Register (FF FFF0 01C8)

BC_CONFIG is a write-only register used to configure the size and speed of the external Bcache array. The bits in this register are initialized to the values indicated in Table 5–32 on reset, but not on timeout reset. Figure 5–52 and Table 5–32 describe the BC_CONFIG register format.

Figure 5–52 Bcache Configuration (BC_CONFIG) Register

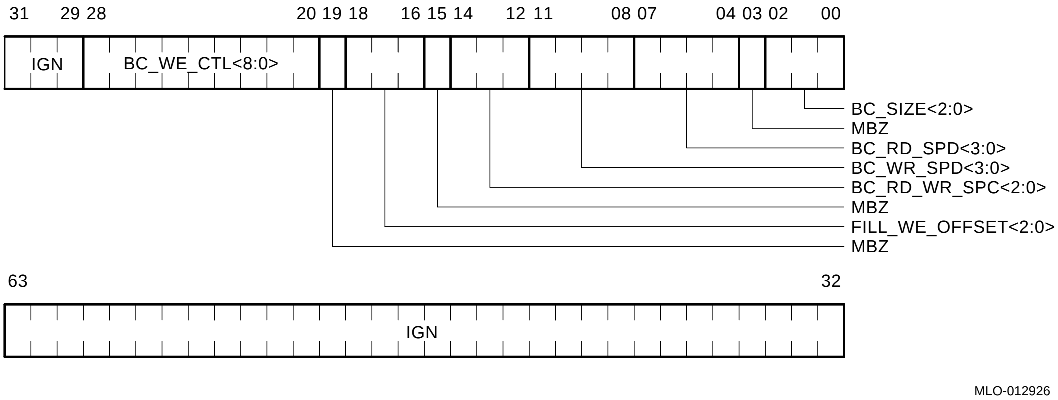


Table 5–32 Bcache Configuration Register Fields

<table>
<tr><th>Field</th><th>Extent</th><th>Type</th><th>Description</th></tr>
<tr><td>BC_SIZE<2:0></td><td><02:00></td><td>WO,1</td><td>The bits in this field are used to indicate the size of the Bcache. At power-on, this field is initialized to a value representing a 1M-byte Bcache. The field encoding is as follows:
<table>
<tr><th>BC_ SIZE<2:0>[1]</th><th>Size</th></tr>
<tr><td>000</td><td>Invalid Bcache size</td></tr>
<tr><td>001</td><td>1 MB</td></tr>
<tr><td>010</td><td>2 MB</td></tr>
<tr><td>011</td><td>4 MB</td></tr>
<tr><td>100</td><td>8 MB</td></tr>
<tr><td>101</td><td>16 MB</td></tr>
<tr><td>110</td><td>32 MB</td></tr>
<tr><td>111</td><td>64 MB</td></tr>
</table>
</td></tr>
<tr><td>Reserved</td><td><03></td><td>WO,0</td><td>Must be zero (MBZ).</td></tr>
</table>

(continued on next page)

Table 5–32 (Cont.) Bcache Configuration Register Fields

Field	Extent	Type	Description
BC_RD_SPD<3:0>	<07:04>	WO,4	The bits in this field are used to indicate to the BIU the read access time of the Bcache, measured in CPU cycles, from the start of a read transaction until data is valid at the input pins. The Bcache read speed must be within 4 to 10 CPU cycles. At power-up, this field is initialized to a value of 4 CPU cycles. The Bcache read and write speeds must be within three cycles of each other ($\mathtt{absolute\ value} = (\mathrm{BC\_RD\_SPD} - \mathrm{BC\_WR\_SPD}) < 4$). For systems without a Bcache, the read speed must be equal to the sysclk to CPU clock ratio. In this configuration, BC_RD_SPD can be set to a value ranging from 3 to 15.
BC_WR_SPD<3:0>	<11:08>	WO,4	The bits in this field are used to indicate to the BIU the write time of the Bcache, measured in CPU cycles. The Bcache write speed must be within 4 to 10 CPU cycles. At power-up, this field is initialized to a value of four CPU cycles. For systems without a Bcache, the write speed must be equal to sysclk to CPU clock ratio.

(continued on next page)

Table 5–32 (Cont.) Bcache Configuration Register Fields

Field	Extent	Type	Description
BC_RD_WR_ SPC<2:0>	<14:12>	WO,7	The bits in this field are used to indicate to the BIU the number of CPU cycles to wait when switching from a private read to a private write Bcache transaction. For other data movement commands, such as READ DIRTY or FILL from main memory, it is up to the system to direct systemwide data movement in a way that is safe. A value of 1 must be the minimum value for this field. The BIU always inserts three CPU cycles between private Bcache read and private Bcache write transactions, in addition to the number of CPU cycles specified by this field. The maximum value (BC_RD_WR_SPC+3) should not be greater than the Bcache READ speed when Bcache is enabled. At power-up, this field is initialized to a read/write spacing of seven CPU cycles.
Reserved	<15>	WO,0	Must be zero (MBZ).
FILL_WE_ OFFSET<2:0>	<18:16>	WO,1	Bcache write-enable pulse offset, from the **sys_clk_out*n_x*** edge, for FILL transactions from the system. This field does not affect private write transactions to Bcache. It is used during FILLs from the system when writing the Bcache to determine the number of CPU cycles to wait before shifting out the contents of the write pulse field. This field is programmed with a value in the range of one to seven CPU cycles. It must never exceed the sysclk ratio. For example, if the sysclk ratio is 3, this field must not be larger than 3. At power-up, this field is initialized to a write offset value of one CPU cycle.
Reserved	<19>	WO,0	Must be zero (MBZ).

(continued on next page)

Table 5–32 (Cont.) Bcache Configuration Register Fields

Field	Extent	Type	Description
BC_WE_CTL<8:0>	<28:20>	WO,0	Bcache write-enable control. This field is used to control the timing of the write-enable during a write or FILL transaction. If the bit is set, the write pulse is asserted. If the bit is clear, the write pulse is not asserted. Each bit corresponds to a CPU cycle. The least-significant bit corresponds to the CPU cycle in which the 21164 starts to drive the index for the write operation. For private Bcache write and shared-write transactions, this field is used to assert the write pulse without any write-enable pulse offset as indicated by the FILL_WE_OFFSET<2:0> field. For FILLs to the Bcache, the FILL_WE_OFFSET<2:0> field determines the number of CPU cycles to wait before asserting the write pulse as programmed in this field. At power-up, all bits in this field are cleared.
Reserved	<63:29>	WO	Ignored.

5.3.6 Bcache Tag Address (BC_TAG_ADDR) Register (FF FFF0 0108)

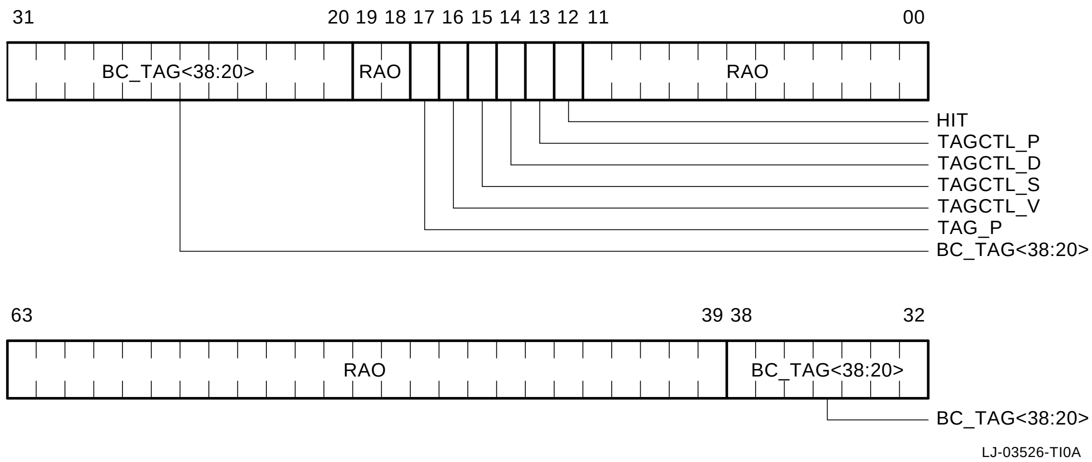
BC_TAG_ADDR is a read-only register. Unless locked, the BC_TAG_ADDR register is loaded with the results of every Bcache tag read. When a tag or tag control parity error occurs, this register is locked against further updates. Software may read this register by using the 21164-specific I/O space address instruction. This register is unlocked whenever the EI_STAT register is read, or the user enters BC_FHIT mode. It is not unlocked by reset.

Note

The correct address is not loaded into BC_TAG_ADDR if a tag parity error is detected when servicing a system command from the Bcache.

Unused tag bits in the TAG field of this register are always zero, based on the size of the Bcache as determined by the BC_SIZE field of the BC_CONTROL register. Figure 5–53 and Table 5–33 describe the BC_TAG_ADDR register format.

Figure 5–53 Bcache Tag Address (BC_TAG_ADDR) Register



5.3 External Interface Control (Cbox) IPRs

Table 5–33 Bcache Tag Address Register Fields

Field	Extent	Type	Description
HIT	<12>	RO	If set, Bcache access resulted in a hit in the Bcache.
TAGCTL_P	<13>	RO	Value of the parity bit for the Bcache tag status bits.
TAGCTL_D	<14>	RO	Value of the Bcache TAG dirty bit.
TAGCTL_S	<15>	RO	Value of the Bcache TAG shared bit.
TAGCTL_V	<16>	RO	Value of the Bcache TAG valid bit.
TAG_P	<17>	RO	Value of the tag parity bit.
BC_TAG<38:20>	<38:20>	RO	Bcache tag bits as read from the Bcache. Unused bits are read as zero.

5.3.7 External Interface Status (EI_STAT) Register (FF FFF0 0168)

EI_STAT is a read-only register. Any PALcode read access of this register unlocks and clears it. A read access of EI_STAT also unlocks the EI_ADDR, BC_TAG, and FILL_SYN registers subject to some restrictions. The EI_STAT register is not unlocked or cleared by reset.

Fill data from Bcache or main memory could have correctable (c) or uncorrectable (u) errors in ECC mode. In parity mode, fill data parity errors are treated as uncorrectable hard errors. System address/command parity errors are always treated as uncorrectable hard errors irrespective of the mode. The sequence for reading, unlocking, and clearing EI_ADDR, BC_TAG, FILL_SYN, and EI_STAT is as follows:

1. Read EI_ADDR, BC_TAG, and FILL_SYN in any order. Does not unlock or clear any register.
2. Read EI_STAT register. Reading this register unlocks EI_ADDR, BC_TAG, and FILL_SYN registers. EI_STAT is also unlocked and cleared when read, subject to conditions described in Table 5–34.

Loading and locking rules for external interface registers are defined in Table 5–34.

Note

If the first error is correctable, the registers are loaded but not locked. On the second correctable error, registers are neither loaded nor locked.

Registers are locked on the first uncorrectable error except the second hard error bit. The second hard error bit is set only for an uncorrectable error followed by an uncorrectable error. If a correctable error follows an uncorrectable error, it is not logged as a second error. Bcache tag parity errors are uncorrectable in this context.

5.3 External Interface Control (Cbox) IPRs

Table 5–34 Loading and Locking Rules for External Interface Registers

Correctable Error	Uncorrectable Error	Second Hard Error	Load Register	Lock Register	Action when EI_STAT is read
0	0	Not possible	No	No	Clears and unlocks everything.
1	0	Not possible	Yes	No	Clears and unlocks everything.
0	1	0	Yes	Yes	Clears and unlocks everything.
1[1]	1	0	Yes	Yes	Clear (c) bit does not unlock. Transition to (0,1,0) state.
0	1	1	No	Already locked	Clears and unlocks everything.
1[1]	1	1	No	Already locked	Clear (c) bit does not unlock. Transition to (0,1,1) state.

[1]These are special cases. It is possible that when EI_ADDR is read, only the correctable error bit is set and the registers are not locked. By the time EI_STAT is read, an uncorrectable error is detected and the registers are loaded again and locked. The value of EI_ADDR read earlier is no longer valid. Therefore, for the (1,1,x) case, when EI_STAT is read correctable, the error bit is cleared and the registers are not unlocked or cleared. Software must reexecute the IPR read sequence. On the second read operation, error bits are in (0,1,x) state, all the related IPRs are unlocked, and EI_STAT is cleared.

The EI_STAT register is a read-only register used to control external interface registers. Figure 5–54 and Table 5–35 describe the EI_STAT register format.

Figure 5–54 External Interface Status (EI_STAT) Register

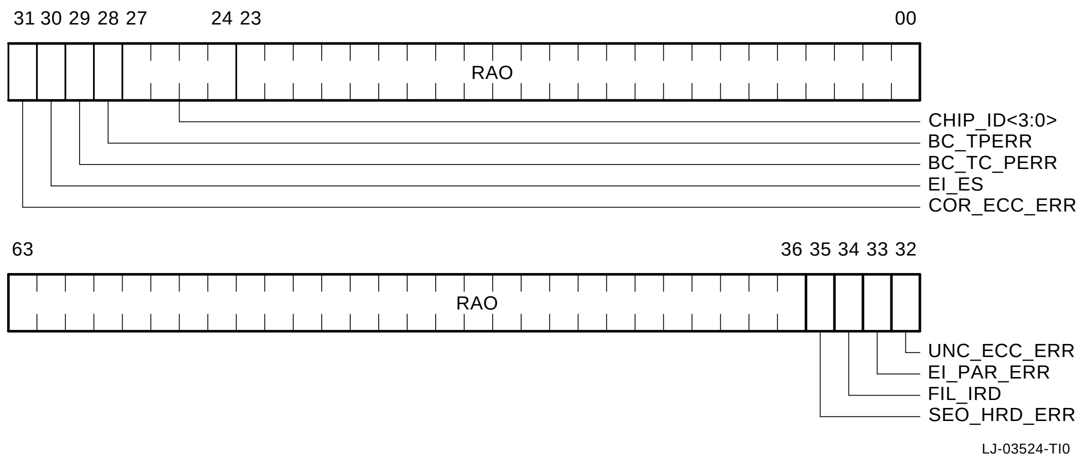


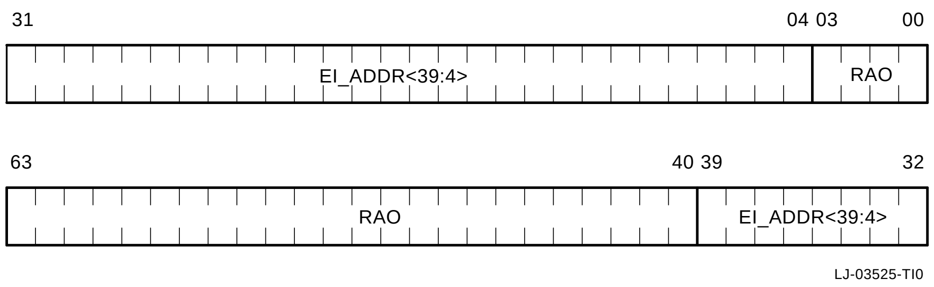
Table 5–35 EI_STAT Register Fields

Field	Extent	Type	Description
CHIP_ID<3:0>	<27:24>	RO	Read as ‘‘4.’’ Future update revisions to the chip will return new unique values.
BC_TPERR	<28>	RO	Indicates that a Bcache read transaction encountered bad parity in the tag address RAM.
BC_TC_PERR	<29>	RO	Indicates that a Bcache read transaction encountered bad parity in the tag control RAM.
EI_ES	<30>	RO	When set, this bit indicates that the error source is fill data from main memory or a system address/command parity error. When clear, the error source is fill data from the Bcache. This bit is only meaningful when COR_ECC_ERR, UNC_ECC_ERR, or EI_PAR_ERR is set. This bit is not defined for a Bcache tag error (BC_TPERR) or a Bcache tag control parity error (BC_TC_ERR).
COR_ECC_ERR	<31>	RO	Correctable ECC error. This bit indicates that a fill data received from outside the CPU contained a correctable ECC error.
UNC_ECC_ERR	<32>	RO	Uncorrectable ECC error. This bit indicates that fill data received from outside the CPU contained an uncorrectable ECC error. In the parity mode, it indicates data parity error.
EI_PAR_ERR	<33>	RO	External interface command/address parity error. This bit indicates that an address and command received by the CPU has a parity error.
FIL_IRD	<34>	RO	This bit has meaning only when one of the ECC or parity error bit is set. It is set to indicate that the error occurred during an I-ref FILL and clear to indicate that the error occurred during a D-ref FILL. This bit is not defined for a Bcache tag error (BC_TPERR) or a Bcache tag control parity error (BC_TC_ERR).
SEO_HRD_ERR	<35>	RO	Second external interface hard error. This bit indicates that a FILL from Bcache or main memory, or a system address/command received by the CPU has a hard error while one of the hard error bits in the EI_STAT register is already set.

5.3.8 External Interface Address (EI_ADDR) Register (FF FFF0 0148)

EI_ADDR is a read-only register that contains the physical address associated with errors reported by the EI_STAT register. Its content is meaningful only when one of the error bits is set. A read of EI_STAT unlocks the EI_ADDR register. Figure 5–55 shows the EI_ADDR register format.

Figure 5–55 External Interface Address (EI_ADDR) Register



5.3.9 Fill Syndrome (FILL_SYN) Register (FF FFF0 0068)

FILL_SYN is a 16-bit read-only register. It is loaded but not locked on a correctable ECC error, so that another correctable error does not reload it. It is loaded and locked if an uncorrectable ECC error or parity error is recognized during a FILL from Bcache or main memory, as shown in Table 5–34. The FILL_SYN register is unlocked when the EI_STAT register is read. This register is not unlocked by reset.

If the 21164 is in ECC mode and an ECC error is recognized during a cache fill transaction, the syndrome bits associated with the bad quadword are loaded in the FILL_SYN register. FILL_SYN<07:00> contains the syndrome associated with the lower quadword of the octaword. FILL_SYN<15:08> contains the syndrome associated with the higher quadword of the octaword. A syndrome value of 0 means that no errors where found in the associated quadword.

If the 21164 is in parity mode and a parity error is recognized during a cache fill transaction, the FILL_SYN register indicates which of the bytes in the octaword has bad parity. FILL_SYNDROME<07:00> is set appropriately to indicate the bytes within the lower quadword that were corrupted. Likewise, FILL_SYN<15:08> is set to indicate the corrupted bytes within the upper quadword. Figure 5–56 shows the FILL_SYN register format.

5.3 External Interface Control (Cbox) IPRs

Figure 5–56 Fill Syndrome (FILL_SYN) Register

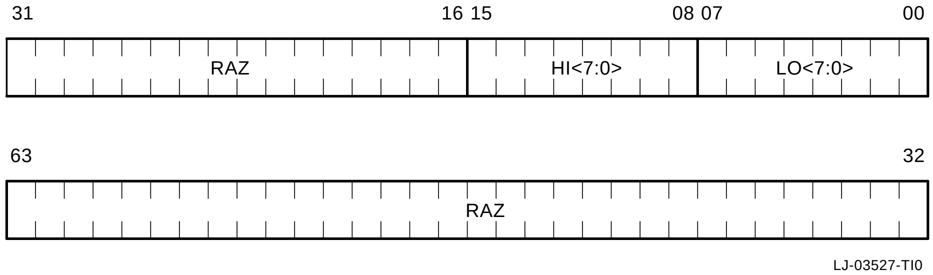


Table 5–36 lists the syndromes associated with correctable single-bit errors.

Table 5–36 Syndromes for Single-Bit Errors

Data Bit	$Syndrome_{16}$	Check Bit	$Syndrome_{16}$
00	CE	00	01
01	CB	01	02
02	D3	02	04
03	D5	03	08
04	D6	04	10
05	D9	05	20
06	DA	06	40
07	DC	07	80
08	23		
09	25		
10	26		
11	29		
12	2A		
13	2C		
14	31		
15	34		
16	0E		
17	0B		

(continued on next page)

Table 5–36 (Cont.) Syndromes for Single-Bit Errors

Data Bit	Syndrome$_{16}$	Check Bit	Syndrome$_{16}$
18	13		
19	15		
20	16		
21	19		
22	1A		
23	1C		
24	E3		
25	E5		
26	E6		
27	E9		
28	EA		
29	EC		
30	F1		
31	F4		
32	4F		
33	4A		
34	52		
35	54		
36	57		
37	58		
38	5B		
39	5D		
40	A2		
41	A4		
42	A7		
43	A8		
44	AB		
45	AD		
46	B0		

(continued on next page)

Table 5–36 (Cont.) Syndromes for Single-Bit Errors

Data Bit	$Syndrome_{16}$	Check Bit	$Syndrome_{16}$
47	B5		
48	8F		
49	8A		
50	92		
51	94		
52	97		
53	98		
54	9B		
55	9D		
56	62		
57	64		
58	67		
59	68		
60	6B		
61	6D		
62	70		
63	75		

5.4 PALcode Storage Registers

The 21164 Ebox register file has eight extra registers that are called the PALshadow registers. The PALshadow registers overlay R8 through R14 and R25 when the CPU is in PALmode and ICSR<SDE> is set. Thus, PALcode can consider R8 through R14 and R25 as local scratch. PALshadow registers can not be written in the last two cycles of a PALcode flow. The normal state of the CPU is ICSR<SDE> = ON. PALcode disables SDE for the unaligned trap and for error flows.

The Ibox holds a bank of 24 PALtemp registers. The PALtemp registers are accessed with the HW_MTPR and HW_MFPR instructions. The latency from a PALtemp read operation to availability is one cycle.

5.5 Restrictions

The following sections list all known register access restrictions. A software tool called the PALcode violation checker (PVC) is available. This tool can be used to verify adherence to many of the PALcode restrictions.

5.5.1 Cbox IPR PALcode Restrictions

Table 5–37 describes the Cbox IPR PALcode restrictions.

Table 5–37 Cbox IPR PALcode Restrictions

Condition	Restriction
Store to SC_CTL, BC_CONTROL, BC_CONFIG except if no bit is changed other than BC_CONTROL<ALLOC_CYC>, BC_CONTROL<PM_MUX_SEL>, or BC_CONTROL<DBG_MUX_SEL>.	Must be preceded by MB, must be followed by MB, must have no concurrent cacheable Istream references or concurrent system commands.
Store to BC_CONTROL that only changes bits BC_CONTROL<ALLOC_CYC>, BC_CONTROL<PM_MUX_SEL>, or BC_CONTROL<DBG_MUX_SEL>.	Must be preceded by MB and must be followed by MB.
Load from SC_STAT.	Unlocks SC_ADDR and SC_STAT.
Load from EI_STAT.	Unlocks EI_ADDR, EI_STAT, FILL_SYN, and BC_TAG_ADDR.
Any Cbox IPR address.	No LD*x*_L or ST*x*_C.
Any undefined Cbox IPR address.	No store instructions.
Scache or Bcache in force hit mode.	No ST*x*_C to cacheable space.
Clearing of SC_FHIT in SC_CTL.	Must be followed by MB, read operation of SC_STAT, then MB prior to subsequent store.
Clearing of BC_FHIT in BC_CONTROL.	Must be followed by MB, read operation of EI_STAT, then MB prior to subsequent store.
Load from any Cbox IPR.	BC_CONTROL<01> (ALLOC_CYCLE) must be clear.

5.5.2 PALcode Restrictions—Instruction Definitions

Mbox instructions are: LD*x*, LDQ_U, LD*x*_L, HW_LD, ST*x*, STQ_U, ST*x*_C, HW_ST, and FETCHx.

Virtual Mbox instructions are: LD*x*, LDQ_U, LD*x*_L, HW_LD (virtual), ST*x*, STQ_U, ST*x*_C, HW_ST (virtual), and FETCHx.

Load instructions are: LD*x*, LDQ_U, LD*x*_L, and HW_LD.

Store instructions are: ST*x*, STQ_U, ST*x*_C, and HW_ST.

Table 5–38 lists PALcode restrictions.

Table 5–38 PALcode Restrictions Table

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
CALL_PAL entry	No HW_REI or HW_REI_STALL in cycle 0.	Y
	No HW_MFPR EXC_ADDR in cycle 0,1.	Y
PALshadow write instruction	No HW_REI or HW_REI_STALL in 0, 1.	Y
HW_LD, lock bit set	PAL must slot to E0.	
	No other Mbox instruction in 0.	
HW_LD, VPTE bit set	No other virtual reference in 0.	
Any load instruction	No Mbox HW_MTPR or HW_MFPR in 0.	Y
	No HW_MFPR MAF_MODE in 1,2 (DREAD_PENDING may not be updated).	Y
	No HW_MFPR DC_PERR_STAT in 1,2.	Y
	No HW_MFPR DC_TEST_TAG slotted in 0.	
Any store instruction	No HW_MFPR DC_PERR_STAT in 1,2.	Y
	No HW_MFPR MAF_MODE in 1,2 (WB_PENDING may not be updated).	Y
Any virtual Mbox instruction	No HW_MTPR DTB_IS in 1.	Y
Any Mbox instruction or WMB, if it traps	HW_MTPR any Ibox IPR not aborted in 0,1 (except that EXC_ADDR is updated with correct faulting PC).	Y
	HW_MTPR DTB_IS not aborted in 0,1.	Y
Any Ibox trap except PC-mispredict, ITBMISS, or OPCDEC due to user mode	HW_MTPR DTB_IS not aborted in 0,1.	
HW_REI_STALL	Only one HW_REI_STALL in an aligned block of four instructions.	

[1]PALcode violation checker

(continued on next page)

5.5 Restrictions

Table 5–38 (Cont.) PALcode Restrictions Table

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR any undefined IPR number	Illegal in any cycle.	
ARITH trap entry	No HW_MFPR EXC_SUM or EXC_MASK in cycle 0,1.	Y
Machine check trap entry	No register file read or write access in 0,1,2,3,4,5,6,7. No HW_MFPR EXC_SUM or EXC_MASK in cycle 0,1.	 Y
HW_MTPR any Ibox IPR (including PALtemp registers)	No HW_MFPR same IPR in cycle 1,2. No floating-point conditional branch in 0. No FEN or OPCDEC instruction in 0.	Y
HW_MTPR ASTRR, ASTER	No HW_MFPR INTID in 0,1,2,3,4,5. No HW_REI in 0,1.	Y Y
HW_MTPR SIRR	No HW_MFPR INTID in 0,1,2,3,4.	Y
HW_MTPR EXC_ADDR	No HW_REI in cycle 0,1.	Y
HW_MTPR IC_FLUSH_CTL	Must be followed by 44 inline PALcode instructions.	
HW_MTPR ICSR: HWE	No HW_REI in 0,1,2,3.	Y
HW_MTPR ICSR: FPE	No floating-point instructions in 0, 1, 2, 3. No HW_REI in 0,1,2.	
HW_MTPR ICSR: SPE, FMS	If HW_REI_STALL, then no HW_REI_STALL in 0,1. If HW_REI, then no HW_REI in 0,1,2,3,4.	Y Y
HW_MTPR ICSR: SPE	Must flush Icache.	
HW_MTPR ICSR: SDE	No PALshadow read/write access in 0,1,2,3. No HW_REI in 0,1,2.	 Y
HW_MTPR ITB_ASN	Must be followed by HW_REI_STALL. No HW_REI_STALL in cycle 0,1,2,3,4. No HW_MTPR ITB_IS in 0,1,2,3.	 Y Y
HW_MTPR ITB_PTE	Must be followed by HW_REI_STALL.	
HW_MTPR ITB_IAP, ITB_IS, ITB_IA	Must be followed by HW_REI_STALL.	
HW_MTPR ITB_IS	HW_REI_STALL must be in the same Istream octaword.	
HW_MTPR IVPTBR	No HW_MFPR IFAULT_VA_FORM in 0,1,2.	Y
HW_MTPR PAL_BASE	No CALL_PAL in 0,1,2,3,4,5,6,7. No HW_REI in 0,1,2,3,4,5,6.	Y Y
HW_MTPR ICM	No HW_REI in 0,1,2. No private CALL_PAL in 0,1,2,3.	Y

[1]PALcode violation checker

(continued on next page)

Table 5–38 (Cont.) PALcode Restrictions Table

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR CC, CC_CTL	No RPCC in 0,1,2.	Y
	No HW_REI in 0,1.	Y
HW_MTPR DC_FLUSH	No Mbox instructions in 1,2.	Y
	No outstanding fills in 0.	
	No HW_REI in 0,1.	Y
HW_MTPR DC_MODE	No Mbox instructions in 1,2,3,4.	Y
	No HW_MFPR DC_MODE in 1,2.	Y
	No outstanding fills in 0.	
	No HW_REI in 0,1,2,3.	Y
	No HW_REI_STALL in 0,1.	Y
HW_MTPR DC_PERR_STAT	No load or store instructions in 1.	Y
	No HW_MFPR DC_PERR_STAT in 1,2.	Y
HW_MTPR DC_TEST_CTL	No HW_MFPR DC_TEST_TAG in 1,2,3.	Y
	No HW_MFPR DC_TEST_CTL issued or slotted in 1,2.	
HW_MTPR DC_TEST_TAG	No outstanding DC fills in 0.	
	No HW_MFPR DC_TEST_TAG in 1,2,3.	Y
HW_MTPR DTB_ASN	No virtual Mbox instructions in 1,2,3.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR DTB_CM, ALT_MODE	No virtual Mbox instructions in 1,2.	Y
	No HW_REI in 0,1.	Y
HW_MTPR DTB_PTE	No virtual Mbox instructions in 2.	Y
	No HW_MTPR DTB_ASN, DTB_CM, ALT_MODE, MCSR, MAF_MODE, DC_MODE, DC_PERR_STAT, DC_TEST_CTL, DC_TEST_TAG in 2.	Y
HW_MTPR DTB_TAG	No virtual Mbox instructions in 1,2,3.	Y
	No HW_MTPR DTB_TAG in 1.	Y
	No HW_MFPR DTB_PTE in 1,2.	Y
	No HW_MTPR DTB_IS in 1,2.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR DTB_IAP, DTB_IA	No virtual Mbox instructions in 1,2,3.	Y
	No HW_MTPR DTB_IS in 0,1,2.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR DTB_IA	No HW_MFPR DTB_PTE in 1.	Y
HW_MTPR MAF_MODE	No Mbox instructions in 1,2,3.	Y
	No WMB in 1,2,3.	Y
	No HW_MFPR MAF_MODE in 1,2.	Y
	No HW_REI in 0,1,2.	Y

[1]PALcode violation checker

(continued on next page)

5.5 Restrictions

Table 5–38 (Cont.) PALcode Restrictions Table

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR MCSR	No virtual Mbox instructions in 0,1,2,3,4.	Y
	No HW_MFPR MCSR in 1,2.	Y
	No HW_MFPR VA_FORM in 1,2,3.	Y
	No HW_REI in 0,1,2,3.	Y
	No HW_REI_STALL in 0,1.	Y
HW_MTPR MVPTBR	No HW_MFPR VA_FORM in 1,2.	Y
HW_MFPR ITB_PTE	No HW_MFPR ITB_PTE_TEMP in 1,2,3.	Y
HW_MFPR DC_TEST_TAG	No outstanding DC fills in 0.	
	No HW_MFPR DC_TEST_TAG_TEMP issued or slotted in 1.	
	No LD*x* instructions slotted in 0.	
	No HW_MTPR DC_TEST_CTL between HW_MFPR DC_TEST_TAG and HW_MFPR DC_TEST_TAG_TEMP.	
HW_MFPR DTB_PTE	No Mbox instructions in 0,1.	Y
	No HW_MTPR DC_TEST_CTL, DC_TEST_TAG in 0,1.	Y
	No HW_MFPR DTB_PTE_TEMP issued or slotted in 1,2,3.	
	No HW_MFPR DTB_PTE in 1.	Y
	No virtual Mbox instructions in 0,1,2.	Y
HW_MFPR VA	Must be done in ARITH, MACHINE CHECK, DTBMISS_SINGLE, UNALIGN, DFAULT traps and ITBMISS flow after the VPTE load.	

[1]PALcode violation checker

6
Privileged Architecture Library Code

This chapter describes the 21164 privileged architecture library code (PALcode). The chapter is organized as follows:

- PALcode description
- PALmode environment
- Invoking PALcode
- PALcode entry points
- Required PALcode function codes
- 21164 implementation of the architecturally reserved opcodes

6.1 PALcode Description

Privileged architecture library code (PALcode) is macrocode that provides an architecturally defined operating-system-specific programming interface that is common across all Alpha microprocessors. The actual implementation of PALcode differs for each operating system.

PALcode runs with privileges enabled, instruction stream mapping disabled, and interrupts disabled. PALcode has privilege to use five special opcodes that allow functions such as physical data stream references and internal processor register (IPR) manipulation.

PALcode can be invoked by the following events:

- Reset
- System hardware exceptions (MCHK, ARITH)
- Memory-management exceptions
- Interrupts
- CALL_PAL instructions

6.1 PALcode Description

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, but that cannot be handled by a normal operating system software routine. Routines to fill the translation buffer (TB), acknowledge interrupts, and dispatch exceptions are some examples. In some architectures, these functions are handled by microcode, but the Alpha architecture is careful not to mandate the use of microcode so as to allow reasonable chip implementations.
- 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 VAX style interlocked memory access might be familiar to someone used to programming on a CISC machine, but is not included in the Alpha 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 Istream code in the same way that all other Alpha code is read. Once invoked, however, PALcode runs in a special mode called PALmode.

6.2 PALmode Environment

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

- Istream memory mapping is disabled. Because the PALcode is used to implement translation buffer fill routines, Istream mapping clearly cannot be enabled. Dstream mapping is still 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.

An important aspect of PALcode is that it uses normal Alpha instructions for most of its operations; that is, the same instruction set that nonprivileged Alpha programmers use. There are a few extra instructions that are only available in PALmode, and will cause a dispatch to the OPCDEC PALcode entry point if attempted while not in PALmode. The Alpha architecture allows some flexibility in what these special PALmode instructions do. In the 21164 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).
- Return from an exception or interrupt (HW_REI) .

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 21164. Refer to Section 6.6 for information on these special PALmode instructions.

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.

6.3 Invoking PALcode

PALcode is invoked at specific entry points, under certain well-defined conditions. These entry points provide access to 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 IPR), and is normally set by the system reset code. Refer to Section 6.4 for additional information on PALcode entry points.

PC<00> is used as the PALmode flag both to the hardware and to PALcode itself. When the CPU enters a PALflow, the Ibox sets PC<00>. This bit remains set as instructions are executed in the PAL Istream. The Ibox hardware ignores this and behaves as if the PC were still longword aligned for the purposes of Istream fetch and execute. On HW_REI, the new state of PALmode is copied from EXC_ADDR<00>.

6.3 Invoking PALcode

When an event occurs that needs to invoke PALcode, the 21164 first drains the pipeline. The current PC is loaded into the EXC_ADDR IPR, and the appropriate PALcode routine is dispatched. 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 EXC_ADDR IPR. The LSB is used to indicate PALmode to the hardware. Generally, the LSB is clear upon return from a PALcode routine, in which case, the hardware loads 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 invocations of PALcode:

- When the 21164 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<00> is cleared. It then continues execution in non-PALmode (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 21164, 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 21164 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 Dstream or Istream translation buffer miss occurs, one of several PALcode routines is called to perform the TB fill.

The 21164 Ebox register file has eight extra registers that are called the PALshadow registers. The PALshadow registers overlay R8, R9, R10, R11, R12, R13, R14, and R25 when the CPU is in PALmode and ICSR<SDE> is asserted. For additional PAL scratch, the Ibox has a register bank of 24 PALtemp registers, which are accessible via HW_MTPR and HW_MFPR instructions.

6.4 PALcode Entry Points

PALcode is invoked at specific entry points. The 21164 has two types of PALcode entry points: CALL_PAL and traps.

6.4.1 CALL_PAL Entry

CALL_PAL entry points are used whenever the Ibox encounters a CALL_PAL instruction in the instruction stream (Istream). CALL_PAL instructions start at the following offsets:

- Privileged CALL_PAL instructions start at offset 2000_{16}.
- Nonnprivileged CALL_PAL instructions start at offset 3000_{16}.

The CALL_PAL itself is issued into pipe E1 and the Ibox stalls for the minimum number of cycles necessary to perform an implicit TRAPB. The PC of the instruction immediately following the CALL_PAL is loaded into EXC_ADDR and is pushed onto the return prediction stack.

The Ibox contains special hardware to minimize the number of cycles in the TRAPB at the start of a CALL_PAL. Software can benefit from this by scheduling CALL_PALs such that they do not fall in the shadow of:

- IMUL
- Any floating-point operate, especially FDIV

Each CALL_PAL instruction includes a function field that will be used in the calculation of the next PC. The PAL OPCDEC flow will be started if the CALL_PAL function field is:

- In the range 40_{16} to $7F_{16}$ inclusive.
- Greater than BF_{16}.
- Between 00_{16} and $3F_{16}$ inclusive, and ICM<04:03> is not equal to kernel.

If no OPCDEC is detected on the CALL_PAL function, then the PC of the instruction to execute after the CALL_PAL is calculated as follows:

- PC<63:14> = PAL_BASE IPR<63:14>
- PC<13> = 1
- PC<12> = CALL_PAL function field<7>
- PC<11:06> = CALL_PAL function field<5:0>
- PC<05:01> = 0
- PC<00> = 1 (PALmode)

6.4 PALcode Entry Points

The minimum number of cycles for a CALL_PAL execution is 4:

Number of Cycles	Description
1	Minimum TRAPB for empty pipe. Typically this will be four cycles.
1	Issue the CALL_PAL instruction.
2	The minimum length of a PAL flow. However, in most cases there will be more than two cycles of work for the CALL_PAL.

6.4.2 PALcode Trap Entry Points

Chip-specific trap entry points start PALcode. (No PALcode assist is required for replay and mispredict type traps.) EXC_ADDR is loaded with the return PC and the Ibox performs a TRAPB in the shadow of the trap. The return prediction stack is pushed with the PC of the trapping instruction for precise traps, and with some later PC for imprecise traps.

Table 6–1 shows the PALcode trap entry points and their offset from the PAL_BASE IPR. Entry points are listed from highest to lowest priority. (Prioritization among the Dstream traps works because DTBMISS is suppressed when there is a sign check error. The priority of ITBMISS and interrupt is reversed if there is an Icache miss.)

Table 6–1 PALcode Trap Entry Points

Entry Name	Offset_{16}	Description
RESET	0000	Reset
IACCVIO	0080	Istream access violation or sign check error on PC
INTERRUPT	0100	Interrupt: hardware, software, and AST
ITBMISS	0180	Istream TBMISS
DTBMISS_SINGLE	0200	Dstream TBMISS
DTBMISS_DOUBLE	0280	Dstream TBMISS during virtual page table entry (PTE) fetch
UNALIGN	0300	Dstream unaligned reference
DFAULT	0380	Dstream fault or sign check error on virtual address

(continued on next page)

Table 6–1 (Cont.) PALcode Trap Entry Points

Entry Name	$Offset_{16}$	Description
MCHK	0400	Uncorrected hardware error
OPCDEC	0480	Illegal opcode
ARITH	0500	Arithmetic exception
FEN	0580	Floating-point operation attempted with: • Floating-point instructions (LD, ST, and operates) disabled through FPE bit in the ICSR IPR • Floating-point IEEE operation with data type other than S, T, or Q

6.5 Required PALcode Function Codes

Table 6–2 lists opcodes required for all Alpha implementations. The notation used is oo.ffff, where oo is the hexadecimal 6-bit opcode and ffff is the hexadecimal 26-bit function code.

Table 6–2 Required PALcode Function Codes

Mnemonic	Type	Function Code
DRAINA	Privileged	00.0002
HALT	Privileged	00.0000
IMB	Unprivileged	00.0086

6.6 Alpha 21164 Implementation of the Architecturally Reserved Opcodes

PALcode uses the Alpha instruction set for most of its operations. Table 6–3 lists the opcodes reserved by the Alpha architecture for implementation-specific use. These opcodes are privileged and are only available in PALmode.

Note

These architecturally reserved opcodes contain different options to the 21064 opcodes of the same names.

6.6 Alpha 21164 Implementation of the Architecturally Reserved Opcodes

Table 6–3 Opcodes Reserved for PALcode

21164 Mnemonic	Opcode	Architecture Mnemonic	Function
HW_LD	1B	PAL1B	Performs Dstream load instructions.
HW_ST	1F	PAL1F	Performs Dstream store instructions.
HW_REI	1E	PAL1E	Returns instruction flow to the program counter (PC) pointed to by EXC_ADDR IPR.
HW_MFPR	19	PAL19	Accesses the Ibox, Mbox, and Dcache internal processor registers (IPRs).
HW_MTPR	1D	PAL1D	Accesses the Ibox, Mbox, and Dcache IPRs.

These instructions produce an OPCDEC exception if executed while not in the PALmode environment. If ICSR<HWE> is set, these instructions can be executed in kernel mode. Any software executing with ICSR<HWE> set must use extreme care to obey all restrictions listed in this chapter and Chapter 5.

Register checking and bypassing logic is provided for PALcode instructions as it is for non-PALcode instructions, when using general purpose registers (GPRs).

Note

Explicit software timing is required for accessing the hardware-specific IPRs and the PAL_TEMP registers. These constraints are described in Table 5–38.

6.6.1 HW_LD Instruction

PALcode uses the HW_LD instruction to access memory outside of the realm of normal Alpha memory management and to do special forms of Dstream loads. Figure 6–1 and Table 6–4 describe the format and fields of the HW_LD instruction. Data alignment traps are inhibited for HW_LD instructions.

6.6 Alpha 21164 Implementation of the Architecturally Reserved Opcodes

Figure 6–1 HW_LD Instruction Format

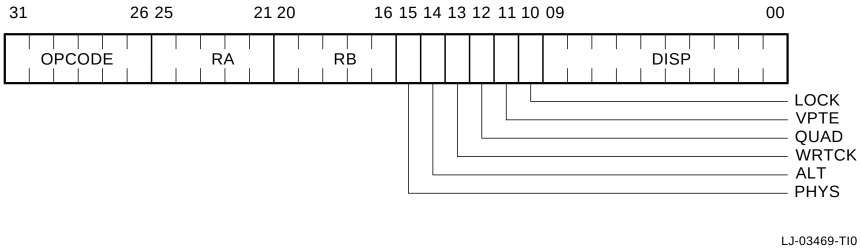


Table 6–4 HW_LD Format Description

Field	Value	Description
OPCODE	$1B_{16}$	The OPCODE field contains $1B_{16}$.
RA		Destination register number.
RB		Base register for memory address.
PHYS	0	The effective address for the HW_LD is virtual.
	1	The effective address for the HW_LD is physical. Translation and memory-management access checks are inhibited.
ALT	0	Memory-management checks use Mbox IPR DTB_CM for access checks.
	1	Memory-management checks use Mbox IPR ALT_MODE for access checks.
WRTCK	0	Memory-management checks fault on read (FOR) and read access violations.
	1	Memory-management checks FOR, fault on write (FOW), read, and write access violations.
QUAD	0	Length is longword.
	1	Length is quadword.
VPTE	1	Flags a virtual PTE fetch. Used by trap logic to distinguish single TBMISS from double TBMISS. Access checks are performed in kernel mode.
LOCK	1	Load lock version of HW_LD. PAL must slot to E0 pipe.
DISP		Holds a 10-bit signed byte displacement.

6.6.2 HW_ST Instruction

PALcode uses the HW_ST instruction to access memory outside of the realm of normal Alpha memory management and to do special forms of Dstream store instructions. Figure 6–2 and Table 6–5 describe the format and fields of the HW_ST instruction. Data alignment traps are inhibited for HW_ST instructions. The Ibox logic will always slot HW_ST to pipe E0.

Figure 6–2 HW_ST Instruction Format

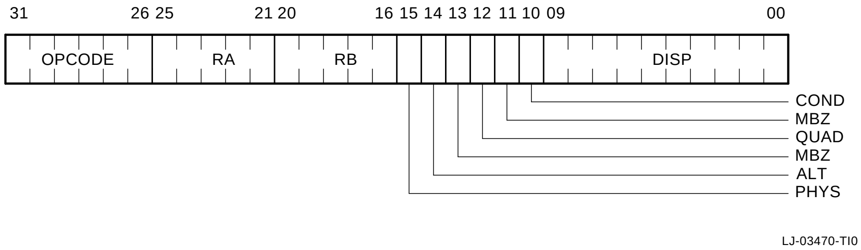


Table 6–5 HW_ST Format Description

Field	Value	Description
OPCODE	$1F_{16}$	The OPCODE field contains $1F_{16}$.
RA		Write data register number.
RB		Base register for memory address.
PHYS	0	The effective address for the HW_ST is virtual.
	1	The effective address for the HW_ST is physical. Translation and memory-management access checks are inhibited.
ALT	0	Memory-management checks use Mbox IPR DTB_CM for access checks.
	1	Memory-management checks use Mbox IPR ALT_MODE for access checks.
QUAD	0	Length is longword.
	1	Length is quadword.
COND	1	Store_conditional version of HW_ST. In this case, RA is written with the value of LOCK_FLAG.
DISP		Holds a 10-bit signed byte displacement.
MBZ		HW_ST<13,11> must be zero.

6.6.3 HW_REI Instruction

The HW_REI instruction is used to return instruction flow to the PC pointed to by the EXC_ADDR IPR. The value in EXC_ADDR<0> will be used as the new value of PALmode after the HW_REI instruction.

The Ibox uses the return prediction stack to speed the execution of HW_REI. There are two different types of HW_REI:

- Prefetch: In this case, the Ibox begins fetching the new Istream as soon as possible. This is the version of HW_REI that is normally used.
- Stall prefetch: This encoding of HW_REI inhibits Istream fetch until the HW_REI itself is issued. Thus, this is the method used to synchronize Ibox changes (such as ITB write instructions) with the HW_REI. There is a rule that PALcode can have only one such HW_REI in an aligned block of four instructions.

Figure 6–3 and Table 6–6 describe the format and fields of the HW_REI instruction. The Ibox logic will slot HW_REI to pipe E1.

Figure 6–3 HW_REI Instruction Format

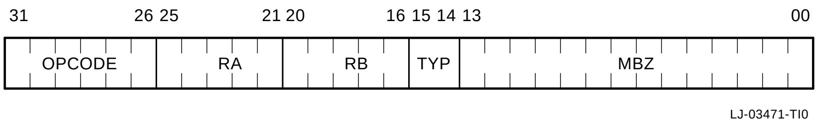


Table 6–6 HW_REI Format Description

Field	Value	Description
OPCODE	$1E_{16}$	The OPCODE field contains $1E_{16}$.
RA/RB		Register numbers, should be R31 to avoid unnecessary stalls.
TYP	10 11	Normal version. Stall version.
MBZ	0	HW_REI<13:00> must be zero.

6.6.4 HW_MFPR and HW_MTPR Instructions

The HW_MFPR and HW_MTPR instructions are used to access internal state from the Ibox, Mbox, and Dcache. The HW_MFPR from Ibox IPRs has a latency of one cycle (HW_MFPR in cycle *n* results in data available to the using instruction in cycle *n*+1). HW_MFPR from Mbox and Dcache IPRs has a latency of two cycles. Ibox hardware slots each type of MXPR to the correct Ebox pipe (refer to Table 5–1).

Figure 6–4 and Table 6–7 describe the format and fields of the HW_MFPR and HW_MTPR instructions.

Figure 6–4 HW_MFPR and HW_MTPR Instruction Format

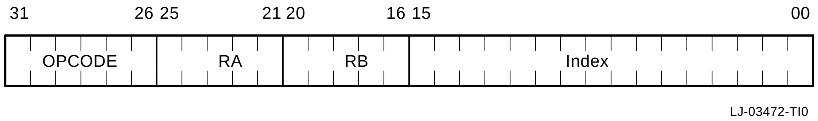


Table 6–7 HW_MTPR and HW_MFPR Format Description

Field	Value	Description
OPCODE	19_{16}	The OPCODE field contains 19_{16} for HW_MFPR.
	$1D_{16}$	The OPCODE field contains $1D_{16}$ for HW_MTPR.
RA/RB		Must be the same, source register for HW_MTPR and destination register for HW_MFPR.
Index		Specifies the IPR. Refer to Table 5–1 for field encoding. Refer to Chapter 5 for more details about specific IPRs.

7
Initialization and Configuration

This chapter provides information on 21164-specific microprocessor/system initialization and configuration. It is organized as follows:

- Input signals **sys_reset_l** and **dc_ok_h** and booting
- Sysclk ratio and delay
- Built-in self-test (BiSt)
- Serial read-only memory (SROM) interface port
- Serial terminal port
- Cache initialization
- External interface initialization
- Internal processor register (IPR) reset state
- Timeout reset
- IEEE 1149.1 test port reset

7.1 Input Signals sys_reset_l and dc_ok_h and Booting

The 21164 reset sequence uses two input signals: **sys_reset_l** and **dc_ok_h**. When transitioning from a powered-down state to a powered-up state, signal **dc_ok_h** must be deasserted, and signal **sys_reset_l** must be asserted until power has reached the proper operating point and the input clock to the 21164 is stable. If the input clock is derived from a PLL it may take many milliseconds for the input oscillator to start and the PLL output to stabilize.

After power has reached the proper operating point, signal **dc_ok_h** must be asserted. Then, signal **sys_reset_l** must be deasserted. At this point, the 21164 recognizes a powered on state. If signal **dc_ok_h** is not asserted, signal **sys_reset_l** is forced asserted internally. After **sys_reset_l** is deasserted, the 21164 begins the following sequence of operations:

1. Icache built-in self-test (BiSt)

7.1 Input Signals sys_reset_l and dc_ok_h and Booting

2. An optional automatic Icache initialization, using an external serial ROM (SROM) interface
3. Dispatch to the reset PALcode trap entry point (physical location 0)
   a. If step 2 initialized the Icache by using the SROM interface, the cache should contain code that appears to be at location 0, that is, the cache should be initialized such that it hits on the dispatch. Typically the code in the Icache should configure the 21164's IPRs as necessary before causing any offchip read or write commands. This allows the 21164 to be configured to match the external system implementation.
   b. If step 2 did not initialize the Icache, the Icache has been flushed by reset. The reset PALcode trap dispatch misses in the Icache and Scache (also flushed by reset) and produces an offchip read command. The external system implementation must be compatible with the 21164's default configuration after reset (refer to Section 7.8). The code that is executed at this point should complete the 21164 configuration as necessary.
4. After configuring the 21164, control can be transferred to code anywhere in memory, including the noncacheable regions. If the SROM interface was used to initialize the Icache, the Icache can be flushed by a write operation to IC_FLUSH_CTL after control is transferred. This transfer of control should be to addresses not loaded in the Icache by the SROM interface or the Icache may provide unexpected instructions.
5. Typically, PALbase and any state required by PALcode are initialized and the console is started (switching out of PALmode and into native mode). The console code initializes and configures the system and boots an operating system from an I/O device such as a disk or the network.

Signal **sys_reset_l** forces the CPU into a known state. Signal **sys_reset_l** must remain asserted while signal **dc_ok_h** is deasserted, and for some period of time after **dc_ok_h** assertion. It should remain asserted for at least 400 internal CPU cycles in length. Then, signal **sys_reset_l** may be deasserted. Signal **sys_reset_l** deassertion need not be synchronous with respect to sysclk. Section 7.8 lists the reset state of each IPR. Table 7–1 provides the reset state of each external signal pin.

Table 7–1 Alpha 21164 Signal Pin Reset State

Signal	Reset State
Clocks	
clk_mode_h<1:0>	NA (input).
cpu_clk_out_h	Clock output.
dc_ok_h	NA (input).
osc_clk_in_h,l	Must be clocking.
ref_clk_in_h	NA (input).
sys_clk_out1_h,l	Clock output.
sys_clk_out2_h,l	Clock output.
sys_reset_l	NA (input).
Bcache	
data_h<127:0>	Tristated.
data_check_h<15:0>	Tristated.
data_ram_oe_h	Deasserted.
data_ram_we_h	Deasserted.
index_h<25:4>	Unspecified.
tag_ctl_par_h	Tristated.
tag_data_h<38:20>	Tristated.
tag_data_par_h	Tristated.
tag_dirty_h	Tristated.
tag_ram_oe_h	Deasserted.
tag_ram_we_h	Deasserted.
tag_shared_h	Tristated.
tag_valid_h	Tristated.

(continued on next page)

Table 7–1 (Cont.) Alpha 21164 Signal Pin Reset State

Signal	Reset State
System Interface	
addr_h<39:4>	Driven or tristated depending upon **addr_bus_req_h** at most recent sysclk edge. If driven, the value is unspecified.
addr_bus_req_h	NA (input).
addr_cmd_par_h	Driven or tristated depending upon **addr_bus_req_h** at most recent sysclk edge. If driven, the command is NOP.
addr_res_h<2:0>	NOP.
cack_h	Must be deasserted.
cfail_h	Must be deasserted.
cmd_h<3:0>	Driven or tristated depending upon **addr_bus_req_h** at most recent sysclk edge. If driven, the command is NOP.
dack_h	Must be deasserted.
data_bus_req_h	NA (input).
fill_h	Must be deasserted.
fill_error_h	Must be deasserted.
fill_id_h	Must be deasserted.
fill_nocheck_h	Must be deasserted.
idle_bc_h	Must be deasserted.
int4_valid_h<3:0>	Unspecified.
scache_set_h<1:0>	Unspecified.
shared_h	NA (input).
system_lock_flag_h	Must be deasserted.
victim_pending_h	Unspecified.
Interrupts	
irq_h<3:0>	Sysclk divisor ratio input.
mch_hlt_irq_h	Sysclk delay input.
pwr_fail_irq_h	Sysclk delay input.
sys_mch_chk_irq_h	Sysclk delay input.

(continued on next page)

Table 7–1 (Cont.) Alpha 21164 Signal Pin Reset State

Signal	Reset State
Test Modes	
port_mode_h<1:0>	NA (input).
srom_clk_h	Deasserted.
srom_data_h	NA (input).
srom_oe_l	Deasserted.
srom_present_l	NA (input).
tck_h	NA (input).
tdi_h	NA (input).
tdo_h	NA (input).
temp_sense	NA (input).
test_status_h<1:0>	Deasserted.
tms_h	NA (input).
trst_l	Must be asserted (input).
Miscellaneous	
perf_mon_h	NA (input).
spare_io	NA.

While signal **dc_ok_h** is deasserted, the 21164 provides its own internal clock source from an onchip ring oscillator. When **dc_ok_h** is asserted, the 21164 clock source is the differential clock input pins **osc_clk_in_h,l**.

When the 21164 is free-running from the internal ring oscillator, the internal clock frequency is in the range of 10 MHz to 100 MHz (varies from chip to chip). The sysclk divisor and **sys_clk_out2_x** delay are determined by input pins while signal **sys_reset_l** remains asserted. Refer to Section 4.2.2 and Section 4.2.3 for ratio and delay values.

7.1.1 Pin State with dc_ok_h Not Asserted

While **dc_ok_h** is deasserted, and **sys_reset_l** is asserted, every output and bidirectional 21164 pin is tristated and pulled weakly to ground by a small pull-down transistor.

7.2 Sysclk Ratio and Delay

While in reset, the 21164 reads sysclk configuration parameters from the interrupt signal pins. These inputs should be driven with the correct configuration values whenever signal **sys_reset_l** is asserted. Refer to Section 4.2.2 and Section 4.2.3 for relevant input signals and ratio/delay values.

If the signal inputs reflecting configuration parameters change while **sys_reset_l** is asserted, allow 20 internal CPU cycles before the new sysclk behavior is correct.

7.3 Built-In Self-Test (BiSt)

Upon deassertion of signal **sys_reset_l**, the 21164 automatically executes the Icache built-in self-test (BiSt). The Icache is automatically tested and the result is made available in the ICSR IPR and on signal **test_status_h<0>**. Internally, the CPU reset continues to be asserted throughout the BiSt process. For additional information, refer to Section 9.4.6.

7.4 Serial Read-Only Memory Interface Port

The serial read-only memory (SROM) interface provides the initialization data load path from a system SROM to the instruction cache (Icache). Following initialization, this interface can function as a diagnostic port using privileged architecture library code (PALcode).

The following signals make up the SROM interface:

srom_present_l
srom_data_h
srom_oe_l
srom_clk_h

During system reset, the 21164 samples the **srom_present_l** signal for the presence of SROM. If **srom_present_l** is deasserted, the SROM load is disabled and the reset sequence clears the Icache valid bits. This causes the first instruction fetch to miss the Icache and read instructions from offchip memory.

If **srom_present_l** is asserted during setup, then the system performs an SROM load as follows:

1. The **srom_oe_l** signal supplies the output enable to the SROM.

2. The **srom_clk_h** signal supplies the clock to the ROM that causes it to advance to the next bit. The cycle time of this clock is 126 ± times the CPU clock period.
3. The **srom_data_h** signal inputs the SROM data.

Every data and tag bit in the Icache is loaded by this sequence.

7.4.1 Serial Instruction Cache Load Operation

All Icache bits, including each block's tag, address space number (ASN), address space match (ASM), valid and branch history bits can be loaded serially from offchip serial ROMs. Once the serial load has been invoked by the chip reset sequence, the entire cache is loaded automatically from the lowest to the highest addresses.

The automatic serial Icache fill invoked by the chip reset sequence operates internally at a frequency of 126*CPU clock period. However, due to the synchronization with the system clocks, consecutive access cycles to SROM may shrink or stretch by a system cycle. For example, for a system with a system clock ratio of 15, the time between the two consecutive SROM accesses may be anywhere in the range 111 to 141 CPU cycles. The SROM used in the system must be able to support access times in this range. Refer to Section 9.4.5 for additional SROM timing information.

The serial bits are received in a 200-bit-long fill scan path, from which they are written in parallel into the Icache address. The fill scan path is organized as shown in the text following this paragraph. The farthest bit (<42>) is shifted in first and the nearest bit (BHT<0>) is shifted in last. The data and predecode bits in the data array are interleaved.

```
srom_data_h          serial input ->
BHT Array              0 ->   1 ->  ... ->   7 ->
Data                 127 ->  95 ->  126 ->  94 -> ...  ->  96 ->  64 ->
Predecodes            19 ->  14 ->   18 ->  13 -> ...  ->  15 ->  10 ->
Data parity            1 ->   0 ->
Predecodes             9 ->   4 ->    8 ->   3 -> ...  ->   5 ->   0 ->
Data                  63 ->  31 ->   62 ->  30 -> ...  ->  32 ->   0 ->
Tag Parity             b ->
Tag Valids             0 ->   1 ->
TAG Phy.Address        b ->
TAG ASN                0 ->   1 ->  ... ->   6 ->
TAG ASM                b ->
TAGs                  13 ->  14 ->  ... ->  42

  b = Single bit signal
```

Refer to Appendix C for example C code that calculates the predecode values of a serial Icache load.

7.5 Serial Terminal Port

After the SROM data is loaded into the Icache, the three SROM interface signals can be used as a software ‘‘UART’’ and the pins become parallel I/O pins that can drive a diagnostic terminal by using an interface such as RS232 or RS423.

7.6 Cache Initialization

Regardless of whether the Icache BiSt is executed, the Icache is flushed during the reset sequence prior to the SROM load. If the SROM load is bypassed, the Icache will be in the flushed state initially.

The second-level cache (Scache) is flushed and enabled by internal reset. This is required if the SROM load is bypassed. The initial Istream reference after reset is location 0. Because that is a cacheable-space reference, the Scache will be probed.

The data cache (Dcache) is disabled by reset. It is not initialized or flushed by reset. It should be initialized by PALcode before being enabled.

The external board-level Bcache is disabled by reset. It should be initialized by PALcode before being enabled.

7.6.1 Icache Initialization

The Icache is not kept coherent with memory. When it is necessary to make it coherent with memory, the following procedure is used. The CALL_PAL IMB function performs this function by using this procedure.

1. Execute an MB instruction. This forces all write data in the write buffer into memory.
   - Stall until write buffer is drained.
   - Carry load or issue a HW_MFPR from any Mbox IPR.
2. Write to IC_FLUSH_CTL with an HW_MTPR to flush the Icache.
3. Execute a total of 44 NOP instructions (BIS r31,r31,r31) to clear the prefetch buffers and Ibox pipeline. The 44 NOP instructions must start on an INT16 boundary. Pad with additional NOP instructions if necessary.

7.6.2 Flushing Dirty Blocks

During a power failure recovery, dirty blocks must be flushed out of the Scache and backup cache (Bcache), if present.

Systems Without a Bcache

To flush out dirty blocks from the Scache on power failure, the following sequence must be used to guarantee that all the dirty blocks have been written back to main memory. The BC_CONFIG<BC_SIZE> field is used for this function in systems without a Bcache. When powering up, this field is initialized to a value representing a 1M-byte Bcache. During system configuration flow, this field must be changed to a value of 0 for normal operation.

To flush out the dirty blocks from all three sets in the Scache, perform the following tasks:

1. Set BC_CONFIG<BC_SIZE><2:0> = 0x1; do loads at a stride of 64 bytes through 128K bytes of continuous memory; guarantees all dirty blocks from set0 are flushed out.
2. Set BC_CONFIG<BC_SIZE><2:0> = 0x2; do loads at a stride of 64 bytes through 96K bytes of continuous memory; guarantees all dirty blocks from set1 are flushed out.
3. Set BC_CONFIG<BC_SIZE><2:0> = 0x4; do loads at a stride of 64 bytes through 64K bytes of continuous memory; guarantees all dirty blocks from set2 are flushed out.

All other values of BC_CONFIG<BC_SIZE><2:0> are undefined in this mode.

Systems with a Bcache

To flush out dirty blocks from the Scache and Bcache on power failure, the following sequence must be used to guarantee that all the dirty blocks have been written back to main memory:

$$\text{perform loads at a stride of Bcache block size} = 2\times \text{ size of the Bcache}$$

7.7 External Interface Initialization

After reset, the cache control and bus interface unit (Cbox) is in the default configuration dictated by the reset state of the IPR bits that select the configuration options. The Cbox response to system commands and internally generated memory accesses is determined by this default configuration. System environments that are not compatible with the default configuration must use the SROM Icache load feature to initially load and execute a PALcode program. This program configures the external interface control (Cbox) IPRs as needed.

7.8 Internal Processor Register Reset State

Many IPR bits are not initialized by reset. They are located in error-reporting registers and other IPR states. They must be initialized by initialization PALcode. Table 7–2 lists the state of all internal processor registers (IPRs) immediately following reset. The table also specifies which registers need to be initialized by power-up PALcode.

Table 7–2 Internal Processor Register Reset State

IPR	Reset State	Comments
Ibox Registers		
ITB_TAG	UNDEFINED	
ITB_PTE	UNDEFINED	
ITB_ASN	UNDEFINED	PALcode must initialize.
ITB_PTE_TEMP	UNDEFINED	
ITB_IAP	UNDEFINED	
ITB_IA	UNDEFINED	PALcode must initialize.
ITB_IS	UNDEFINED	
IFAULT_VA_FORM	UNDEFINED	
IVPTBR	UNDEFINED	PALcode must initialize.
ICPERR_STAT	UNDEFINED	PALcode must initialize.
IC_FLUSH_CTL	UNDEFINED	
EXC_ADDR	UNDEFINED	

(continued on next page)

Table 7–2 (Cont.) Internal Processor Register Reset State

IPR	Reset State	Comments
EXC_SUM	UNDEFINED	PALcode must clear exception summary and exception register write mask by writing EXC_SUM.
EXC_MASK	UNDEFINED	
PAL_BASE	Cleared	Cleared on reset.
ICM	UNDEFINED	PALcode must set current mode.
ICSR	See Comments	All bits are cleared on reset except ICSR<37>, which is set, and ICSR<38>, which is UNDEFINED.
IPLR	UNDEFINED	PALcode must initialize.
INTID	UNDEFINED	
ASTRR	UNDEFINED	PALcode must initialize.
ASTER	UNDEFINED	PALcode must initialize.
SIRR	UNDEFINED	PALcode must initialize.
HWINT_CLR	UNDEFINED	PALcode must initialize.
ISR	UNDEFINED	
SL_XMIT	Cleared	Appears on external pin.
SL_RCV	UNDEFINED	
PMCTR	See Comments	PMCTR<15:10> are cleared on reset. All other bits are UNDEFINED.
Mbox Registers		
DTB_ASN	UNDEFINED	PALcode must initialize.
DTB_CM	UNDEFINED	PALcode must initialize.
DTB_TAG	Cleared	Valid bits are cleared on chip reset but not on timeout reset.
DTB_PTE	UNDEFINED	
DTB_PTE_TEMP	UNDEFINED	
MM_STAT	UNDEFINED	Must be unlocked by PALcode by reading VA register.

(continued on next page)

7.8 Internal Processor Register Reset State

Table 7–2 (Cont.) Internal Processor Register Reset State

IPR	Reset State	Comments
VA	UNDEFINED	Must be unlocked by PALcode by reading VA register.
VA_FORM	UNDEFINED	Must be unlocked by PALcode by reading VA register.
MVPTBR	UNDEFINED	PALcode must initialize.
DC_PERR_STAT	UNDEFINED	PALcode must initialize.
DTB_IAP	UNDEFINED	
DTB_IA	UNDEFINED	
DTB_IS	UNDEFINED	
MCSR	Cleared	Cleared on chip reset but not on timeout reset.
DC_MODE	Cleared	Cleared on chip reset but not on timeout reset.
MAF_MODE	Cleared	Cleared on chip reset. MAF_MODE<05> cleared on timeout reset.
DC_FLUSH	UNDEFINED	PALcode must write this register to clear Dcache valid bits.
ALT_MODE	UNDEFINED	
CC	UNDEFINED	CC is disabled on chip reset.
CC_CTL	UNDEFINED	
DC_TEST_CTL	UNDEFINED	
DC_TEST_TAG	UNDEFINED	
DC_TEST_TAG_TEMP	UNDEFINED	
Cbox Registers		
SC_CTL	See Comments	SC_CTL<11:00> cleared on reset. SC_CTL<12> is set at power-up.
SC_STAT	UNDEFINED	PALcode must read to unlock.
SC_ADDR	UNDEFINED	

(continued on next page)

Table 7–2 (Cont.) Internal Processor Register Reset State

IPR	Reset State	Comments
BC_CONTROL	See Comments	BC_CONTROL<01:00>, <07>, <14:13>, <16>, and <27:19> cleared. BC_CONTROL<06:04> and <15> set on reset but not timeout reset. All other bits are UNDEFINED and must be initialized by PALcode.
BC_CONFIG	See Comments	At power-up, BC_CONFIG is initialized to a value of 0000 0000 0001 7441_{16}.
BC_TAG_ADDR	UNDEFINED	
EI_STAT	UNDEFINED	PALcode must read twice to unlock.
EI_ADDR	UNDEFINED	
FILL_SYN	UNDEFINED	

Note

The Bcache parameters BC_SIZE (size), BC_RD_SPD (read speed), BC_WR_SPD (write speed), and BC_WE_CTL (write-enable control) are all configured to default values on reset and must be initialized in the BC_CONFIG register before enabling the Bcache.

7.9 Timeout Reset

The instruction fetch/decode unit and branch unit (Ibox) contains a timer that times out when a very long period of time passes with no instruction completing. When this timeout occurs, an internal reset event occurs. This clears sufficient internal state to allow the CPU to begin executing again. Registers, IPRs (except as noted in Table 7–2), and caches are not affected. Dispatch to the PALcode MCHK trap entry point occurs immediately.

7.10 IEEE 1149.1 Test Port Reset

Signal **trst_l** must be asserted when **sys_reset_l** is asserted or when **dc_ok_h** is deasserted. Continuous **trst_l** assertion during normal operation is used to guarantee that the IEEE 1149.1 test port does not affect 21164 operation.

8 Error Detection and Error Handling

This chapter provides an overview of the 21164's error handling strategy. Each internal cache (instruction cache [Icache], data cache [Dcache], and second-level cache [Scache]) implements parity protection for tag and data. Error correction code (ECC) protection is implemented for memory and backup cache (Bcache) data. (The implementation provides detection of all double-bit errors and correction of all single-bit errors.) Correctable instruction stream (Istream) and data stream (Dstream) ECC errors are corrected in hardware without privileged architecture library code (PALcode) intervention. Bcache tags are parity protected. The instruction fetch/decode unit and branch unit (Ibox) implements logic that detects when no progress has been made for a very long time and forces a machine check trap.

PALcode handles all error traps (machine checks and correctable error interrupts). Where possible, the address of affected data is latched in an IPR. Most of the Istream errors can be retried by the operating system because the machine check occurs before any part of the instruction causing the error is executed. In some other cases, the system may be able to recover from an error by terminating all processes that had access to the affected memory location.

8.1 Error Flows

The following flows describe the events that take place during an error, the recommended responses necessary to determine the source of the error, and the suggested actions to resolve them.

8.1.1 Icache Data or Tag Parity Error

- Machine check occurs before the instruction causing the parity error is executed.
- EXC_ADDR contains either the PC of the instruction that caused the parity error or that of an earlier trapping instruction.
- ICPERR_STAT<TPE> or <DPE> is set.

- Can be retried.

Note

The Icache is not flushed by hardware in this event. If an Icache parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

- Recommendation: Flush the Icache early in the MCHK routine.

8.1.2 Scache Data Parity Error—Istream

- Machine check occurs before the instruction causing the parity error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Can be retried if there are no multiple errors.
- Recommendation: Flush the Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- SC_STAT: SC_DPERR<7:0> is set; <SC_SCND_ERR> is set if there are multiple errors.
- SC_STAT: CBOX_CMD is IRD.
- SC_ADDR: Contains the address of the 32-byte block containing the error. (Bit 4 indicates which octaword was accessed first, but the error may be in either octaword.)

Note

If the Istream parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

- Recommendation: On data parity errors, it may be feasible for the operating system to ''flush'' the block of data out of the Scache by requesting a block of data with the same Bcache index, but a different tag. This may not be feasible on tag parity errors, because the tag address is suspect. If the requested block is loaded with no problems, then the ''bad data'' has been replaced. If the ''bad data'' is marked dirty, then when the new data tries to replace the old data, another parity error may result during the write-back (this is a reason not to attempt this in PALcode, because a MCHK from PALcode is always fatal).

8.1.3 Scache Tag Parity Error—Istream

- Machine check occurs before the instruction causing the parity error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Cannot be retried. Probably will not be able to recover by deleting a single process because the exact address is unknown.
- Recommendation: Flush the Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- SC_STAT: SC_TPERR<2:0> is set; <SC_SCND_ERR> is set if there are multiple errors.
- SC_STAT: CBOX_CMD is IRD.
- SC_ADDR: Contains the address of the 32-byte block containing the error. (Bit 4 indicates which octaword was accessed first, but the error may be in either octaword.)

Note

If the Istream parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

8.1.4 Scache Data Parity Error—Dstream Read/Write, READ_DIRTY

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred.
- SC_STAT: SC_DPERR<7:0> is set; SC_SCND_ERR is set if there are multiple errors.
- SC_STAT: CBOX_CMD is DRD, DWRITE, or READ_DIRTY.
- SC_ADDR: Contains the address of the 32-byte block containing the error. (Bit 4 indicates which octaword was accessed first, but the error may be in either octaword.)

8.1.5 Scache Tag Parity Error—Dstream or System Commands

- Machine check occurs. Machine state may have changed.
- Cannot be retried. Probably will not be able to recover by deleting a single process because the exact address is unknown.
- SC_STAT: SC_TPERR<7:0> is set; <SC_SCND_ERR> is set if there are multiple errors.
- SC_STAT: CBOX_CMD is DRD, DWRITE, READ_DIRTY, SET_SHARED, or INVAL.
- SC_ADDR: records physical address bits <39:04> of location with error.

8.1.6 Dcache Data Parity Error

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred.
- DCPERR_STAT: <DP0> or <DP1> is set. <LOCK> is set. <SEO> is set if there are multiple errors.

Note

For multiple parity errors in the same cycle, the <SEO> bit is not set, but more than one error bit will be set.

- VA: Contains the virtual address of the quadword with the error.
- MM_STAT locked. Contents contain information about instruction causing parity error.

Note

Fault information on another instruction in same cycle may be lost.

8.1.7 Dcache Tag Parity Error

- Machine check occurs. Machine state may have changed.
- DCPERR_STAT: <TP0> or <TP1> is set. <LOCK> is set. <SEO> is set if there are multiple errors.

Note

For multiple parity errors in the same cycle, the <SEO> bit is not set, but more than one error bit will be set.

- VA: Contains the virtual address of the Dcache block (hexword) with the error.
- MM_STAT locked. Contents contain information about instruction causing parity error. <WR> bit is set if error occurred on a store instruction.

Note

Fault information on another instruction in the same cycle may be lost.

- Probably will not be able to recover by deleting a single process, because exact address is unknown, and a load may have falsely hit.

8.1.8 Istream Uncorrectable ECC or Data Parity Errors (Bcache or Memory)

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Can be retried if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- EI_STAT: <UNC_ECC_ERR> is set; <SEO_HRD_ERR> is set if there are multiple errors.
- EI_STAT: <EI_ES> is set if source of fill data is memory/system, clear if Bcache.
- EI_STAT: <FIL_IRD> is set.

- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- FILL_SYN: Contains syndrome bits associated with the failing octaword. This register contains byte parity error status if in parity mode.
- BC_TAG_ADDR: Holds results of external cache tag probe if external cache was enabled for this transaction.

Note

If the Istream ECC or parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

- Recommendation: On data ECC/parity errors, it may be feasible for the operating system to ‘‘flush’’ the block of data out of the Bcache by requesting a block of data with the same Bcache index, but a different tag. If the requested block is loaded with no problems, then the ‘‘bad data’’ has been replaced. If the ‘‘bad data’’ is marked dirty, then when the new data tries to replace the old data, another ECC/parity error may result during the write-back (this is a reason not to attempt this in PALcode, because a MCHK from PALcode is always fatal).

8.1.9 Dstream Uncorrectable ECC or Data Parity Errors (Bcache or Memory)

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred.
- EI_STAT: <UNC_ECC_ERR> is set; <SEO_HRD_ERR> is set if there are multiple errors.
- EI_STAT: <EI_ES> is set if source of fill data is memory/system, is clear if Bcache.
- EI_STAT: <FIL_IRD> is clear.
- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- FILL_SYN: Contains syndrome bits associated with the failing octaword. This register contains byte parity error status if in parity mode.

- BC_TAG_ADDR: Holds results of external cache tag probe if external cache was enabled for this transaction.

8.1.10 Bcache Tag Parity Errors—Istream

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Can be retried if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- EI_STAT: <BC_TPERR> or <BC_TC_PERR> is set; <SEO_HRD_ERR> is set if there are multiple errors.
- EI_STAT: <EI_ES> is clear.
- EI_STAT: <FIL_IRD> is set.
- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- BC_TAG_ADDR: Holds results of external cache tag probe.

Note

The Bcache hit is determined based on the tag alone, not the parity bit. The victim is processed according to the status bits in the tag, ignoring the control field parity. PALcode can distinguish fatal from nonfatal occurrences by checking for the case in which a potentially dirty block is replaced without the victim being properly written back and the case of false hit when the tag parity is incorrect.

8.1.11 Bcache Tag Parity Errors—Dstream

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred. Bcache hit is determined based on the tag alone, not the parity bit. The victim is processed according to the status bits in the tag, ignoring the control field parity. PALcode can distinguish fatal from nonfatal occurrences by checking for the case in which a potentially dirty block is replaced without

the victim being properly written back and the case of false hit when the tag parity is incorrect.

- EI_STAT: <BC_TPERR> or <BC_TC_PERR> is set; <SEO_HRD_ERR> is set if there are multiple errors.
- EI_STAT: <EI_ES> is clear.
- EI_STAT: <FIL_IRD> is clear.
- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- BC_TAG_ADDR: Holds results of external cache tag probe.

8.1.12 System Command/Address Parity Error

- Machine check occurs. Machine state may have changed.
- EI_STAT: <EI_PAR_ERR> is set; <SEO_HRD_ERR> is set if there are multiple errors.
- EI_STAT: <EI_ES> is set.
- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- BC_TAG_ADDR: Holds results of external cache tag probe if external cache was enabled for this transaction.
- When the 21164 detects a command or address parity error, the command is unconditionally NOACKed.

Note

For a sysclk-to-CPU clock ratio of 3, if the 21164 detects a system command/address parity error on a NOP, and immediately receives a valid command from the system, then the 21164 may not acknowledge the command. The 21164 does take the machine check.

8.1.13 System Read Operations of the Bcache

The 21164 does not check the ECC on outgoing Bcache data. If it is bad, the receiving processor will detect it.

8.1.14 Istream or Dstream Correctable ECC Error (Bcache or Memory)

- The 21164 hardware corrects the data before filling the Scache and Icache. The Dcache is completely invalidated. The data in the Bcache contains the ECC error, but is scrubbed by PALcode in the correctable error interrupt routine. (Using LD*x*L or ST*x*C, if the ST*x*C fails, the location can be assumed to be scrubbed.)
- A separately maskable correctable error interrupt occurs at IPL 31 (same as machine check). (Masked by clearing ICSR<CRDE>.)
- ISR: <CRD> is set.
- EI_STAT: <COR_ECC_ERR> is set.
- EI_STAT: <FIL_IRD> is set if Istream; is clear if Dstream.
- EI_STAT: <EI_ES> is clear if source of error is Bcache, is set otherwise.
- EI_ADDR: Contains the physical address bits <39:04> of the octaword associated with the error.
- FILL_SYN: Contains syndrome bits associated with the octaword containing the ECC error.
- BC_TAG_ADDR: Unpredictable (not loaded on correctable errors).

Note

There will be performance degradation in systems when extremely high rates of correctable ECC errors are present due to the internal handling of this error (the implementation utilizes a replay trap and automatic Dcache flush to prevent use of the incorrect data).

8.1.15 Fill Timeout (FILL_ERROR_H)

- For systems in which fill timeout can occur, the system environment should detect fill timeout and cleanly terminate the reference to 21164. If the system environment expects fill timeout to occur, it should detect them. If it does not expect them (as might be true in small systems with fixed memory access timing), it is likely that the internal Ibox timeout will eventually detect a stall if a fill fails to occur. To properly terminate a fill in an error case, the **fill_error_h** pin is asserted for one cycle and the normal fill sequence involving the **fill_h**, **fill_id_h**, and **dack_h** pins is generated by the system environment.

- A **fill_error_h** assertion forces a PALcode trap to the MCHK entry point, but has no other effect.

Note

No internal status is saved to show that this happened. If necessary, systems must save this status, and include read operations of the appropriate status registers in the MCHK PALcode.

8.1.16 System Machine Check

- The 21164 has a maskable machine check interrupt input pin. It is used by system environments to signal fatal errors that are not directly connected to a read access from the 21164. It is masked at IPL 31 and anytime the 21164 is in PALmode.
- ISR: <MCK> is set.

8.1.17 Ibox Timeout

- When the Ibox detects a timeout, it causes a PALcode trap to the MCHK entry point.
- Simultaneously, a partial internal reset occurs: most states except IPR state is reset. This should not be depended on by systems in which fill timeouts occur in typical use (such as, operating system or console code probing locations to determine if certain hardware is present). The purpose of this error detection mechanism is to attempt to prevent system hang in order to write a machine check stack frame.
- ICPERR_STAT: <TMR> is set.

8.1.18 cfail_h and Not cack_h

- Assertion of **cfail_h** in a sysclk cycle in which **cack_h** is not asserted causes the 21164 to immediately execute a partial internal reset.
- PALcode trap to the MCHK entry point.
- Simultaneously, a partial internal reset occurs: most states except IPR state is reset.
- ICPERR_STAT: <TMR> is set.

- This can be used to restore 21164 and the external environment to a consistent state after the external environment detects a command or address parity error.

Note

There is no internal status saved to differentiate the **cfail_h**/no **cack_h** case from the Ibox timeout reset case. If necessary, systems must save this status, and include read operations of the appropriate status registers in the MCHK PALcode.

8.2 MCHK Flow

The following flow is the recommended IPR access order to determine the source of a machine check.

- Must flush Icache to remove bad data on Istream errors. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- Read EXC_ADDR.
- If EXC_ADDR=PAL, then halt.
- Issue MB to clear out Mbox/Cbox before reading Cbox registers or issuing DC_FLUSH.
- Flush Dcache to remove bad data on Dstream errors.
- Read ICSR.
- Read ICPERR_STAT.
- Read DCPERR_STAT.
- Read SC_ADDR.
- Use register dependencies or MB to ensure read operation of SC_ADDR finishes before subsequent read operation of SC_STAT.
- Read SC_STAT (unlocks SC_ADDR).
- Read EI_ADDR, BC_TAG_ADDR, and FILL_SYN.
- Use register dependencies or MB to ensure read operations of EI_ADDR, BC_TAG_ADDR, and FILL_SYN finish before subsequent read operation of EI_STAT.
- Read EI_STAT and save (unlocks EI_ADDR, BC_TAG_ADDR, FILL_SYN).

8.2 MCHK Flow

- Read EI_STAT again to be sure it is unlocked, discard result.
- Check for cases that cannot be retried. If any one of the following are true, then skip retry:
  - EI_STAT<TPERR>
  - EI_STAT<TC_PERR>
  - EI_STAT<EI_PAR_ERR>
  - EI_STAT<SEO_HRD_ERR>
  - EI_STAT<UNC_ECC_ERR> and not EI_STAT<FIL_IRD>
  - DCPERR_STAT<LOCK>
  - SC_STAT<SC_SCND_ERR>
  - SC_STAT<SC_TPERR>
  - Not (SC_STAT<CMD> = IRD) and SC_STAT<SC_DPERR>
  - ICPERR_STAT<TMR>
  - ISR<MCK>
- If none of the previous conditions are true, then there is either an IRD that can be retried or the source of the MCHK is a **fill_error_h**. Add code for query of system status.
- The case can be retried if any one or several of the following are true (and none of the previous conditions were true):
  - EI_STAT<UNC_ECC_ERR> and EI_STAT<FIL_IRD>
  - SC_STAT<SC_DPERR> and (SC_STAT<CMD> = IRD)
  - ICPERR_STAT<TPE>
  - ICPERR_STAT<DPE>
- Unlock the following IPRs:
  - ICPERR_STAT (write 0x1800)
  - DCPERR_STAT (write 0x03)
  - VA, SC_STAT, and EI_STAT are already unlocked.
- Check for arithmetic exceptions:
  - Read EXC_SUM.

  - Check for arithmetic errors and handle according to operating-system-specific requirements.
  - Clear EXC_SUM (unlocks EXC_MASK).
- Report the processor-uncorrectable MCHK according to operating-system-specific requirements.

8.3 Processor-Correctable Error Interrupt Flow (IPL 31)

The following flow is the recommended way to report correctable errors:

- Arrived here through interrupt routine because ISR<CRD> bit set.
- Read EI_ADDR and FILL_SYN.
- Use register dependencies or MB to ensure read operations of EI_ADDR and FILL_SYN finish before subsequent read operation of EI_STAT.
- Read EI_STAT. (Unlocks EI_STAT, EI_ADDR, and FILL_SYN.)
- Scrub the memory location by using LDQ_L/STQ_C to one of the quadwords in each octaword of the Bcache block whose address is reported in EI_ADDR. No need to scrub I/O space addresses as these are noncacheable.
- ACK the CRD Interrupt by writing a ‘‘0’’ to HWINT_CLR<CRDC>.
- No need to unlock any registers because conditions that would cause a lock would also cause a MCHK. VA will not be locked because DTB_MISS and FAULT PALcode routines will not ever be interrupted.
- Report the processor-correctable MCHK according to operating-system-specific requirements.

Note

Only read EI_STAT once in the CRD flow, and then only if ISR<CRD> is set. If an uncorrectable error were to occur just after a second read operation from EI_STAT was issued, then there could be a race between the unlocking of the register and the loading of the new error status, potentially resulting in the loss of the error status.

8.4 MCK_INTERRUPT Flow

- Arrived here through interrupt routine because ISR<MCK> bit set.
- Report the system-uncorrectable MCHK according to operating-system-specific requirements.

8.5 System-Correctable Error Interrupt Flow (IPL 20)

The system-correctable error interrupt is system specific.

9
Electrical Data

This chapter describes the electrical characteristics of the 21164 component and its interface pins. It is organized as follows:

- Electrical characteristics
- dc characteristics
- Clocking scheme
- ac characteristics
- Power supply considerations

9.1 Electrical Characteristics

Table 9–1 lists the maximum ratings for the 21164.

Table 9–1 Alpha 21164 Absolute Maximum Ratings

Characteristics	Ratings
Storage temperature	–55°C to 125°C (–67°F to 257°F)
Junction temperature	15°C to 90°C (59°F to 194°F)
Supply voltage	**Vss** –0.5 V, **Vdd** 3.6 V
Input or output applied[1]	–0.5 V to 6.3 V
Typical worst case power @**Vdd** = 3.3 V Frequency = 266 MHz Frequency = 300 MHz	 46 W 51 W

[1]Refer to Section 9.5.2.

9.1 Electrical Characteristics

Caution

Stress beyond the absolute maximum rating can cause permanent damage to the 21164. Exposure to absolute maximum rating conditions for extended periods of time can affect the 21164 reliability.

9.2 dc Characteristics

The 21164 is designed to run in a CMOS/TTL environment. The 21164 is tested and characterized in a CMOS environment.

9.2.1 Power Supply

The **Vss** pins are connected to 0.0 V, and the **Vdd** pins are connected to 3.3 V, ±5%.

9.2.2 Input Signal Pins

Nearly all input signals are ordinary CMOS inputs with standard TTL levels (see Table 9–2). (See Section 9.3.1 for a description of an exception—**osc_clk_in_h,l**.)

After power has been applied, input and bidirectional pins can be driven to a maximum dc voltage of 6.3 V (6.8 V for 1 ns) without harming the 21164. (It is not necessary to use static RAMs with 3.3-V outputs.)

9.2.3 Output Signal Pins

Output pins are ordinary 3.3-V CMOS outputs. Although output signals are rail-to-rail, timing is specified to $\frac{Vdd}{2}$.

Bidirectional pins are either input or output pins, depending on control timing. When functioning as output pins, they are ordinary 3.3-V CMOS outputs.

Table 9–2 shows the CMOS dc input and output pins.

Table 9–2 CMOS dc Input/Output Characteristics

Parameter		Requirements			
Symbol	**Description**	**Min.**	**Max.**	**Units**	**Test Conditions**
Vih	High-level input voltage	2.0	—	V	—
Vil	Low-level input voltage	—	0.8	V	—
Voh	High-level output voltage	2.4	—	V	**Ioh** = –6.0 mA
Vol	Low-level output voltage	—	0.4	V	**Iol** = 6.0 mA
Iil_pd	Input with pull-down leakage current	—	±50	μA	**Vin** = 0 V
Iih_pd	Input with pull-down current	—	200	μA	**Vin** = 2.4 V
Iil_pu	Input with pull-up current	—	–800	μA	**Vin** = 0.4 V
Iih_pu	Input with pull-up leakage current	—	±50	μA	**Vin** = **Vdd** V
Iozl_pd	Output with pull-down leakage current (tristate)	—	±50	μA	**Vin** = 0 V
Iozh_pd	Output with pull-down current (tristate)	—	200	μA	**Vin** = 2.4 V
Iozl_pu	Output with pull-up current (tristate)	—	–800	μA	**Vin** = 0.4 V
Iozh_pu	Output with pull-up leakage current (tristate)	—	±50	μA	**Vin** = **Vdd** V
Idd	Peak power supply current	—	18	A	**Vdd** = 3.465 V Frequency = 266 MHz
Idd	Peak power supply current	—	20	A	**Vdd** = 3.465 V Frequency = 300 MHz

Most pins have low current pull-down devices to **Vss**. However, two pins have a pull-up device to **Vdd**. The pull-downs (or pull-ups) are always enabled. This means that some current will flow from the 21164 (if the pin has a pull-up device) or into the 21164 (if the pin has a pull-down device) even when the pin is in the high-impedance state. All pins have pull-down devices, except for the pins in the following table:

9.2 dc Characteristics

Signal Name	Notes
tms_h	Has a pull-up device
tdi_h	Has a pull-up device
osc_clk_in_h	50 Ω to **Vterm** ($\approx\frac{Vdd}{2}$) (See Figure 9–1)
osc_clk_in_l	50 Ω to **Vterm** ($\approx\frac{Vdd}{2}$) (See Figure 9–1)
temp_sense	150 Ω to **Vss**

9.3 Clocking Scheme

The differential input clock signals **osc_clk_in_h,l** run at two times the internal frequency of the time base for the 21164. Input clocks are divided by two onchip to generate a 50% duty cycle clock for internal distribution. The output signal **cpu_clk_out_h** toggles with an unspecified propagation delay relative to the transitions on **osc_clk_in_h,l**.

System designers have a choice of two system clocking schemes to run the 21164 synchronous to the system:

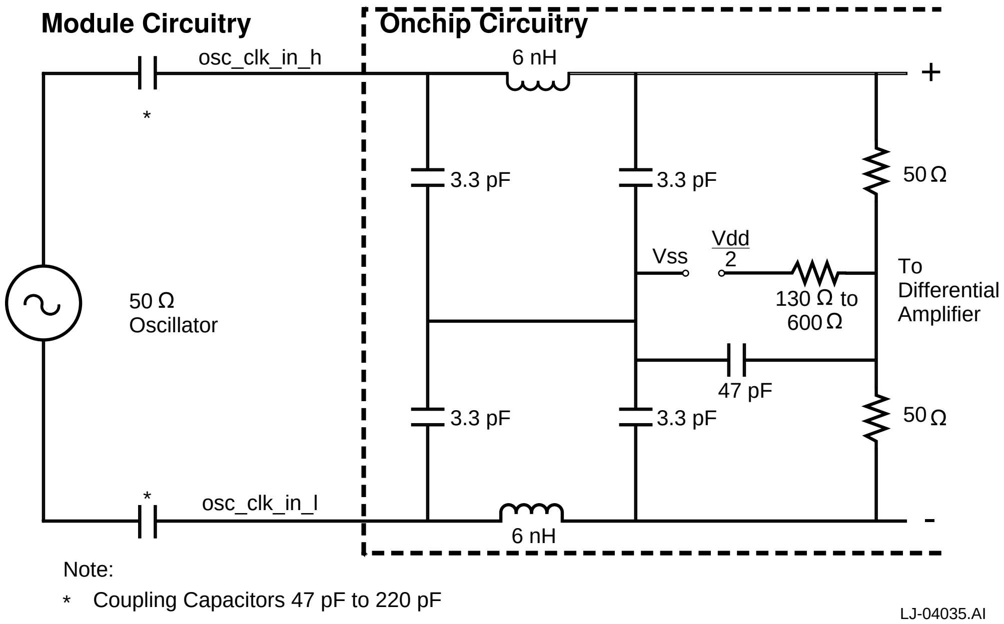
1. The 21164 generates and drives out a system clock, **sys_clk_out1_h,l**. It runs synchronous to the internal clock at a selected ratio of the internal clock frequency. There is a small clock skew between the internal clock and **sys_clk_out1_h,l**.
2. The 21164 synchronizes to a system clock, **ref_clk_in_h**, supplied by the system. The **ref_clk_in_h** clock runs at a selected ratio of the 21164 internal clock frequency. The internal clock is synchronized to the reference clock by an onchip digital phase-locked loop (DPLL).

Refer to Section 4.2 for more information on clock functions.

9.3.1 Input Clocks

The differential input clocks **osc_clk_in_h,l** provide the time base for the chip when **dc_ok_h** is asserted. These pins are self-biasing, and must be capacitively coupled to the clock source on the module, or they can be directly driven. The terminations on these signals are designed to be compatible with system oscillators of arbitrary dc bias. The oscillator must have a duty cycle of 60%/40% or tighter. Figure 9–1 shows the input network and the schematic equivalent of **osc_clk_in_h,l** terminations.

Figure 9–1 osc_clk_in_h,l Input Network and Terminations



Ring Oscillator

When signal **dc_ok_h** is deasserted, the clock outputs follow the internal ring oscillator. The 21164 runs off the ring oscillator, just as it would when an external clock is applied. The frequency of the ring oscillator varies from chip to chip within a range of 10 MHz to 100 MHz. This corresponds to an internal CPU clock frequency range of 5 MHz to 50 MHz. The system clock divisor is forced to 8, and the **sys_clk_out2** delay is forced to 3.

Clock Sniffer

A special onchip circuit monitors the **osc_clk_in** pins and detects when input clocks are not present. When activated, this circuit switches the 21164 clock generator from the **osc_clk_in** pins to the internal ring oscillator. This happens independently of the state of the **dc_ok_h** pin. The **dc_ok_h** pin functions normally if clocks are present on the **osc_clk_in** pins.

9.3.2 Clock Termination and Impedance Levels

In Figure 9–1, the clock is designed to approximate a 50-Ω termination for the purpose of impedance matching for those systems that drive input clocks across long traces. The clock input pins appear as a 50-Ω series termination resistor connected to a high impedance voltage source. The voltage source produces a nominal voltage value of $\frac{Vdd}{2}$. The source has an impedance of between 130 Ω and 600 Ω. This voltage is called the self-bias voltage and sources current when the applied voltage at the clock input pins is less than the self-bias voltage. It sinks current when the applied voltage exceeds the self-bias voltage. This high impedance bias driver allows a clock source of arbitrary dc bias to be ac coupled to the 21164. The peak-to-peak amplitude of the clock source must be between 0.6 V and 3.0 V. Either a square-wave or a sinusoidal source may be used. Full-rail clocks may be driven by testers. In any case, the oscillator should be ac coupled to the **osc_clk_in_h,l** inputs by 47 pF through 220 pF capacitors.

9.3.3 ac Coupling

Using series coupling (blocking) capacitors renders the 21164 clock input pins insensitive to the oscillator's dc level. When connected this way, oscillators with any dc offset relative to **Vss** can be used provided they can drive a signal into the **osc_clk_in_h,l** pins with a peak-to-peak level of at least 600 mV, but no greater than 3.0 V peak to peak.

The value of the coupling capacitor is not overly critical. However, it should be sufficiently low impedance at the clock frequency so that the oscillator's output signal (when measured at the **osc_clk_in_h,l** pins) is not attenuated below the 600 mV peak-to-peak lower limit. For sine waves or oscillators producing nearly sinusoidal (pseudo square wave) outputs, 220 pF is recommended at 533.3 MHz (266.6 MHz $\times$ 2). A high quality dielectric such as NPO is required to avoid dielectric losses.

Table 9–3 shows the input clock specification.

Table 9–3 Input Clock Specification

Signal Parameter	Nominal Bin[1]	Unit
osc_clk_in_h,l symmetry	50 ± 10	%
osc_clk_in_h,l minimum voltage	0.6	V (peak-to-peak)
osc_clk_in_h,l Z input	50	Ω

[1]Minimum clock frequency = 50.0 MHz
Maximum clock frequency = Tcycle (Tcycle = 266.6 MHz or 300 MHz)

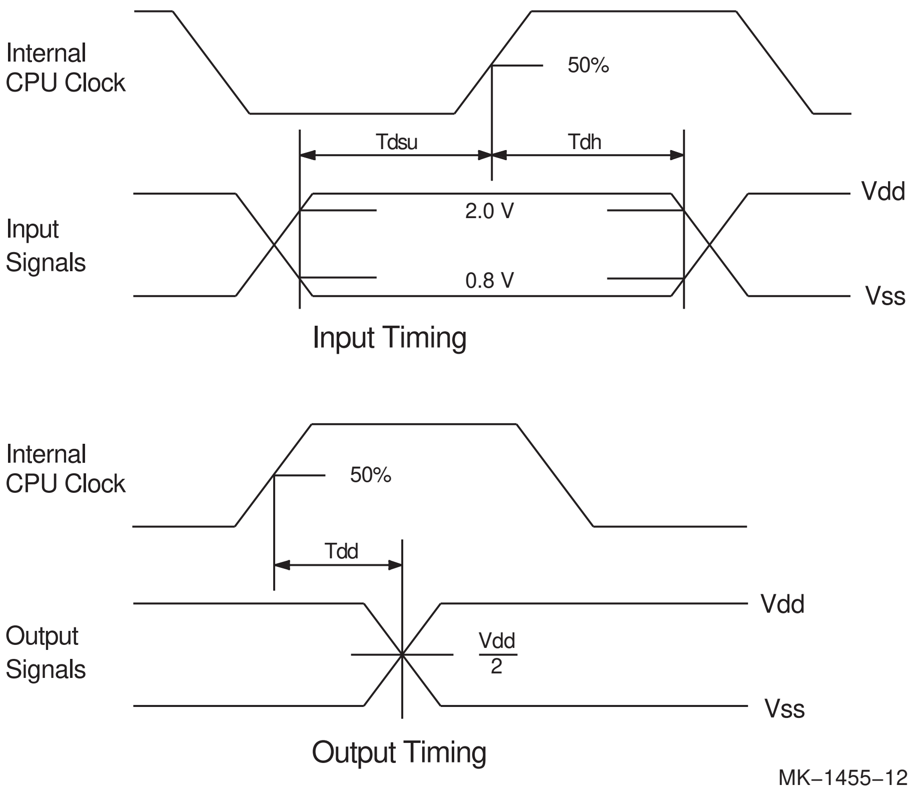
9.4 ac Characteristics

This section describes the ac timing specifications for the 21164.

9.4.1 Test Configuration

All input timing is specified relative to the crossing of standard TTL input levels of 0.8 V and 2.0 V. Output timing is to the nominal CMOS switch point of $\frac{Vdd}{2}$ (see Figure 9–2).

Figure 9–2 Input/Output Pin Timing



9.4 ac Characteristics

Because the speed and complexity of microprocessors has increased substantially over the years, it is necessary to change the way they are tested. Traditional assumptions that all loads can be lumped into some accumulation of capacitance cannot be employed any more. Rather, the model of a transmission line with discrete loads is a much more realistic approach for current test technology.

Typically, printed circuit board (PCB) etch has a characteristic impedance of approximately 75 Ω. This may vary from 60 Ω to 90 Ω with tolerances. If the line is driven in the electrical center, the load could be as low as 30 Ω. Therefore, a characteristic impedance range of 30 Ω to 90 Ω could be experienced.

The 21164 output drivers are designed with typical printed circuit board applications in mind rather than trying to accommodate a 40-pF test load specification. As such, it ''launches'' a voltage step into a characteristic impedance, ranging from 30 Ω to 90 Ω.

To prevent signal quality problems due to overshoot or ringing, ''near end'' terminated transmission line design rules are used. By combining the source impedance of the driver transistors with an additional 20-Ω onchip resistor, a source impedance of approximately 40 Ω is achieved. Additionally, a load value of 10 pF, when added to the PCB etch delays, provides a realistic estimate of actual system timing. When employing this test configuration, the signal at the end of the line will transition cleanly through the TTL input specification range of 0.8 V to 2.0 V without plateaus, or reversal into the range.

9.4.2 Pin Timing

The following sections describe Bcache loop timing, sys_clk-based system timing, and reference clock-based system timing.

9.4.2.1 Backup Cache Loop Timing

The 21164 can be configured to support an optional offchip backup cache (Bcache). Private Bcache read or write (Scache victims) transactions initiated by the 21164 are independent of the system clocking scheme. Bcache loop timing must be an integer multiple of the 21164 cycle time.

Table 9–4 lists the Bcache loop timing.

Table 9–4 Bcache Loop Timing

Signal	Specification	Value	Name
data_h<127:0>	Input setup	1.1 ns	**Tdsu**
data_h<127:0>	Input hold	0.0 ns	**Tdh**
index_h<25:4>	Output delay	**Tdd** + 0.4 ns[1]	**Tiod**
index_h<25:4>	Output hold time	**Tmdd**	**Tioh**
data_h<127:0>	Output delay	**Tdd** + **Tcycle** + 0.4 ns[1]	**Tdod**
data_h<127:0>	Output hold	**Tmdd** + **Tcycle**	**Tdoh**

[1]The value 0.4 ns accounts for onchip driver and clock skew.

Outgoing Bcache index and data signals are driven off the internal clock edge and the incoming Bcache tag and data signals are latched on the same internal clock edge. Table 9–5 shows the output driver characteristics.

Table 9–5 Output Driver Characteristics

Specification	40-pF Load	10-pF Load	Name
Maximum driver delay	2.6 ns	1.6 ns	**Tdd**
Minimum driver delay	1.0 ns	1.0 ns	**Tmdd**

Output pin timing is specified for lumped 40-pF and 10-pF loads. In some cases, the circuit may have loads higher than 40 pF. The 21164 can safely drive higher loads provided the average charging or discharging current from each pin is 10 mA or less. The following equation can be used to determine the maximum capacitance that can be safely driven by each pin:

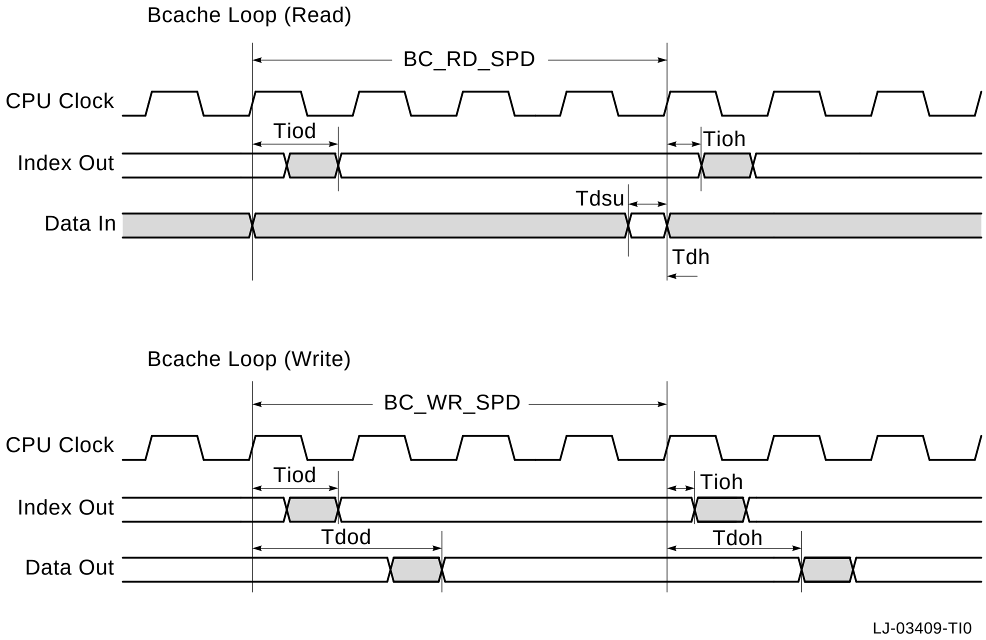
> Cmax (in pF) = 3t, where t is the waveform period (measured from rising to rising or falling to falling edge), in nanoseconds.

For example, if the waveform appearing on a given I/O pin has a 20.4-ns period, it can safely drive up to and including 61 pF.

Figure 9–3 shows the Bcache read and write timing.

9.4 ac Characteristics

Figure 9–3 Bcache Timing



9.4.2.2 sys_clk-Based Systems

All timing is specified relative to the rising edge of the internal CPU clock.

Table 9–6 shows 21164 system clock **sys_clk_out1_h,l** output timing. Setup and hold times are specified independent of the relative capacitive loading of **sys_clk_out1_h,l**, **addr_h<39:4>**, **data_h<127:0>**, and **cmd_h<3:0>** signals. The **ref_clk_in_h** signal must be tied to **Vdd** for proper operation.

Table 9–6 Alpha 21164 System Clock Output Timing (sysclk=$T_{\varnothing}$)

Signal	Specification	Value	Name
sys_clk_out1_h,l	Output delay	**Tdd**	**Tsysd**
sys_clk_out1_h,l	Minimum output delay	**Tmdd**	**Tsysdm**
data_bus_req_h, **data_h<127:0>**, **addr_h<39:4>**	Input setup	1.1 ns	**Tdsu**
data_bus_req_h, **data_h<127:0>**, **addr_h<39:4>**	Input hold	0 ns	**Tdh**
addr_h<39:4>	Output delay	**Tdd** + 0.4 ns[1]	**Taod**
addr_h<39:4>	Output hold time	**Tmdd**	**Taoh**
data_h<127:0>	Output delay	**Tdd** + **Tcycle** + 0.4 ns[1]	**Tdod**[2]
data_h<127:0>	Output hold time	**Tmdd** + **Tcycle**[1]	**Tdoh**[2]
	Non-Pipe_Latch Mode		
addr_bus_req_h	Input setup	3.8 ns	**Tabrsu**
addr_bus_req_h	Input hold	–1.0 ns	**Tabrh**
dack_h	Input setup	3.4 ns	**Tntacksu**
cack_h	Input setup	3.7 ns	**Tntcacksu**
cack, **dack**	Input hold	–1.0 ns	**Tntackh**
	Pipe_Latch Mode[3]		
addr_bus_req_h, **cack_h**, **dack_h**	Input setup	1.1 ns	**Ttacksu**
addr_bus_req_h, **cack_h**, **dack_h**	Input hold	0 ns	**Ttackh**

[1]The value 0.4 ns accounts for onchip driver and clock skew.

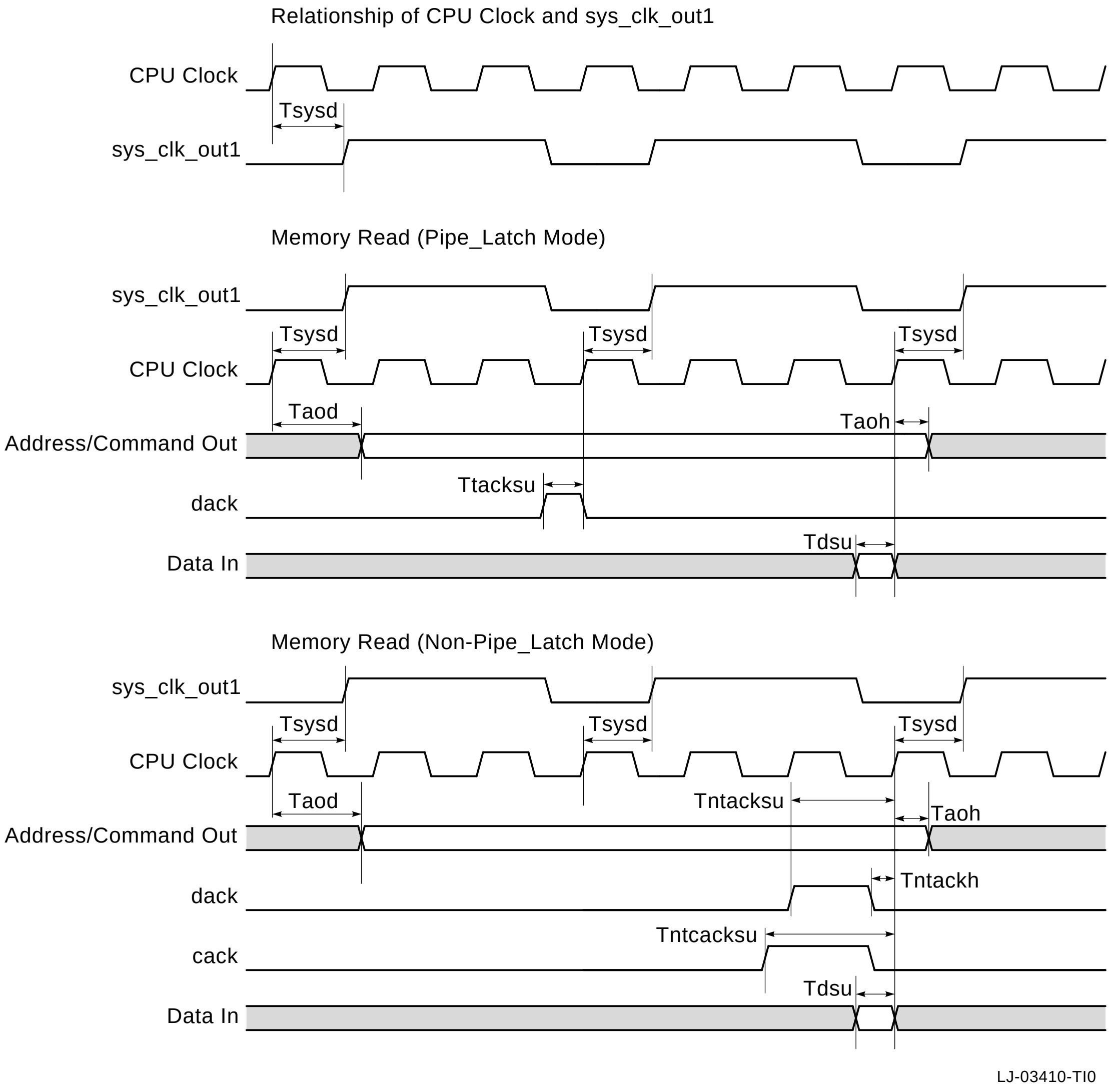
[2]For all write transactions initiated by the 21164, data is driven one CPU cycle after the **sys_clk_out1** or **index_h<25:4>** pins.

[3]In pipe_latch mode, control signals are piped onchip for one **sys_clk_out1_h,l** before usage.

Figure 9–4 shows sys_clk system timing.

9.4 ac Characteristics

Figure 9–4 sys_clk System Timing



9.4.2.3 Reference Clock-Based Systems

Systems that generate their own system clock expect the 21164 to synchronize its **sys_clk_out1_h,l** outputs to their system clock. The 21164 uses a digital phase-locked loop (DPLL) to synchronize its **sys_clk_out1** signals to the system clock that is applied to the **ref_clk_in_h** signal. For additional information on reference clock timing, refer to Section 4.2.4.

Table 9–7 shows all timing relative to the rising edge of **ref_clk_in_h**.

Table 9–7 Alpha 21164 Reference Clock Input Timing

Signal	Specification	Value	Name
data_bus_req_h, **data_h<127:0>**, **addr_h<39:4>**	Input setup	1.1 ns	**Tdsu**
data_bus_req_h, **data_h<127:0>**, **addr_h<39:4>**	Input hold	0.5 x **Tcycle**	**Troh**
addr_h<39:4>	Output delay	**Tdd** + 0.5 x **Tcycle** + 0.9 ns[1]	**Traod**
addr_h<39:4>	Output hold time	**Tmdd**	**Traoh**
data_h<127:0>	Output delay	**Tdd** + 1.5 x **Tcycle** + 0.9 ns[1]	**Trdod**[2]
data_h<127:0>	Output hold time	**Tmdd + Tcycle**	**Trdoh**[2]
		Non-Pipe_Latch Mode	
addr_bus_req_h	Input setup	3.8 ns	**Tntrabrsu**
addr_bus_req_h	Input hold	0.5 x **Tcycle**	**Tntrabrh**
dack_h	Input setup	3.3 ns	**Tntracksu**
cack_h	Input setup	3.7 ns	**Tntrcacksu**
cack_h, **dack_h**	Input hold	(0.5 x **Tcycle**)	**Tntrackh**

[1]The value 0.9 ns accounts for onchip skews that include 0.4 ns for driver and clock skew, phase detector skews due to circuit delay (0.2 ns), and delay in **ref_clk_in_h** due to the package (0.3 ns).

[2]For all write transactions initiated by the 21164, data is driven one CPU cycle later.

(continued on next page)

Table 9–7 (Cont.) Alpha 21164 Reference Clock Input Timing

Signal	Specification	Value	Name
		Pipe_Latch Mode[3]	
addr_bus_req_h, **cack_h**, **dack_h**	Input setup	1.1 ns	**Ttracksu**
addr_bus_req_h, **cack_h**, **dack_h**	Input hold	0.5 x **Tcycle**	**Ttrackh**

[3]In pipe_latch mode, control signals are piped onchip for one **sys_clk_out1_h,l** before usage.

9.4.3 Digital Phase-Locked Loop

Figure 9–5 and Table 9–8 describe the digital phase-locked loop (DPLL) stages of operation.

Figure 9–5 ref_clk System Timing

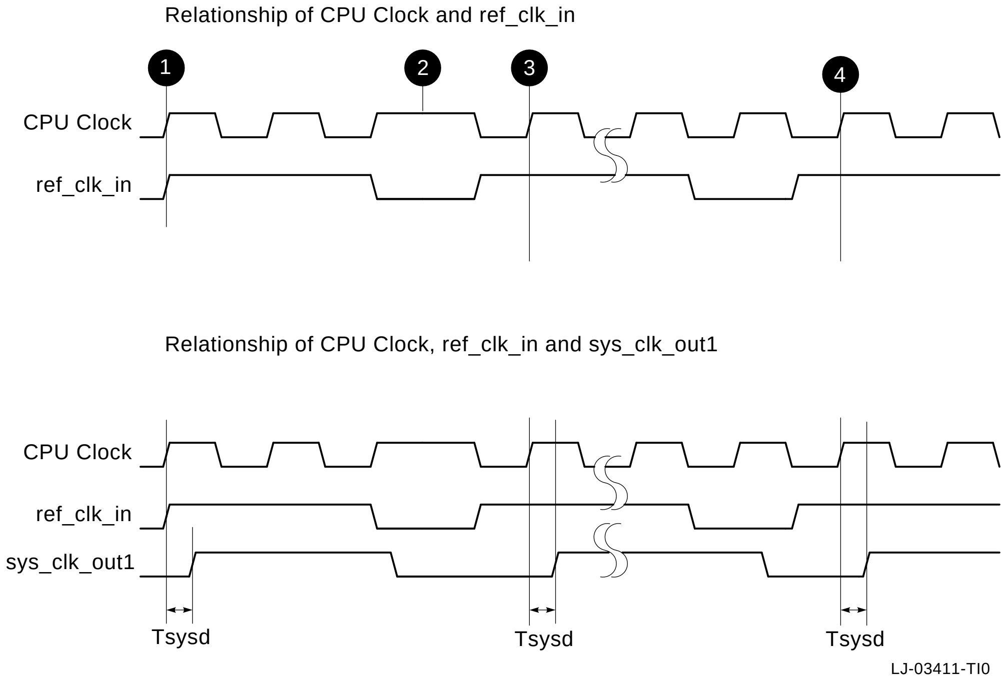


Table 9–8 ref_clk System Timing Stages

Stage	Description
❶	The internal CPU clock rising edge coincides with the rising edge of **ref_clk_in_h**.
❷	The DPLL causes the internal CPU clock to stretch for one phase (1 cycle of **osc_clk_in_h,l**).
❸	The stretch causes **ref_clk_in_h** to lead the internal CPU clock by one phase.
❹	The CPU clock is always slightly faster than the external **ref_clk_in_h** and gains on **ref_clk_in_h** over time. Eventually the gain equals one phase and a new stretch phase follows.

Although systems that supply a **ref_clk_in_h** do not use **sys_clk_out1_h,l**, a relationship between the two signals exists, just as in the **sys_clk**-based systems, because the 21164 uses **sys_clk_out1_h,l** internally to determine timing during system transactions.

9.4.4 Timing—Additional Signals

This section lists timing for all other signals.

Asynchronous Input Signals

The following is a list of the asynchronous input signals:

clk_mode_h	**dc_ok_h**	**ref_clk_in_h**	
sys_reset_l[1]			
perf_mon_h[2]			
irq_h<3:0>[2]	**mch_hlt_irq_h**[2]	**pwr_fail_irq_h**[2]	**sys_mch_chk_irq_h**[2]

[1] Signal **sys_reset_l** may be deasserted synchronously.

[2] These signals can also be used synchronously.

Miscellaneous Signals

Table 9–9 and Table 9–10 list the timing for miscellaneous input-only and output-only signals. All timing is expressed in nanoseconds.

Table 9–9 Input Timing for sys_clk_out- or ref_clk_in-Based Systems

		Value		Name	
Signal	**Specification**	**sys_clk_out**	**ref_clk_in**	**sys_clk_out**	**ref_clk_in**
cfail_h, **fill_h**, **fill_error_h**, **fill_id_h**, **fill_nocheck_h**, **idle_bc_h**, **shared_h**, **system_lock_flag_h** **irq_h<3:0>**, **mch_hlt_irq_h**, **pwr_fail_irq_h**, **sys_mch_chk_irq_h** Testability pins: **port_mode_h**, **srom_data_h**, **srom_present_l**	Input setup	1.1 ns	1.1 ns	**Tdsu**	**Tdsu**
cfail_h, **fill_h**, **fill_error_h**, **fill_id_h**, **fill_nocheck_h**, **idle_bc_h**, **shared_h**, **system_lock_flag_h** **irq_h<3:0>**, **mch_hlt_irq_h**, **pwr_fail_irq_h**, **sys_mch_chk_irq_h** **sys_reset_l** Testability pins: **port_mode_h**, **srom_data_h**, **srom_present_l**	Input hold	0 ns	0.5 ∗**Tcycle**	**Tdh**	**Troh**

Table 9–10 Output Timing for sys_clk_out- or ref_clk_in-Based Systems

		Clocking System Value		Clocking System Name	
Signal	**Specification**	**sys_clk_out**	**ref_clk_in**	**sys_clk_out**	**ref_clk_in**
Unidirectional Signals					
addr_res_h, **int4_valid_h**,[1] **scache_set_h**, **srom_clk_h**, **srom_oe_l**, **victim_pending_h**	Output delay	**Tdd**+0.4 ns	**Tdd**+0.5 ∗**Tcycle**+0.9 ns	**Taod**	**Traod**
addr_res_h, **int4_valid_h**,[1] **scache_set_h**, **srom_clk_h**, **srom_oe_l**, **victim_pending_h**	Output hold	**Tmdd**	**Tmdd**	**Taoh**	**Traoh**
int4_valid_h[2]	Output delay	**Tdd**+**Tcycle**+0.4 ns	**Tdd**+1.5∗**Tcycle**+0.9 ns	**Tdod**	**Trdod**
int4_valid_h[2]	Output hold	**Tmdd+Tcycle**	**Tmdd+Tcycle**	**Tdoh**	**Trdoh**
Bidirectional Signals					
Input mode:					
addr_cmd_par_h, **cmd_h**, **data_check_h**,[1] **tag_ctl_par_h**,[3] **tag_dirty_h**,[3] **tag_shared_h**[3]	Input setup	1.1 ns	1.1 ns	**Tdsu**	**Tdsu**
addr_cmd_par_h, **cmd_h**, **data_check_h**,[1] **tag_ctl_par_h**,[3] **tag_dirty_h**,[3] **tag_shared_h**[3]	Input hold	0 ns	0.5∗**Tcycle**	**Tdh**	**Tsdadh**

[1]Read transaction
[2]Write transaction
[3]Fills from memory

(continued on next page)

9.4 ac Characteristics

Table 9–10 (Cont.) Output Timing for sys_clk_out- or ref_clk_in-Based Systems

		Clocking System Value		Clocking System Name	
Signal	**Specification**	**sys_clk_out**	**ref_clk_in**	**sys_clk_out**	**ref_clk_in**
Bidirectional Signals					
Output mode:					
addr_cmd_par_h, **cmd_h**, **tag_ctl_par_h**,[4] **tag_dirty_h**,[4] **tag_shared_h**,[4] **tag_valid_h**[4]	Output delay	**Tdd**+0.4 ns	**Tdd**+0.5 ∗**Tcycle**+0.9 ns	**Taod**	**Traod**
data_check_h[2]	Output delay	**Tdd+Tcycle**+0.4 ns	**Tdd**+1.5∗**Tcycle**+0.9 ns	**Tdod**	**Trdod**
addr_cmd_par_h, **cmd_h**, **tag_ctl_par_h**,[4] **tag_dirty_h**,[4] **tag_shared_h**,[4] **tag_valid_h**[4]	Output hold	**Tmdd**	**Tmdd**	**Taoh**	**Traoh**
data_check_h[2]	Output hold	**Tmdd+Tcycle**	**Tmdd+Tcycle**	**Tdoh**	**Trdoh**

[2]Write transaction

[4]Only for write broadcasts and system transactions

Signals in Table 9–11 are used to control Bcache data transfers. These signals are driven off the CPU clock. The choice of **sys_clk_out** or **ref_clk_in** has no impact on the timing of these signals.

Table 9–11 Bcache Control Signal Timing

Signal	Specification	Value	Name
Input mode:			
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Input setup	1.1 ns	**Tdsu**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Input hold	0 ns	**Tdh**
Output mode:			
data_ram_oe_h, **data_ram_we_h**,[1] **tag_ram_oe_h**, **tag_ram_we_h**[1]	Output delay	**Tdd**+0.4 ns	**Taod**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Output delay	**Tdd**+0.4 ns	**Taod**
data_ram_oe_h, **data_ram_we_h**,[1] **tag_ram_oe_h**, **tag_ram_we_h**[1]	Output hold	**Tmdd**	**Taoh**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Output hold	**Tmdd**	**Taoh**

[1]Pulse width for this signal is controlled through the BC_CONFIG IPR.

9.4.5 Timing of Test Features

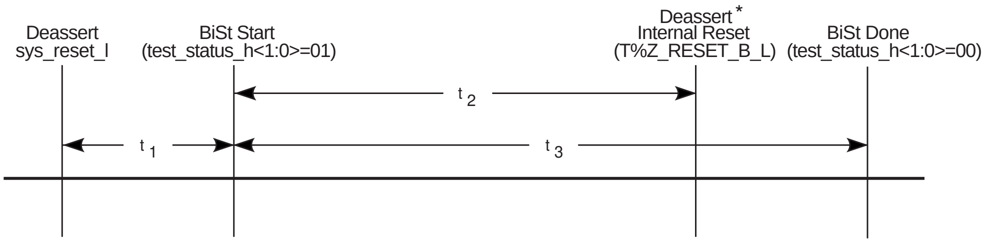
Timing of 21164 testability features depends on the system clock rate and the test port's operating mode. This section provides timing information that may be needed for most common operations.

9.4.6 Icache BiSt Operation Timing

The Icache BiSt is invoked by deasserting the external reset signal **sys_reset_l**. Figure 9–6 shows the timing between various events relevant to BiSt operations.

9.4 ac Characteristics

Figure 9–6 BiSt Timing Event–Time Line



The timing for deassertion of internal reset (time t_2, see asterisk) is valid only if an SROM is not present (indicated by keeping signal **srom_present_l** deasserted). If an SROM is present, the SROM load is performed once the BiSt completes. The internal reset signal T%Z_RESET_B_L is extended until the end of the SROM load (Section 9.4.7). In this case, the end of the time line shown in Figure 9–6 connects to the beginning of the time line shown in Figure 9–7.

Table 9–12 and Table 9–13 list timing shown in Figure 9–6 for some of the system clock ratios. Time t_1 is measured starting from the rising edge of sysclk following the deassertion of the **sys_reset_l** signal.

Table 9–12 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (System Cycles)

Sysclk	System Cycles		
Ratio	t_1	t_2	t_3
3	8	22644+2½	22645
4	7	19721+2½	19722
15	7	13291+14½	13292

Table 9–13 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (CPU Cycles)

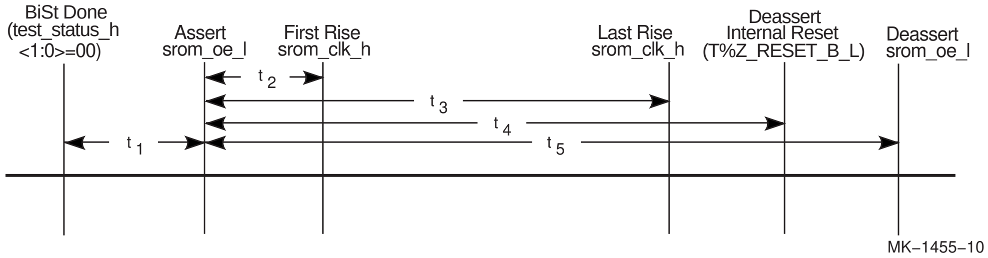
Sysclk	CPU Cycles		
Ratio	t_1	t_2	t_3
3	24	67934½	67935
4	28	78886½	78888
15	105	199379½	199380

9.4.7 Automatic SROM Load Timing

The SROM load is triggered by the conclusion of BiSt if **srom_present_l** is asserted. The SROM load occurs at the internal cycle time of approximately 126 CPU cycles for **srom_clk_h**, but the behavior at the pins may shift slightly. Refer to Chapter 7 for more information on input signals, booting, and the SROM interface port.

Timing events are shown in Figure 9–7 and are listed in Table 9–14 and Table 9–15.

Figure 9–7 SROM Load Timing Event–Time Line



9.4 ac Characteristics

Table 9–14 SROM Load Timing for Some System Clock Ratios (System Cycles)

Sysclk	System Cycles[1]				
Ratio	t_1	t_2	t_3	t_4	t_5
3	4	22	4408090	4408216+½	4408217
4	3	48	3306099	3306193+2½	3306194
15	3	13	881627	881651+9½	881652

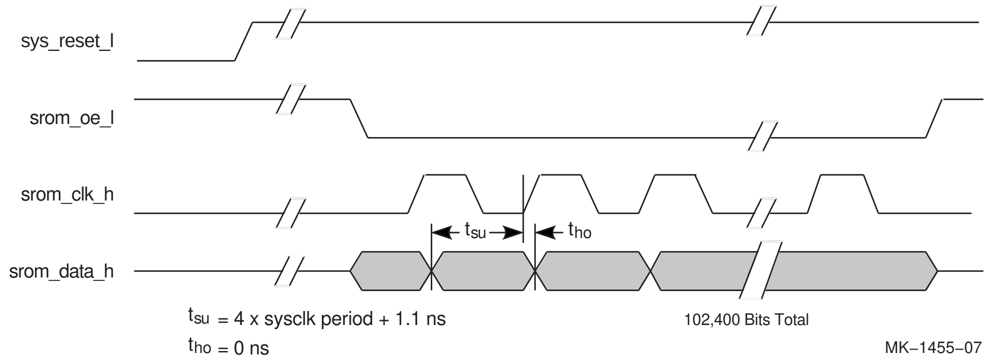
[1]Measured in sysclk cycles, where +*n* refers to an additional *n* CPU cycles.

Table 9–15 SROM Load Timing for Some System Clock Ratios (CPU Cycles)

Sysclk	CPU Cycles				
Ratio	t_1	t_2	t_3	t_4	t_5
3	12	66	13224270	13224648½	13224651
4	12	192	13224396	13224774½	13224776
15	45	195	13224405	13224774½	13224780

Figure 9–8 is a timing diagram of an SROM load sequence.

Figure 9–8 Serial ROM Load Timing



The minimum **srom_clk_h** cycle = (126 − sysclk ratio) ∗ (CPU cycle time).

The maximum **srom_clk_h** to **srom_data_h** delay allowable (in order to meet the required setup time) = [126 − (5 ∗ sysclk ratio)] ∗ (CPU cycle time).

9.4.8 Clock Test Modes

This section describes the 21164 clock test modes.

9.4.8.1 Normal Mode

When the **clk_mode_h<1:0>** signals are not asserted, the **osc_clk_in_h,l** frequency is divided by 2. This is the normal operational mode of the clock circuitry.

9.4.8.2 Chip Test Mode

To lower the maximum frequency that the chip manufacturing tester is required to supply, a divide-by-1 mode has been designed into the clock generator circuitry. When the **clk_mode_h<0>** signal is asserted and **clk_mode_h<1>** is not asserted, the clock frequency that is applied to the input clock signals **osc_clk_in_h,l** bypasses the clock divider and is sent to the chip clock driver. This allows the chip internal circuitry to be tested at full speed with a one-half frequency **osc_clk_in_h,l**.

9.4.8.3 Module Test Mode

When the **clk_mode_h<0>** signal is not asserted and **clk_mode_h<1>** is asserted, the clock frequency that is applied to the input clock signals **osc_clk_in_h,l** is divided by 4 and is sent to the chip clock driver. The digital phase-locked loop (DPLL) continues to keep the onchip **sys_clk_out1_h,l** locked to **ref_clk_in_h** within the normal limits if a **ref_clk_in_h** signal is applied (0 ns to 1 **osc_clk_in_h,l** cycle after **ref_clk_in_h**).

9.4.8.4 Clock Test Reset Mode

When both the **clk_mode_h<0>** and the **clk_mode_h<1>** signals are asserted, the **sys_clk_out** generator circuit is forced to reset to a known state. This allows the chip manufacturing tester to synchronize the chip to the tester cycle. Table 9–16 lists the test modes.

Table 9–16 Test Modes

Mode	clk_mode_h<0>	clk_mode_h<1>
Normal	0	0
Chip test	1	0
Module test	0	1
Clock reset	1	1

9.4.9 IEEE 1149.1 (JTAG) Performance

Table 9–17 lists the standard mandated performance specifications for the IEEE 1149.1 circuits.

Table 9–17 IEEE 1149.1 Circuit Performance Specifications

Item	Specification
trst_l is asynchronous. Minimum pulse width.	4 ns
trst_l setup time for deassertion before a transition on **tck_h**.	4 ns
Maximum acceptable **tck_h** clock frequency.	16.6 MHz
tdi_h/tms_h setup time (referenced to **tck_h** rising edge).	4 ns
tdi_h/tms_h hold time (referenced to **tck_h** rising edge).	4 ns
Maximum propagation delay at pin **tdo_h** (referenced to **tck_h** falling edge).	14 ns
Maximum propagation delay at system output pins (referenced to **tck_h** falling edge).	20 ns

9.5 Power Supply Considerations

For correct operation of the 21164, all of the **Vss** pins must be connected to ground and all of the **Vdd** pins must be connected to a 3.3 V ±5% power source. This source voltage should be guaranteed (even under transient conditions) at the 21164 pins, and not just at the PCB edge.

Plus 5 V is not used in the 21164. The voltage difference between the **Vdd** pins and **Vss** pins must never be greater than 3.6 V. If the differential exceeds this limit, the 21164 chip will be damaged.

9.5.1 Decoupling

The effectiveness of decoupling capacitors depends on the amount of inductance placed in series with them. The inductance depends both on the capacitor style (construction) and on the module design. In general, the use of small, high frequency capacitors placed close to the chip package's power and ground pins with very short module etch will give best results. Depending on the user's power supply and power supply distribution system, bulk decoupling may also be required on the module.

Each individual case must be separately analyzed, but generally designers should plan to use at least 6 μF of capacitance. Typically, 40 to 60 small, high frequency 0.1-μF capacitors are placed near the chip's **Vdd** and **Vss** pins. Actually placing the capacitors in the pin field is the best approach. Several tens of μF of bulk decoupling (comprised of tantalum and ceramic capacitors) should be positioned near the 21164 chip.

Use capacitors that are as physically small as possible. Connect the capacitors directly to the 21164 **Vdd** and **Vss** pins by short (0.64 cm [0.25 in] or less) surface etch. The small capacitors generally have better electrical characteristics than the larger units, and will more readily fit close to the IPGA pin field.

9.5.2 Power Supply Sequencing

Although the 21164 uses a 3.3-V (nominal) power source, most of the other logic on the PCB probably requires a 5-V power supply. These 5-V devices can damage the 21164's I/O circuits if the 5-V power source powering the PCB logic and the 3.3-V (**Vdd**) supply feeding the 21164 are not sequenced correctly.

Caution

To avoid damaging the 21164's I/O circuits, the I/O pin voltages must not exceed 4 V until the **Vdd** supply is at least 3 V or greater.

This rule can be satisfied if the **Vdd** and the 5-V supplies come up together, or if the **Vdd** supply comes up before the 5-V supply is asserted. Bringing the lower voltage up before the higher voltage is the opposite of the way that CMOS systems with multiple power supplies of different voltages are usually sequenced, but it is required for the 21164.

A three-terminal voltage regulator can be used to make 3.3-V **Vdd** from the 5-V supply, provided the output of the regulator (**Vdd**) tracks the 5-V supply with only a small offset. The requirement is that when the 5-V supply reaches 4.0 V, **Vdd** must be 3.0 V or higher. While the 5-V supply is below 4.0 V, **Vdd** can be less than 3.0 V.

All 5-V sources on the 21164's I/O pins should be disabled if the power supply sequencing is such that the 5-V supply will exceed 4.0 V before **Vdd** is at least 3.0 V. The 5-V sources should remain disabled until the **Vdd** power supply is equal to or greater than 3.0 V.

9.5 Power Supply Considerations

Disabling all 5-V sources can be very difficult because there are so many possible sneak paths. Inputs, for example, on bipolar TTL logic can be a source of current, and will put a voltage across a 21164 I/O pin high enough to violate the (no higher than 4.0 V until there is 3.0 V) rule. TTL outputs are specified to drive a logic one to at least 2.4 V, but usually drive voltages much higher. CMOS logic and CMOS SRAMs usually drive ‘‘full rail’’ signals that match the value of the 5-V power supply.

Another concern is parallel (dc) terminations or pull-ups connected between the 21164 and the 5-V supply. The 3.3 V (**Vdd**) supply should be used to power parallel terminations.

Disabling the non-21164 5-V outputs of PCB logic is generally possible, but raises the PCB complexity and can reduce system performance by increasing critical path timing. If the 5-V logic device has an enable pin, circuits (such as power supply supervisor chips) on the PCB can monitor the **Vdd** and 5-V supplies. When the supervision circuit detects that 5.0 V is increasing from zero while the **Vdd** supply is below 3.0 V, the power supply supervisor circuit produces a disable signal to force all PCB logic with 5-V outputs into the high impedance state. This technique will not prevent bipolar TTL inputs from acting as a 5-V source, but it can be used to disable sources such as cache RAM outputs.

10

Thermal Management

This chapter describes the 21164 thermal management and thermal design considerations.

10.1 Operating Temperature

The 21164 is specified to operate when the temperature at the center of the heat sink (T_c) is 72°C (266 MHz) or 70°C (300 MHz). Temperature (T_c) should be measured at the center of the heat sink (between the two package studs). The GRAFOIL pad is the interface material between the package and the heat sink.

Table 10–1 lists the values for the center of heat-sink-to-ambient ($\theta_c a$) for the 499-pin grid array. Table 10–2 shows the allowable T_a (without exceeding T_c) at various airflows.

Note

Digital recommends using the heat sink because it greatly improves the ambient temperature requirement.

10.1 Operating Temperature

Table 10–1 $\theta_c a$ at Various Airflows

Airflow (linear ft/min)	100	200	400	600	800	1000
Frequency: 266 MHz and 300 MHz						
$\theta_c a$ with heat sink 1 (°C/W)	2.30	1.30	0.70	0.53	0.45	0.41
$\theta_c a$ with heat sink 2 (°C/W)	1.25	0.75	0.48	0.40	0.35	0.32

Table 10–2 Maximum T_a at Various Airflows

Airflow (linear ft/min)	100	200	400	600	800	1000
Frequency: 266 MHz, Power: 46 W @Vdd = 3.3 V						
T_a with heat sink 1 (°C)	—	—	39.8	47.6	51.3	53.2
T_a with heat sink 2 (°C)	14.5	37.5	49.9	53.6	55.9	57.3
Frequency: 300 MHz, Power: 51 W @Vdd = 3.3 V						
T_a with heat sink 1 (°C)	—	—	34.3	43.0	47.1	49.1
T_a with heat sink 2 (°C)	—	31.8	45.5	49.6	52.2	53.7

10.2 Heat Sink Specifications

Two heat sinks are specified. Heat sink type 1 mounting holes are in line with the cooling fins. Heat sink type 2 mounting holes are rotated 90° from the cooling fins. The heat sink composition is aluminum alloy 6063. Type 1 heat sink is shown in Figure 10–1, and type 2 heat sink is shown in Figure 10–2, along with their approximate dimensions.

Figure 10–1 Type 1 Heat Sink

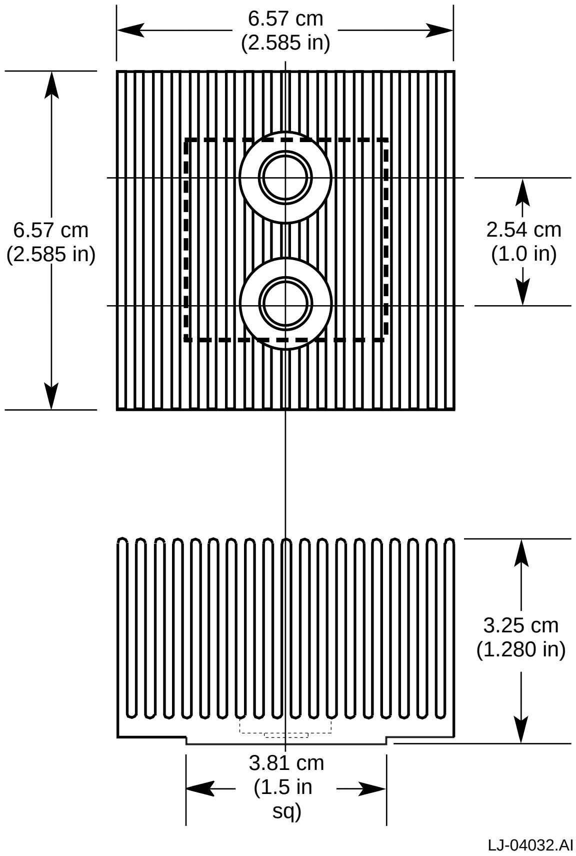
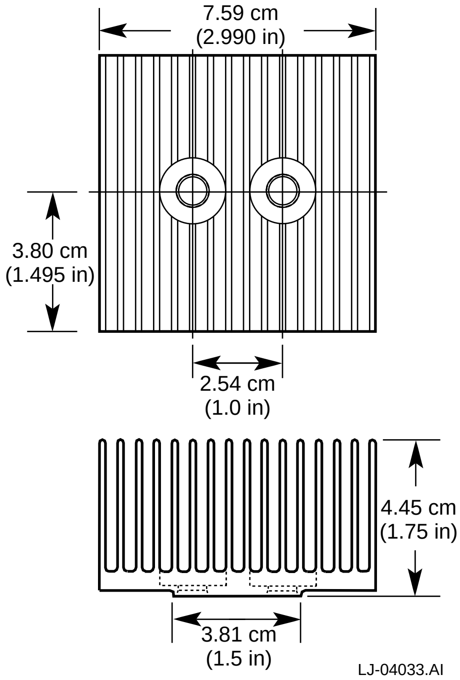


Figure 10–2 Type 2 Heat Sink



10.3 Thermal Design Considerations

Follow these guidelines for printed circuit board (PCB) component placement:

- Orient the 21164 on the PCB with the heat sink fins aligned with the airflow direction.
- Avoid preheating ambient air. Place the 21164 on the PCB so that inlet air is not preheated by any other PCB components.
- Do not place other high power devices in the vicinity of the 21164.
- Do not restrict the airflow across the 21164 heat sink. Placement of other devices must allow for maximum system airflow in order to maximize the performance of the heat sink.

11
Mechanical Data and Packaging Information

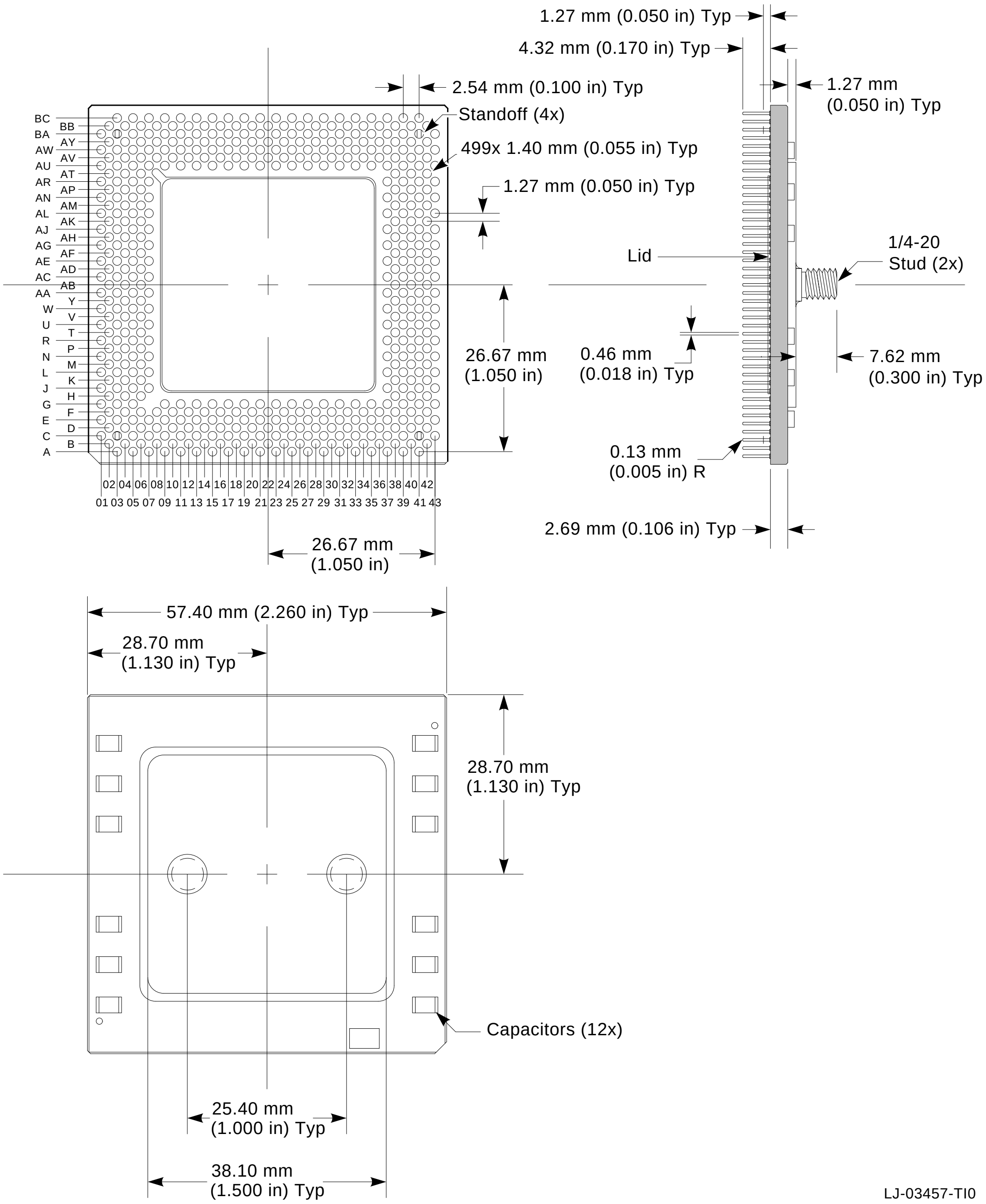
This chapter describes the 21164 mechanical packaging including chip package physical specifications and a signal/pin list. For heat sink dimensions, refer to Chapter 10.

11.1 Mechanical Specifications

Figure 11–1 shows the package physical dimensions without a heat sink.

11.1 Mechanical Specifications

Figure 11–1 Package Dimensions



11.2 Signal Descriptions and Pin Assignment

This section provides detailed information about the 21164 pinout. The 21164 has 499 pins aligned in an interstitial pin grid array (IPGA) design.

11.2.1 Signal Pin Lists

Table 11–1 lists the 21164 signal pins and their corresponding pin grid array (PGA) locations in alphabetic order. There are 292 functional signal pins, 2 spare (unused) signal pins, 104 power (**Vdd**) pins, and 101 ground (**Vss**) pins, for a total of 499 pins in the array.

Table 11–1 Alphabetic Signal Pin List

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
addr_bus_req_h	E23	**addr_cmd_par_h**	B20	**addr_h<4>**	BB14
addr_h<5>	BC13	**addr_h<6>**	BA13	**addr_h<7>**	AV14
addr_h<8>	AW13	**addr_h<9>**	BC11	**addr_h<10>**	BA11
addr_h<11>	AV12	**addr_h<12>**	AW11	**addr_h<13>**	BC09
addr_h<14>	BA09	**addr_h<15>**	AV10	**addr_h<16>**	AW09
addr_h<17>	BC07	**addr_h<18>**	BA07	**addr_h<19>**	AV08
addr_h<20>	AW07	**addr_h<21>**	BC05	**addr_h<22>**	BC39
addr_h<23>	AW37	**addr_h<24>**	AV36	**addr_h<25>**	BA37
addr_h<26>	BC37	**addr_h<27>**	AW35	**addr_h<28>**	AV34
addr_h<29>	BA35	**addr_h<30>**	BC35	**addr_h<31>**	AW33
addr_h<32>	AV32	**addr_h<33>**	BA33	**addr_h<34>**	BC33
addr_h<35>	AW31	**addr_h<36>**	AV30	**addr_h<37>**	BA31
addr_h<38>	BC31	**addr_h<39>**	BB30	**addr_res_h<0>**	C27
addr_res_h<1>	F26	**addr_res_h<2>**	E27	**cack_h**	G21
cfail_h	C25	**clk_mode_h<0>**	AU21	**clk_mode_h<1>**	BA23
cmd_h<0>	F20	**cmd_h<1>**	A19	**cmd_h<2>**	C19
cmd_h<3>	E19	**cpu_clk_out_h**	BA25	**dack_h**	B24
data_bus_req_h	E25	**data_check_h<0>**	J41	**data_check_h<1>**	K38
data_check_h<2>	J39	**data_check_h<3>**	G43	**data_check_h<4>**	G41

(continued on next page)

11.2 Signal Descriptions and Pin Assignment

Table 11–1 (Cont.) Alphabetic Signal Pin List

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
data_check_h<5>	H38	**data_check_h<6>**	G39	**data_check_h<7>**	E43
data_check_h<8>	J03	**data_check_h<9>**	K06	**data_check_h<10>**	J05
data_check_h<11>	G01	**data_check_h<12>**	G03	**data_check_h<13>**	H06
data_check_h<14>	G05	**data_check_h<15>**	E01	**data_h<0>**	J43
data_h<1>	L39	**data_h<2>**	M38	**data_h<3>**	L41
data_h<4>	L43	**data_h<5>**	N39	**data_h<6>**	P38
data_h<7>	N41	**data_h<8>**	N43	**data_h<9>**	P42
data_h<10>	R39	**data_h<11>**	T38	**data_h<12>**	R41
data_h<13>	R43	**data_h<14>**	U39	**data_h<15>**	V38
data_h<16>	U41	**data_h<17>**	U43	**data_h<18>**	W39
data_h<19>	W41	**data_h<20>**	W43	**data_h<21>**	Y38
data_h<22>	Y42	**data_h<23>**	AA39	**data_h<24>**	AA41
data_h<25>	AA43	**data_h<26>**	AB38	**data_h<27>**	AC43
data_h<28>	AC41	**data_h<29>**	AC39	**data_h<30>**	AD42
data_h<31>	AD38	**data_h<32>**	AE43	**data_h<33>**	AE41
data_h<34>	AE39	**data_h<35>**	AG43	**data_h<36>**	AG41
data_h<37>	AF38	**data_h<38>**	AG39	**data_h<39>**	AJ43
data_h<40>	AJ41	**data_h<41>**	AH38	**data_h<42>**	AJ39
data_h<43>	AK42	**data_h<44>**	AL43	**data_h<45>**	AL41
data_h<46>	AK38	**data_h<47>**	AL39	**data_h<48>**	AN43
data_h<49>	AN41	**data_h<50>**	AM38	**data_h<51>**	AN39
data_h<52>	AR43	**data_h<53>**	AR41	**data_h<54>**	AP38
data_h<55>	AR39	**data_h<56>**	AU43	**data_h<57>**	AU41
data_h<58>	AT38	**data_h<59>**	AU39	**data_h<60>**	AW43
data_h<61>	AW41	**data_h<62>**	AV38	**data_h<63>**	AW39
data_h<64>	J01	**data_h<65>**	L05	**data_h<66>**	M06
data_h<67>	L03	**data_h<68>**	L01	**data_h<69>**	N05
data_h<70>	P06	**data_h<71>**	N03	**data_h<72>**	N01

(continued on next page)

Table 11–1 (Cont.) Alphabetic Signal Pin List

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
data_h<73>	P02	**data_h<74>**	R05	**data_h<75>**	T06
data_h<76>	R03	**data_h<77>**	R01	**data_h<78>**	U05
data_h<79>	V06	**data_h<80>**	U03	**data_h<81>**	U01
data_h<82>	W05	**data_h<83>**	W03	**data_h<84>**	W01
data_h<85>	Y06	**data_h<86>**	Y02	**data_h<87>**	AA05
data_h<88>	AA03	**data_h<89>**	AA01	**data_h<90>**	AB06
data_h<91>	AC01	**data_h<92>**	AC03	**data_h<93>**	AC05
data_h<94>	AD02	**data_h<95>**	AD06	**data_h<96>**	AE01
data_h<97>	AE03	**data_h<98>**	AE05	**data_h<99>**	AG01
data_h<100>	AG03	**data_h<101>**	AF06	**data_h<102>**	AG05
data_h<103>	AJ01	**data_h<104>**	AJ03	**data_h<105>**	AH06
data_h<106>	AJ05	**data_h<107>**	AK02	**data_h<108>**	AL01
data_h<109>	AL03	**data_h<110>**	AK06	**data_h<111>**	AL05
data_h<112>	AN01	**data_h<113>**	AN03	**data_h<114>**	AM06
data_h<115>	AN05	**data_h<116>**	AR01	**data_h<117>**	AR03
data_h<118>	AP06	**data_h<119>**	AR05	**data_h<120>**	AU01
data_h<121>	AU03	**data_h<122>**	AT06	**data_h<123>**	AU05
data_h<124>	AW01	**data_h<125>**	AW03	**data_h<126>**	AV06
data_h<127>	AW05	**data_ram_oe_h**	F22	**data_ram_we_h**	A23
dc_ok_h	AU23	**fill_error_h**	A25	**fill_h**	G23
fill_id_h	F24	**fill_nocheck_h**	G25	**idle_bc_h**	A27
index_h<4>	A29	**index_h<5>**	C29	**index_h<6>**	F28
index_h<7>	E29	**index_h<8>**	B30	**index_h<9>**	A31
index_h<10>	C31	**index_h<11>**	F30	**index_h<12>**	E31
index_h<13>	A33	**index_h<14>**	C33	**index_h<15>**	F32
index_h<16>	E33	**index_h<17>**	A35	**index_h<18>**	C35
index_h<19>	F34	**index_h<20>**	E35	**index_h<21>**	A37
index_h<22>	C37	**index_h<23>**	F36	**index_h<24>**	E37

(continued on next page)

11.2 Signal Descriptions and Pin Assignment

Table 11–1 (Cont.) Alphabetic Signal Pin List

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
index_h<25>	A39	**int4_valid_h<0>**	F38	**int4_valid_h<1>**	E41
int4_valid_h<2>	F06	**int4_valid_h<3>**	E03	**irq_h<0>**	BA29
irq_h<1>	AU27	**irq_h<2>**	BC29	**irq_h<3>**	AW27
mch_hlt_irq_h	AU25	**osc_clk_in_h**	BC21	**osc_clk_in_l**	BB22
perf_mon_h	AW29	**port_mode_h<0>**	AY20	**port_mode_h<1>**	BB20
pwr_fail_irq_h	AV26	**ref_clk_in_h**	AW25	**scache_set_h<0>**	C17
scache_set_h<1>	A17	**shared_h**	C23	**srom_clk_h**	BA19
srom_data_h	BC19	**srom_oe_l**	AW19	**srom_present_l**	AV20
st_clk_h	E05	**system_lock_flag_h**	G27	**sys_clk_out1_h**	AW23
sys_clk_out1_l	BB24	**sys_clk_out2_h**	AV24	**sys_clk_out2_l**	BC25
sys_mch_chk_irq_h	BA27	**sys_reset_l**	BC27	**tag_ctl_par_h**	F18
tag_data_h<20>	A05	**tag_data_h<21>**	E07	**tag_data_h<22>**	F08
tag_data_h<23>	C07	**tag_data_h<24>**	A07	**tag_data_h<25>**	E09
tag_data_h<26>	F10	**tag_data_h<27>**	C09	**tag_data_h<28>**	A09
tag_data_h<29>	E11	**tag_data_h<30>**	F12	**tag_data_h<31>**	C11
tag_data_h<32>	A11	**tag_data_h<33>**	E13	**tag_data_h<34>**	F14
tag_data_h<35>	C13	**tag_data_h<36>**	A13	**tag_data_h<37>**	B14
tag_data_h<38>	E15	**tag_data_par_h**	C15	**tag_dirty_h**	E17
tag_ram_oe_h	C21	**tag_ram_we_h**	A21	**tag_shared_h**	A15
tag_valid_h	F16	**tck_h**	AW17	**tdi_h**	BC17
tdo_h	BA17	**temp_sense**	AW15	**test_status_h<0>**	BA15
test_status_h<1>	AV16	**tms_h**	AV18	**trst_l**	BC15
victim_pending_h	E21	**spare_in<438>**	E39	**spare_io<250>**	AV28

(continued on next page)

Table 11–1 (Cont.) Alphabetic Signal Pin List

Signal	PGA Location
Vss—Metal planes 2[1] and 5[2]	A03, A41, AA07, AA37, AC07, AC37, AD04, AD40, AF02, AF42, AG07, AG37, AH04, AH40, AL07, AL37, AM04, AM40, AP02, AP42, AR07, AR37, AT04, AT40, AU09, AU13, AU17, AU31, AU35, AV02, AV22, AV42, AW21, AY04, AY08, AY12, AY16, AY22, AY24, AY28, AY32, AY36, AY40, B02, B06, B10, B18, B26, B34, B38, B42, BA01, BA21, BA43, BB02, BB06, BB10, BB18, BB26, BB34, BB38, BB42, BC03, BC41, C01, C43, D04, D08, D12, D16, D20, D24, D28, D32, D36, D40, F02, F42, G09, G13, G17, G31, G35, H04, H40, J07, J37, K02, K42, M04, M40, N07, N37, T04, T40, U07, U37, V02, V42, Y04, Y40
Vdd Metal planes 4 and 6	AB02, AB04, AB40, AB42, AE07, AE37, AF04, AF40, AH02, AH42, AJ07, AJ37, AK04, AK40, AM02, AM42, AN07, AN37, AP04, AP40, AT02, AT42, AU07, AU11, AU15, AU19, AU29, AU33, AU37, AV04, AV40, AY02, AY06, AY10, AY14, AY18, AY26, AY30, AY34, AY38, AY42, B04, B08, B12, B16, B22, B28, B32, B36, B40, BA03, BA05, BA39, BA41, BB04, BB08, BB12, BB16, BB28, BB32, BB36, BB40, BC23, C03, C05, C39, C41, D02, D06, D10, D14, D18, D22, D26, D30, D34, D38, D42, F04, F40, G11, G15, G19, G29, G33, G37, H02, H42, K04, K40, L07, L37, M02, M42, P04, P40, R07, R37, T02, T42, V04, V40, W07, W37

[1]Metal plane 2—Seal ring connection tied to **Vss**

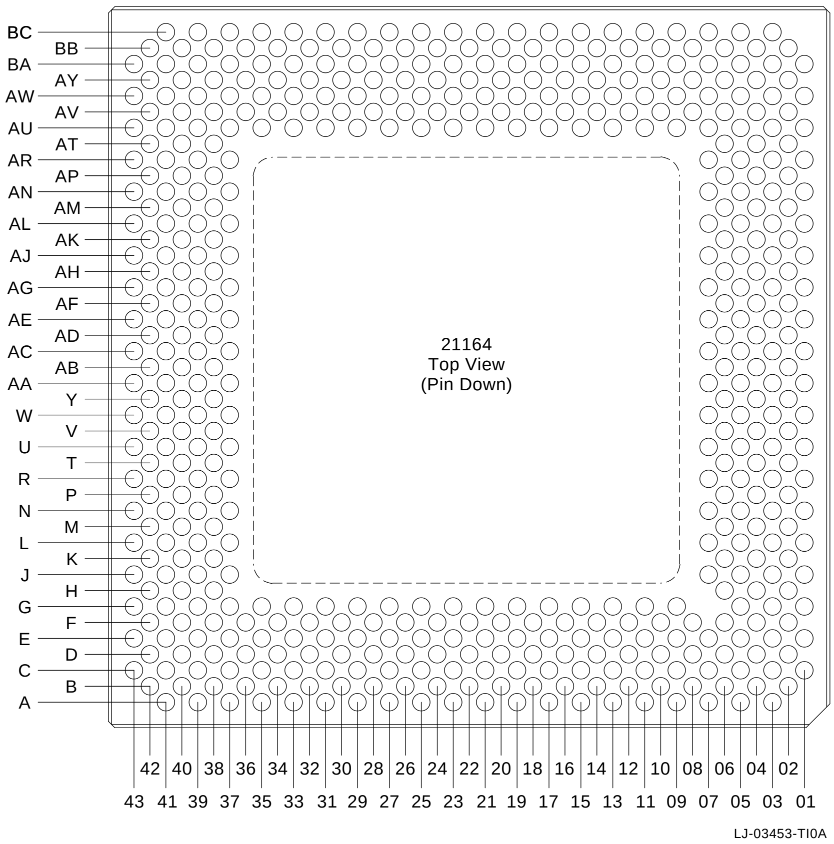
[2]Metal plane 5—Heat slug braze pad connections tied to **Vss**

11.2 Signal Descriptions and Pin Assignment

11.2.2 Pin Assignment

Figure 11–2 shows the 21164 pinout from the top view with pins facing down.

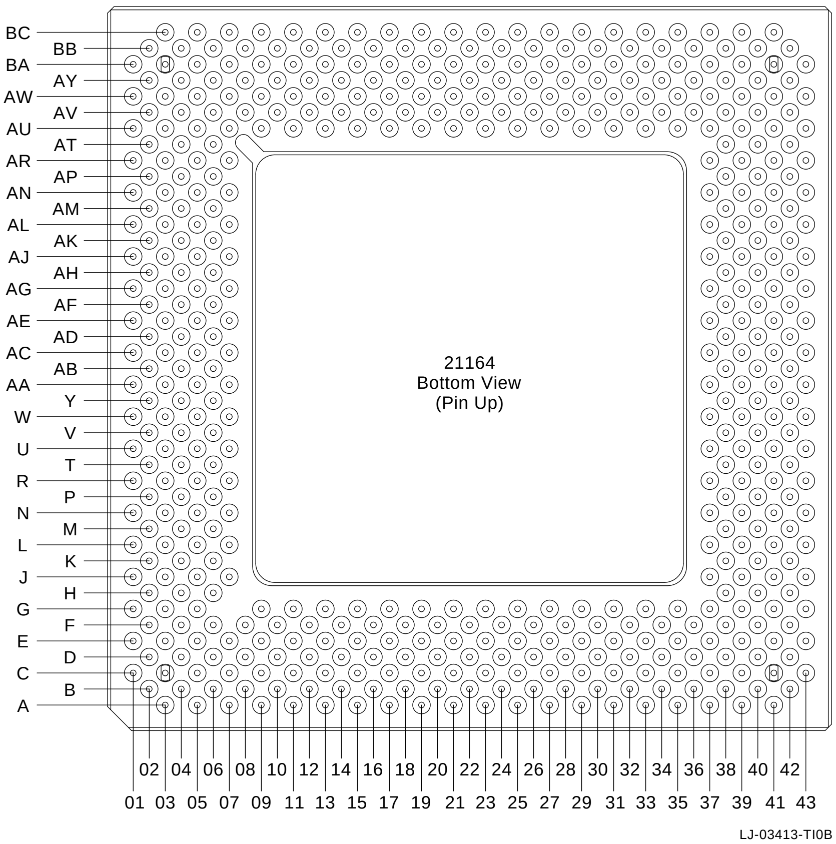
Figure 11–2 Alpha 21164 Top View (Pin Down)



11.2 Signal Descriptions and Pin Assignment

Figure 11–3 shows the 21164 pinout from the bottom view with pins facing up.

Figure 11–3 Alpha 21164 Bottom View (Pin Up)



12
Testability and Diagnostics

This chapter describes the 21164 user-oriented testability features. The 21164 also has several internal testability features that are implemented for factory use only. These features are beyond the scope of this document.

12.1 Test Port Pins

Table 12–1 summarizes the test port pins and their function.

Table 12–1 Alpha 21164 Test Port Pins

Pin Name	Type	Function
port_mode_h<1>	I	Must be false.
port_mode_h<0>	I	Must be false.
srom_present_l	I	Tied low if serial ROMs (SROMs) are present in system.
srom_data_h/Rx	I	Receives SROM or serial terminal data.
srom_clk_h/Tx	O	Supplies clock to SROMs or transmits serial terminal data.
srom_oe_l	O	SROM enable.
tdi_h	I	IEEE 1149.1 TDI port.
tdo_h	O	IEEE 1149.1 TDO port.
tms_h	I	IEEE 1149.1 TMS port.
tck_h	I	IEEE 1149.1 TCK port.
trst_l	I	IEEE 1149.1 optional TRST port.
test_status_h<0>	O	Indicates Icache BiSt status.
test_status_h<1>	O	Outputs an IPR-written value and timeout reset.

12.2 Test Interface

The 21164 test interface supports a serial ROM interface, a serial diagnostic terminal interface, and an IEEE 1149.1 test access port. These ports are available and set to normal test interface mode when **port_mode_h<1:0>**=00. Driving these pins to a value of anything other than 00 redefines all other test interface pins and invokes special factory test modes not covered in this document.

The SROM port is described in Section 7.4 and the serial terminal port is described in Section 7.5.

12.2.1 IEEE 1149.1 Test Access Port

Pins **tdi_h**, **tdo_h**, **tck_h**, **tms_h**, and **trst_l** constitute the IEEE 1149.1 test access port. This port accesses the 21164 chip's boundary scan register and chip tristate functions for board level manufacturing test. The port also allows access to factory manufacturing features not described in this document. The port is compliant with most requirements of IEEE 1149.1 test access port.

Compliance Enable Inputs

Table 12–2 shows the compliance enable inputs and the pattern that must be driven to those inputs in order to activate the 21164 IEEE 1149.1 circuits.

Table 12–2 Compliance Enable Inputs

Input	Compliance Enable Pattern
port_mode_h<1:0>	00
dc_ok_h	1

Exceptions to Compliance

The 21164 is compliant with IEEE Standard 1149.1–1993, with two exceptions. Both exceptions provide enhanced value to the user.

1. **trst_l** pin

   The optional **trst_l** pin has an internal pull-down, instead of a pull-up as required by IEEE 1149.1 (non-complied spec 3.6.1(b) in IEEE 1149.1–1993). The **trst_l** pull-down allows the chip to automatically force reset to the IEEE 1149.1 circuits in a system in which the IEEE 1149.1 port is unconnected. This may be considered a feature for most system designs that use IEEE 1149.1 circuits solely during module manufacturing.

Note

Digital recommends that the **trst_l** pin be driven low (asserted) when the JTAG (IEEE 1149.1) logic is not in use.

2. Coverage of oscillator differential input pins

   The two differential clock input pins, **osc_clk_in_h** and **osc_clk_in_l**, do not have any boundary scan cells associated with them (non-complied spec 10.4.1(b) in IEEE 1149.1–1993). Instead, there is an extra input BSR cell in the boundary scan register in bit position 255 (at pin **dc_ok_h**). This cell captures the output of a ''clock sniffer'' circuit. It captures a ''1'' when the oscillator is connected, and captures a ''0'' if the chip's oscillator connections are broken.

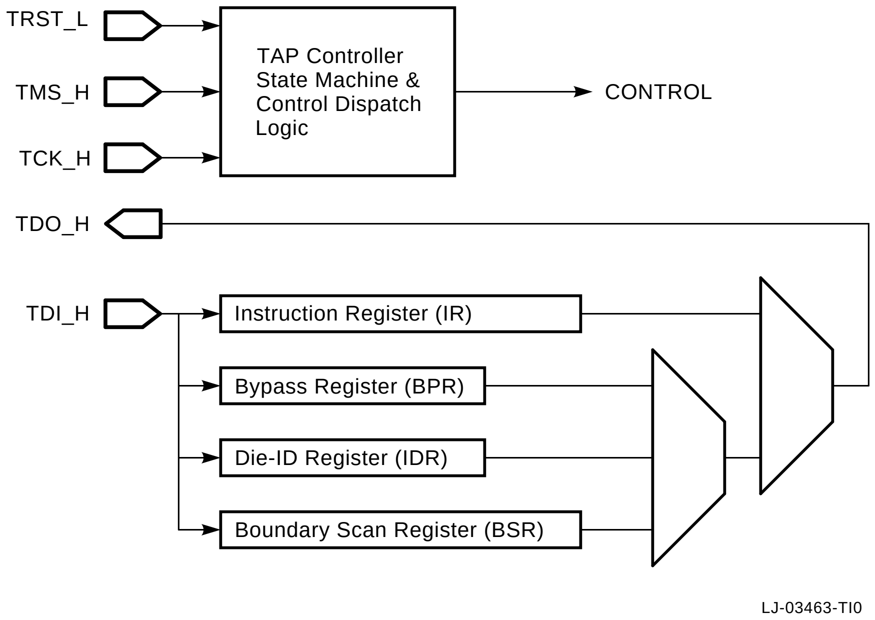
   This exception to the standard is made to permit a meaningful test of the oscillator input pins.

Refer to IEEE Standard 1149.1-1993 *A Test Access Port and Boundary Scan Architecture* for a full description of the specification.

Figure 12–1 shows the user-visible features from this port.

12.2 Test Interface

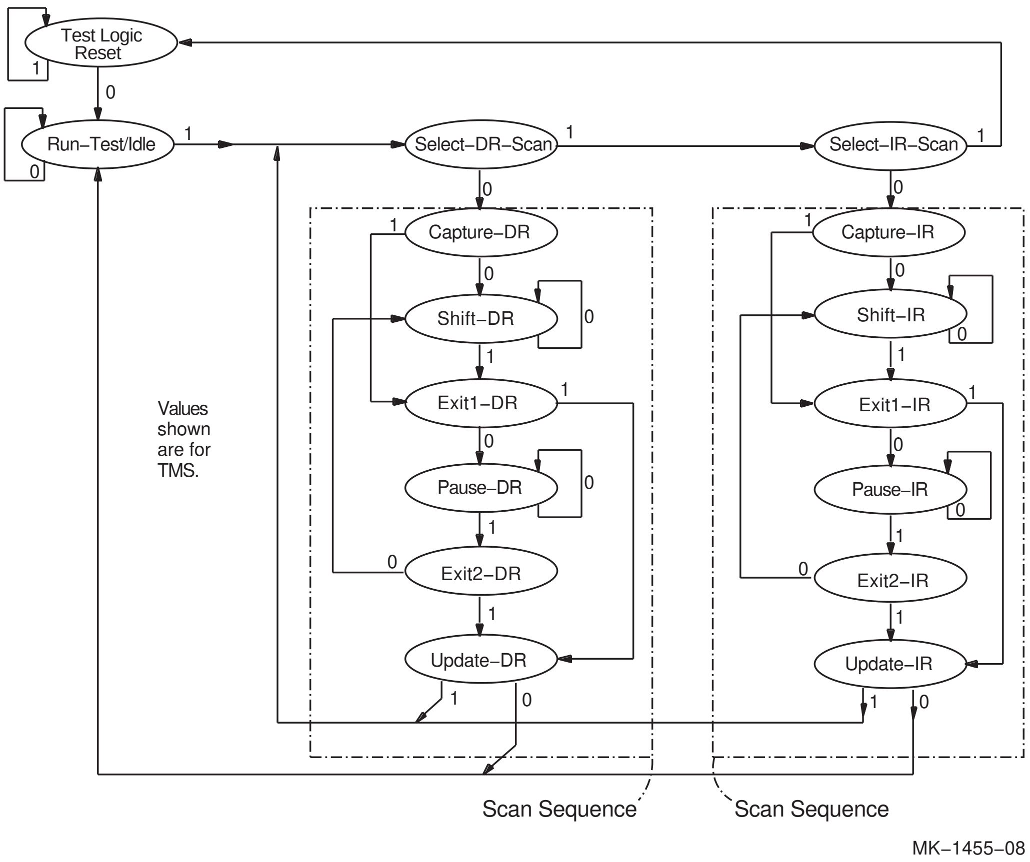
Figure 12–1 IEEE 1149.1 Test Access Port



TAP Controller

The TAP controller contains a state machine. It interprets IEEE 1149.1 protocols received on signal **tms_h** and generates appropriate clocks and control signals for the testability features under its jurisdiction. The state machine is shown in Figure 12–2

Figure 12–2 TAP Controller State Machine



Instruction Register

The 5-bit-wide instruction register (IR) supports IEEE 1149.1 mandated public instructions (EXTEST, SAMPLE, BYPASS, HIGHZ) and a number of optional instructions for public and private factory use. Table 12–3 summarizes the public instructions and their functions.

During the capture operation, the shift register stage of IR is loaded with the value 00001. This automatic load feature is useful for testing the integrity of the IEEE 1149.1 scan chain on the module.

Table 12–3 Instruction Register

IR<4:0>	Name	Scan Register Selected	Operation
00000	EXTEST	BSR	BSR drives pins. Interconnect test mode.
00010	SAMPLE/ PRELOAD	BSR	Preloads BSR.
00010	Private	BSR	Private.
00011	Private	BSR	Private.
00100	CLAMP	BPR	BSR drives pins.
00101	HIGHZ	BPR	Tristate all output and I/O pins.
00110	Private	IDR	Private.
00111	Private	IDR	Private.
01000 through 11110	Private	BPR	Private.
11111	BYPASS	BPR	Default.

Bypass Register

The bypass register is a 1-bit shift register. It provides a short single-bit scan path through the port (chip).

Boundary Scan Register

The 289-bit boundary scan register is accessed during SAMPLE, EXTEST, and CLAMP instructions. Refer to Section 12.3 for the organization of this register.

12.2.2 Test Status Pins

Two test status signal **test_status_h<1:0>** pins are used for extracting test status information from the chip. System reset drives both test status pins low. The default operation for **test_status_h<0>** is to output the BiSt results. The default operation for **test_status_h<1>** is to output the IPR-written value.

- During Icache BiSt Operation

  test_status_h<0> is forced high at the start of the Icache BiSt. If the Icache BiSt passes, the pin is deasserted at the end of the BiSt operation, otherwise it remains high.

- IPR read and write operations to test status pins

PALcode can write to the **test_status_h<1>** signal pin and can read the **test_status_h<0>** signal pin through hardware IPR access. Refer to Chapter 6.

- Timeout Reset

  The 21164 generates a timeout reset signal under two conditions:

  1. If an instruction is not retired within 1 billion cycles.
  2. If the system asserts **cfail_h** when **cack_h** is deasserted.

  In either of these conditions, the CPU signals the timeout reset event by outputting a 256 CPU cycle wide pulse on the **test_status_h<1>** pin. The pulse on **test_status_h<1>** pin is clocked by sysclk and therefore appears as an approximately 256 CPU cycle pulse that rises and falls on system clock rising edges.

12.3 Boundary Scan Register

The 21164 boundary scan register (BSR) is 289 bits long. Table 12–4 provides the boundary scan register organization. The BSR is connected between the **tdi_h** and **tdo_h** pins whenever an instruction selects it (Table 12–3). The scan register runs clockwise beginning at the upper left corner of the chip.

There are seven groups of bidirectional pins, each group controlled from a group control cell. Loading a value of ‘‘1’’ in the control cell tristates the output drivers and all bidirectional pins in the group are configured as input pins. The bidirectional pin groups are identified as groups gr_1 through gr_7 in the Control Group column in Table 12–4.

Information on Boundary Scan Description Language (BSDL) as it applies to the 21164 boundary scan register is available through your local Digital distributor (see Appendix E).

Notes

The following notes apply to Table 12–4:

- The direction of shift is from top to bottom, and from left to right.
- The bottom most signals appear first at the **tdo_h** pin when shifting.
- Given an arrayed signal of the form signal<a:b>, signal<b> appears at the **tdo_h** pin prior to signal<a>.

12.3 Boundary Scan Register

Table 12–4 Boundary Scan Register Organization

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
TR_ADL	Control	288	io_bcell	gr_1	Upper left corner.
addr_h<21:4>	B	287:270	io_bcell	gr_1	—
temp_sense	O	—	None	—	Analog pin.
test_status_h<1:0>	O	269:268	io_bcell	—	—
trst_l	I	—	None	—	—
tck_h	I	—	None	—	—
tms_h	I	—	None	—	—
tdo_h	O	—	None	—	—
tdi_h	I	—	None	—	—
srom_oe_l	O	267	io_bcell	—	—
srom_clk_h	O	266	io_bcell	—	—
srom_data_h	I	265	in_bcell	—	—
srom_present_l	I	264	in_bcell	—	—
port_mode_h<0:1>	I	—	None		Compliance enable pins.
clk_mode_h<0>	I	263	in_bcell	—	—
osc_clk_in_h,l	I	—	None	—	Analog pins.
clk_mode_h<1>	I	262	in_bcell	—	—
sys_clk_out1_h,l	O	261:260	io_bcell	—	—
sys_clk_out2_h,l	O	259:258	io_bcell	—	—
cpu_clk_out_h	O	—	none	—	For chip test.
ref_clk_in_h	I	257	in_bcell	—	—
sys_reset_l	I	256	in_bcell	—	—
dc_ok_h	I	—	None	—	Compliance enable pin.
OSC_SNIFFER_H	Internal	255	in_bcell	—	Captures 1 if osc is connected, otherwise captures 0.
sys_mch_chk_irq_h	I	254	in_bcell	—	—
pwr_fail_irq_h	I	253	in_bcell	—	—
mch_hlt_irq_h	I	252	in_bcell	—	—

(continued on next page)

Table 12–4 (Cont.) Boundary Scan Register Organization

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
irq_h<3:0>	I	251:248	in_bcell	—	—
SPARE_IO<250>	B	247	io_bcell	—	Tied off as input.
perf_mon_h	I	246	in_bcell	—	—
TR_ADR	Control	245	io_bcell	gr_2	—
addr_h<39:22>	B	244:227	io_bcell	gr_2	Upper right corner.
TR_DDR	Control	226	io_bcell	gr_3	—
data_h<63:0>	B	225:162	io_bcell	gr_3	—
data_check_h<0:7>	B	161:154	io_bcell	gr_3	—
int4_valid_h<1:0>	O	153:152	io_bcell	—	—
SPARE_IO<438>	—	—	None	—	Lower right corner, unpopulated.
index_h<25:4>	O	151:130	io_bcell	—	—
addr_res_h<2:0>	O	129:127	io_bcell	—	—
idle_bc_h	I	126	in_bcell	—	—
system_lock_flag_h	I	125	in_bcell	—	—
data_bus_req_h	I	124	in_bcell	—	—
cfail_h	I	123	in_bcell	—	—
fill_nocheck_h	I	122	in_bcell	—	—
fill_error_h	I	121	in_bcell	—	—
fill_id_h	I	120	in_bcell	—	—
fill_h	I	119	in_bcell	—	—
dack_h	I	118	in_bcell	—	—
addr_bus_req_h	I	117	in_bcell	—	—
cack_h	I	116	in_bcell	—	—
shared_h	I	115	in_bcell	—	—
data_ram_we_h	O	114	io_bcell	—	—
data_ram_oe_h	O	113	io_bcell	—	—
tag_ram_we_h	O	112	io_bcell	—	—

(continued on next page)

12.3 Boundary Scan Register

Table 12–4 (Cont.) Boundary Scan Register Organization

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
tag_ram_oe_h	O	111	io_bcell	—	—
victim_pending_h	O	110	io_bcell	—	—
TMIS1	Control	109	io_bcell	gr_4	—
addr_cmd_par_h	B	108	io_bcell	gr_4	—
cmd_h<0:3>	B	107:104	io_bcell	gr_4	—
scache_set_h<1:0>	O	103:102	io_bcell	—	—
TTAG1	Control	101	io_bcell	gr_5	—
tag_ctl_par_h	B	100	io_bcell	gr_5	—
tag_dirty_h	B	99	io_bcell	gr_5	—
tag_shared_h	B	98	io_bcell	gr_5	—
TTAG2	Control	97	io_bcell	gr_6	—
tag_data_par_h	B	96	io_bcell	gr_6	—
tag_valid_h	B	95	io_bcell	gr_6	—
tag_data_h<38:20>	B	94:76	io_bcell	gr_6	—
st_clk_h	O	75	io_bcell	—	Lower left corner.
int4_valid_h<2:3>	O	74:73	io_bcell	—	—
TR_DDL	Control	72	io_bcell	gr_7	—
data_check_h<15:8>	B	71:64	io_bcell	gr_7	—
data_h<64:127>	B	63:0	io_bcell	gr_7	—

A

Alpha Instruction Set

A.1 Alpha Instruction Summary

This appendix contains a summary of all Alpha architecture instructions. All values are in hexadecimal radix. Table A–1 describes the contents of the Format and Opcode columns that are in Table A–2.

Table A–1 Instruction Format and Opcode Notation

Instruction Format	Format Symbol	Opcode Notation	Meaning
Branch	Bra	oo	oo is the 6-bit opcode field.
Floating-point	F-P	oo.fff	oo is the 6-bit opcode field. fff is the 11-bit function code field.
Memory	Mem	oo	oo is the 6-bit opcode field.
Memory/ function code	Mfc	oo.ffff	oo is the 6-bit opcode field. ffff is the 16-bit function code in the displacement field.
Memory/ branch	Mbr	oo.h	oo is the 6-bit opcode field. h is the high-order 2 bits of the displacement field.
Operate	Opr	oo.ff	oo is the 6-bit opcode field. ff is the 7-bit function code field.
PALcode	Pcd	oo	oo is the 6-bit opcode field; the particular PALcode instruction is specified in the 26-bit function code field.

A.1 Alpha Instruction Summary

Qualifiers for operate instructions are shown in Table A–2. Qualifiers for IEEE and VAX floating-point instructions are shown in Tables A–5 and A–6, respectively.

Table A–2 Architecture Instructions

Mnemonic	Format	Opcode	Description
ADDF	F-P	15.080	Add F_floating
ADDG	F-P	15.0A0	Add G_floating
ADDL	Opr	10.00	Add longword
ADDL/V	Opr	10.40	Add longword
ADDQ	Opr	10.20	Add quadword
ADDQ/V	Opr	10.60	Add quadword
ADDS	F-P	16.080	Add S_floating
ADDT	F-P	16.0A0	Add T_floating
AND	Opr	11.00	Logical product
BEQ	Bra	39	Branch if $=$ zero
BGE	Bra	3E	Branch if $\geq$ zero
BGT	Bra	3F	Branch if $>$ zero
BIC	Opr	11.0	Bit clear
BIS	Opr	11.20	Logical sum
BLBC	Bra	38	Branch if low bit clear
BLBS	Bra	3C	Branch if low bit set
BLE	Bra	3B	Branch if $\leq$ zero
BLT	Bra	3A	Branch if $<$ zero
BNE	Bra	3D	Branch if $\neq$ zero
BR	Bra	30	Unconditional branch
BSR	Mbr	34	Branch to subroutine
CALL_PAL	Pcd	00	Trap to PALcode
CMOVEQ	Opr	11.24	CMOVE if $=$ zero
CMOVGE	Opr	11.46	CMOVE if $\geq$ zero
CMOVGT	Opr	11.66	CMOVE if $>$ zero
CMOVLBC	Opr	11.16	CMOVE if low bit clear
CMOVLBS	Opr	11.14	CMOVE if low bit set
CMOVLE	Opr	11.64	CMOVE if $\leq$ zero
CMOVLT	Opr	11.44	CMOVE if $<$ zero
CMOVNE	Opr	11.26	CMOVE if $\neq$ zero
CMPBGE	Opr	10.0F	Compare byte
CMPEQ	Opr	10.2D	Compare signed quadword equal
CMPGEQ	F-P	15.0A5	Compare G_floating equal
CMPGLE	F-P	15.0A7	Compare G_floating less than or equal

(continued on next page)

Table A–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
CMPGLT	F-P	15.0A6	Compare G_floating less than
CMPLE	Opr	10.6D	Compare signed quadword less than or equal
CMPLT	Opr	10.4D	Compare signed quadword less than
CMPTEQ	F-P	16.0A5	Compare T_floating equal
CMPTLE	F-P	16.0A7	Compare T_floating less than or equal
CMPTLT	F-P	16.0A6	Compare T_floating less than
CMPTUN	F-P	16.0A4	Compare T_floating unordered
CMPULE	Opr	10.3D	Compare unsigned quadword less than or equal
CMPULT	Opr	10.1D	Compare unsigned quadword less than
CPYS	F-P	17.020	Copy sign
CPYSE	F-P	17.022	Copy sign and exponent
CPYSN	F-P	17.021	Copy sign negate
CVTDG	F-P	15.09E	Convert D_floating to G_floating
CVTGD	F-P	15.0AD	Convert G_floating to D_floating
CVTGF	F-P	15.0AC	Convert G_floating to F_floating
CVTGQ	F-P	15.0AF	Convert G_floating to quadword
CVTLQ	F-P	17.010	Convert longword to quadword
CVTQF	F-P	15.0BC	Convert quadword to F_floating
CVTQG	F-P	15.0BE	Convert quadword to G_floating
CVTQL	F-P	17.030	Convert quadword to longword
CVTQL/SV	F-P	17.530	Convert quadword to longword
CVTQL/V	F-P	17.130	Convert quadword to longword
CVTQS	F-P	16.0BC	Convert quadword to S_floating
CVTQT	F-P	16.0BE	Convert quadword to T_floating
CVTST	F-P	16.2AC	Convert S_floating to T_floating
CVTTQ	F-P	16.0AF	Convert T_floating to quadword
CVTTS	F-P	16.0AC	Convert T_floating to S_floating
DIVF	F-P	15.083	Divide F_floating
DIVG	F-P	15.0A3	Divide G_floating
DIVS	F-P	16.083	Divide S_floating
DIVT	F-P	16.0A3	Divide T_floating
EQV	Opr	11.48	Logical equivalence
EXCB	Mfc	18.0400	Exception barrier
EXTBL	Opr	12.06	Extract byte low
EXTLH	Opr	12.6A	Extract longword high

(continued on next page)

A.1 Alpha Instruction Summary

Table A–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
EXTLL	Opr	12.26	Extract longword low
EXTQH	Opr	12.7A	Extract quadword high
EXTQL	Opr	12.36	Extract quadword low
EXTWH	Opr	12.5A	Extract word high
EXTWL	Opr	12.16	Extract word low
FBEQ	Bra	31	Floating branch if = zero
FBGE	Bra	36	Floating branch if $\geq$ zero
FBGT	Bra	37	Floating branch if > zero
FBLE	Bra	33	Floating branch if $\leq$ zero
FBLT	Bra	32	Floating branch if < zero
FBNE	Bra	35	Floating branch if $\neq$ zero
FCMOVEQ	F-P	17.02A	FCMOVE if = zero
FCMOVGE	F-P	17.02D	FCMOVE if $\geq$ zero
FCMOVGT	F-P	17.02F	FCMOVE if > zero
FCMOVLE	F-P	17.02E	FCMOVE if $\leq$ zero
FCMOVLT	F-P	17.02C	FCMOVE if < zero
FCMOVNE	F-P	17.02B	FCMOVE if $\neq$ zero
FETCH	Mfc	18.80	Prefetch data
FETCH_M	Mfc	18.A0	Prefetch data, modify intent
INSBL	Opr	12.0B	Insert byte low
INSLH	Opr	12.67	Insert longword high
INSLL	Opr	12.2B	Insert longword low
INSQH	Opr	12.77	Insert quadword high
INSQL	Opr	12.3B	Insert quadword low
INSWH	Opr	12.57	Insert word high
INSWL	Opr	12.1B	Insert word low
JMP	Mbr	1A.0	Jump
JSR	Mbr	1A.1	Jump to subroutine
JSR_COROUTINE	Mbr	1A.3	Jump to subroutine return
LDA	Mem	08	Load address
LDAH	Mem	09	Load address high
LDF	Mem	20	Load F_floating
LDG	Mem	21	Load G_floating
LDL	Mem	28	Load sign-extended longword
LDL_L	Mem	2A	Load sign-extended longword locked
LDQ	Mem	29	Load quadword
LDQ_L	Mem	2B	Load quadword locked
LDQ_U	Mem	0B	Load unaligned quadword

(continued on next page)

Table A–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
LDS	Mem	22	Load S_floating
LDT	Mem	23	Load T_floating
MB	Mfc	18.4000	Memory barrier
MF_FPCR	F-P	17.025	Move from floating-point control register
MSKBL	Opr	12.02	Mask byte low
MSKLH	Opr	12.62	Mask longword high
MSKLL	Opr	12.22	Mask longword low
MSKQH	Opr	12.72	Mask quadword high
MSKQL	Opr	12.32	Mask quadword low
MSKWH	Opr	12.52	Mask word high
MSKWL	Opr	12.12	Mask word low
MT_FPCR	F-P	17.024	Move to floating-point control register
MULF	F-P	15.082	Multiply F_floating
MULG	F-P	15.0A2	Multiply G_floating
MULL	Opr	13.00	Multiply longword
MULL/V	Opr	13.40	Multiply longword
MULQ	Opr	13.20	Multiply quadword
MULQ/V	Opr	13.60	Multiply quadword
MULS	F-P	16.082	Multiply S_floating
MULT	F-P	16.0A2	Multiply T_floating
ORNOT	Opr	11.28	Logical sum with complement
RC	Mfc	18.E0	Read and clear
RET	Mbr	1A.2	Return from subroutine
RPCC	Mfc	18.C0	Read process cycle counter
RS	Mfc	18.F000	Read and set
S4ADDL	Opr	10.02	Scaled add longword by 4
S4ADDQ	Opr	10.22	Scaled add quadword by 4
S4SUBL	Opr	10.0B	Scaled subtract longword by 4
S4SUBQ	Opr	10.2B	Scaled subtract quadword by 4
S8ADDL	Opr	10.12	Scaled add longword by 8
S8ADDQ	Opr	10.32	Scaled add quadword by 8
S8SUBL	Opr	10.1B	Scaled subtract longword by 8
S8SUBQ	Opr	10.3B	Scaled subtract quadword by 8
SLL	Opr	12.39	Shift left logical
SRA	Opr	12.3C	Shift right arithmetic
SRL	Opr	12.34	Shift right logical
STF	Mem	24	Store F_floating

(continued on next page)

A.1 Alpha Instruction Summary

Table A–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
STG	Mem	25	Store G_floating
STS	Mem	26	Store S_floating
STL	Mem	2C	Store longword
STL_C	Mem	2E	Store longword conditional
STQ	Mem	2D	Store quadword
STQ_C	Mem	2F	Store quadword conditional
STQ_U	Mem	0F	Store unaligned quadword
STT	Mem	27	Store T_floating
SUBF	F-P	15.081	Subtract F_floating
SUBG	F-P	15.0A1	Subtract G_floating
SUBL	Opr	10.09	Subtract longword
SUBL/V		10.49	
SUBQ	Opr	10.29	Subtract quadword
SUBQ/V		10.69	
SUBS	F-P	16.081	Subtract S_floating
SUBT	F-P	16.0A1	Subtract T_floating
TRAPB	Mfc	18.00	Trap barrier
UMULH	Opr	13.30	Unsigned multiply quadword high
WMB	Mfc	18.44	Write memory barrier
XOR	Opr	11.40	Logical difference
ZAP	Opr	12.30	Zero bytes
ZAPNOT	Opr	12.31	Zero bytes not

A.1.1 Opcodes Reserved for Digital

Table A–3 lists opcodes reserved for Digital.

Table A–3 Opcodes Reserved for Digital

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
OPC01	01	OPC05	05	OPC0B	0B
OPC02	02	OPC06	06	OPC0C	0C
OPC03	03	OPC07	07	OPC0D	0D
OPC04	04	OPC0A	0A	OPC14	14

A.1.2 Opcodes Reserved for PALcode

Table A–4 lists the 21164-specific instructions. For more information, refer to Section 6.6.

Table A–4 Opcodes Reserved for PALcode

21164 Mnemonic	Opcode	Architecture Mnemonic	Function
HW_LD	1B	PAL1B	Performs Dstream load instructions.
HW_ST	1F	PAL1F	Performs Dstream store instructions.
HW_REI	1E	PAL1E	Returns instruction flow to the program counter (PC) pointed to by EXC_ADDR internal processor register (IPR).
HW_MFPR	19	PAL19	Accesses the Ibox, Mbox, and Dcache IPRs.
HW_MTPR	1D	PAL1D	Accesses the Ibox, Mbox, and Dcache IPRs.

A.2 IEEE Floating-Point Instructions

Table A–5 lists the hexadecimal value of the 11-bit function code field for the IEEE floating-point instructions, with and without qualifiers. The opcode for these instructions is 16_{16}.

Table A–5 IEEE Floating-Point Instruction Function Codes

Mnemonic	None	/C	/M	/D	/U	/UC	/UM	/UD
ADDS	080	000	040	0C0	180	100	140	1C0
ADDT	0A0	020	060	0E0	1A0	120	160	1E0
CMPTEQ	0A5							
CMPTLT	0A6							
CMPTLE	0A7							
CMPTUN	0A4							
CVTQS	0BC	03C	07C	0FC				
CVTQT	0BE	03E	07E	0FE				

(continued on next page)

A.2 IEEE Floating-Point Instructions

Table A–5 (Cont.) IEEE Floating-Point Instruction Function Codes

Mnemonic	None	/C	/M	/D	/U	/UC	/UM	/UD
CVTTS	0AC	02C	06C	0EC	1AC	12C	16C	1EC
DIVS	083	003	043	0C3	183	103	143	1C3
DIVT	0A3	023	063	0E3	1A3	123	163	1E3
MULS	082	002	042	0C2	182	102	142	1C2
MULT	0A2	022	062	0E2	1A2	122	162	1E2
SUBS	081	001	041	0C1	181	101	141	1C1
SUBT	0A1	021	061	0E1	1A1	121	161	1E1

Mnemonic	/SU	/SUC	/SUM	/SUD	/SUI	/SUIC	/SUIM	/SUID
ADDS	580	500	540	5C0	780	700	740	7C0
ADDT	5A0	520	560	5E0	7A0	720	760	7E0
CMPTEQ	5A5							
CMPTLT	5A6							
CMPTLE	5A7							
CMPTUN	5A4							
CVTQS					7BC	73C	77C	7FC
CVTQT					7BE	73E	77E	7FE
CVTTS	5AC	52C	56C	5EC	7AC	72C	76C	7EC
DIVS	583	503	543	5C3	783	703	743	7C3
DIVT	5A3	523	563	5E3	7A3	723	763	7E3
MULS	582	502	542	5C2	782	702	742	7C2
MULT	5A2	522	562	5E2	7A2	722	762	7E2
SUBS	581	501	541	5C1	781	701	741	7C1
SUBT	5A1	521	561	5E1	7A1	721	761	7E1

Mnemonic	None	/S
CVTST	2AC	6AC

Mnemonic	None	/C	/V	/VC	/SV	/SVC	/SVI	/SVIC
CVTTQ	0AF	02F	1AF	12F	5AF	52F	7AF	72F

Mnemonic	D	/VD	/SVD	/SVID	/M	/VM	/SVM	/SVIM
CVTTQ	0EF	1EF	5EF	7EF	06F	16F	56F	76F

Programming Note

Because underflow cannot occur for CMPT*xx*, there is no difference in function or performance between CMPT*xx*/S and CMPT*xx*/SU. It is intended that software generate CMPT*xx*/SU in place of CMPT*xx*/S.

In the same manner, CVTQS and CVTQT can take an inexact result trap, but not an underflow. Because there is no encoding for a CVTQ*x*/SI instruction, it is intended that software generate CVTQ*x*/SUI in place of CVTQ*x*/SI.

A.3 VAX Floating-Point Instructions

Table A–6 lists the hexadecimal value of the 11-bit function code field for the VAX floating-point instructions. The opcode for these instructions is 15_{16}.

Table A–6 VAX Floating-Point Instruction Function Codes

Mnemonic	None	/C	/U	/UC	/S	/SC	/SU	/SUC
ADDF	080	000	180	100	480	400	580	500
CVTDG	09E	01E	19E	11E	49E	41E	59E	51E
ADDG	0A0	020	1A0	120	4A0	420	5A0	520
CMPGEQ	0A5				4A5			
CMPGLT	0A6				4A6			
CMPGLE	0A7				4A7			
CVTGF	0AC	02C	1AC	12C	4AC	42C	5AC	52C
CVTGD	0AD	02D	1AD	12D	4AD	42D	5AD	52D
CVTQF	0BC	03C						
CVTQG	0BE	03E						
DIVF	083	003	183	103	483	403	583	503
DIVG	0A3	023	1A3	123	4A3	423	5A3	523
MULF	082	002	182	102	482	402	582	502
MULG	0A2	022	1A2	122	4A2	422	5A2	522
SUBF	081	001	181	101	481	401	581	501
SUBG	0A1	021	1A1	121	4A1	421	5A1	521

Mnemonic	None	/C	/V	/VC	/S	/SC	/SV	/SVC
CVTGQ	0AF	02F	1AF	12F	4AF	42F	5AF	52F

A.4 Opcode Summary

Table A–7 lists all Alpha opcodes from 00 (CALL_PAL) through 3F (BGT). In the table, the column headings that appear over the instructions have a granularity of 8_{16}. The rows beneath the Offset column supply the individual hexadecimal number to resolve that granularity.

If an instruction column has a 0 in the right (low) hexadecimal digit, replace that 0 with the number to the left of the backslash in the Offset column on the instruction's row. If an instruction column has an 8 in the right (low) hexadecimal digit, replace that 8 with the number to the right of the backslash in the Offset column.

For example, the third row (2/A) under the 10_{16} column contains the symbol INTS*, representing the all-integer shift instructions. The opcode for those instructions would then be 12_{16} because the 0 in 10 is replaced by the 2 in the Offset column. Likewise, the third row under the 18_{16} column contains the symbol JSR*, representing all jump instructions. The opcode for those instructions is 1A because the 8 in the heading is replaced by the number to the right of the backslash in the Offset column.

The instruction format is listed under the instruction symbol.

A.4 Opcode Summary

Table A–7 Opcode Summary

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

Symbol	**Meaning**
FLTI*	IEEE floating-point instruction opcodes
FLTL*	Floating-point operate instruction opcodes
FLTV*	VAX floating-point instruction opcodes
INTA*	Integer arithmetic instruction opcodes
INTL*	Integer logical instruction opcodes
INTM*	Integer multiply instruction opcodes
INTS*	Integer shift instruction opcodes
JSR*	Jump instruction opcodes
MISC*	Miscellaneous instruction opcodes
PAL*	PALcode instruction (CALL_PAL) opcodes
\PAL\	Reserved for PALcode
Res	Reserved for Digital

A.5 Required PALcode Function Codes

The opcodes listed in Table A–8 are required for all Alpha implementations. The notation used is oo.ffff, where oo is the hexadecimal 6-bit opcode and ffff is the hexadecimal 26-bit function code.

Table A–8 Required PALcode Function Codes

Mnemonic	Type	Function Code
DRAINA	Privileged	00.0002
HALT	Privileged	00.0000
IMB	Unprivileged	00.0086

A.6 Alpha 21164 Microprocessor IEEE Floating-Point Conformance

The 21164 supports the IEEE floating-point operations as defined by the Alpha architecture. Support for a complete implementation of the IEEE *Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Standard 754 1985) is provided by a combination of hardware and software as described in the *Alpha Architecture Reference Manual*.

Additional information about writing code to support precise exception handling (necessary for complete conformance to the standard) is in the *Alpha Architecture Reference Manual*.

The following information is specific to the 21164:

- Invalid operation (INV)

  The invalid operation trap is always enabled. If the trap occurs, then the destination register is UNPREDICTABLE. This exception is signaled if any VAX architecture operand is non-finite (reserved operand or dirty zero) and the operation can take an exception (that is, certain instructions, such as CPYS, never take an exception). This exception is signaled if any IEEE operand is non-finite (NAN, INF, denorm) and the operation can take an exception. This trap is also signaled for an IEEE format divide of +/– 0 divided by +/– 0. If the exception occurs, then FPCR<INV> is set and the trap is signaled to the Ibox.

A.6 Alpha 21164 Microprocessor IEEE Floating-Point Conformance

- Divide-by-zero (DZE)

  The divide-by-zero trap is always enabled. If the trap occurs, then the destination register is UNPREDICTABLE. For VAX architecture format, this exception is signaled whenever the numerator is valid and the denominator is zero. For IEEE format, this exception is signaled whenever the numerator is valid and non-zero, with a denominator of +/– 0. If the exception occurs, then FPCR<DZE> is set and the trap is signaled to the Ibox.

  For IEEE format divides, 0/0 signals INV, not DZE.

- Floating overflow (OVF)

  The floating overflow trap is always enabled. If the trap occurs, then the destination register is UNPREDICTABLE. The exception is signaled if the rounded result exceeds in magnitude the largest finite number, which can be represented by the destination format. This applies only to operations whose destination is a floating-point data type. If the exception occurs, then FPCR<OVF> is set and the trap is signaled to the Ibox.

- Underflow (UNF)

  The underflow trap can be disabled. If underflow occurs, then the destination register is forced to a true zero, consisting of a full 64 bits of zero. This is done even if the proper IEEE result would have been –0. The exception is signaled if the rounded result is smaller in magnitude than the smallest finite number that can be represented by the destination format. If the exception occurs, then FPCR<UNF> is set. If the trap is enabled, then the trap is signaled to the Ibox. The 21164 never produces a denormal number; underflow occurs instead.

- Inexact (INE)

  The inexact trap can be disabled. The destination register always contains the properly rounded result, whether the trap is enabled. The exception is signaled if the rounded result is different from what would have been produced if infinite precision (infinitely wide data) were available. For floating-point results, this requires both an infinite precision exponent and fraction. For integer results, this requires an infinite precision integer and an integral result. If the exception occurs, then FPCR<INE> is set. If the trap is enabled, then the trap is signaled to the Ibox.

  The IEEE-754 specification allows INE to occur concurrently with either OVF or UNF. Whenever OVF is signaled (if the inexact trap is enabled), INE is also signaled. Whenever UNF is signaled (if the inexact trap is enabled), INE is also signaled. The inexact trap also occurs concurrently with integer overflow. All valid opcodes that enable INE also enable both overflow and underflow.

  If a CVTQL results in an integer overflow (IOV), then FPCR<INE> is automatically set. (The INE trap is never signaled to the Ibox because there is no CVTQL opcode that enables the inexact trap.)

- Integer overflow (IOV)

  The integer overflow trap can be disabled. The destination register always contains the low-order bits (<64> or <32>) of the true result (not the truncated bits). Integer overflow can occur with CVTTQ, CVTGQ, or CVTQL. In conversions from floating to quadword integer or longword integer, an integer overflow occurs if the rounded result is outside the range -2^{63} ..2^{63-1}. In conversions from quadword integer to longword integer, an integer overflow occurs if the result is outside the range -2^{31} ..2^{31-1}. If the exception occurs, then the appropriate bit in the FPCR is set. If the trap is enabled, then the trap is signaled to the Ibox.

- Software completion (SWC)

  The software completion signal is not recorded in the FPCR. The state of this signal is always sent to the Ibox. If the Ibox detects the assertion of any of the listed exceptions concurrent with the assertion of the SWC signal, then it sets EXC_SUM<SWC>.

Input exceptions always take priority over output exceptions. If both exception types occur, then only the input exception is recorded in the FPCR and only the input exception is signaled to the Ibox.

B

Alpha 21164 Microprocessor Specifications

Table B–1 lists specifications for the 21164.

Table B–1 Alpha 21164 Microprocessor Specifications

Feature	Description
Cycle time range	4.4 ns (227 MHz) to 3.3 ns (300 MHz).
Process technology	0.5-micron CMOS.
Transistor count	9.3 million.
Die size	664×732 mils.
Package	499-pin IPGA (interstitial pin grid array).
Number of signal pins	292.
Typical worst case power @**Vdd** = 3.3 V	46 W @ 3.75 ns cycle time (266 MHz)[1], 51 W @ 3.33 ns cycle time (300 MHz)[1].
Power supply	3.3 V dc.
Clocking input	Two times the internal clock speed (for example, 571.4 MHz at a 3.5-ns cycle time).
Virtual address size	43 bits.
Physical address size	40 bits.
Page size	8K bytes.
Issue rate	2 integer instructions and 2 floating-point instructions per cycle.
Integer instruction pipeline	7 stage.
Floating instruction pipeline	9 stage.
Onchip L1 Dcache	8K-byte, physical, direct-mapped, write-through, 32-byte block, 32-byte fill.
Onchip L1 Icache	8K-byte, virtual, direct-mapped, 32-byte block, 32-byte fill, 128 address space numbers (ASNs) (MAX_ASN=127).
Onchip L2 Scache	96K-byte, physical, 3-way set-associative, write-back, 32- or 64-byte block, 32- or 64-byte fill.
Onchip data translation buffer	64-entry, fully associative, not-last-used replacement, 8K pages, 128 ASNs (MAX_ASN=127), full granularity hint support.
Onchip instruction translation buffer	48-entry, fully associative, not-last-used replacement, 128 ASNs (MAX_ASN=127), full granularity hint support.
Floating-point unit	Onchip FPU supports both IEEE and Digital floating point.
Bus	Separate data and address bus, 128-bit/64-bit data bus.
Serial ROM interface	Allows microprocessor to access a serial ROM.

[1]Power consumption scales linearly with frequency over the frequency range 225 MHz to 300 MHz.

C

Serial Icache Load Predecode Values

The following C code calculates the predecode values of a serial Icache load. A software tool called the SROM Packer converts a binary image into a format suitable for Icache serial loading. This tool is available from Digital.

```
#include <stdio.h>

/* fillmap [0 - 127] maps data 127:0, etc. */
/* fillmap[n] is bit position in output vector. */
/*  bit 0 of this vector is first-in; bit 199 is last */

const int dfillmap [128] = {
                                      /* data 0:127 -- fillmap[0:127]*/
  42,44,46,48,50,52,54,56,            /* 0:7 */
  58,60,62,64,66,68,70,72,            /* 8:15 */
  74,76,78,80,82,84,86,88,            /* 16:23 */
  90,92,94,96,98,100,102,104,         /* 24:31 */
  43,45,47,49,51,53,55,57,            /* 32:39 */
  59,61,63,65,67,69,71,73,            /* 40:47 */
  75,77,79,81,83,85,87,89,            /* 48:55 */
  91,93,95,97,99,101,103,105,         /* 56:63 */
  128,130,132,134,136,138,140,142,    /* 64:71 */
  144,146,148,150,152,154,156,158,    /* 72:79 */
  160,162,164,166,168,170,172,174,    /* 80:87 */
  176,178,180,182,184,186,188,190,    /* 88:95 */
  129,131,133,135,137,139,141,143,    /* 96:103 */
  145,147,149,151,153,155,157,159,    /* 104:111 */
  161,163,165,167,169,171,173,175,    /* 112:119 */
  177,179,181,183,185,187,189,191     /* 120:127 */
  };

const int BHTfillmap[8] = {           /* BHT vector 0:7 -- BHTfillmap[0:7] */
  199,198,197,196,195,194,193,192     /* 0:7 */
  };

const int predfillmap[20] = {         /* predecodes 0:19 -- predfillmap[0:19] */
  106,108,110,112,114,                /* 0:4 */
  107,109,111,113,115,                /* 5:9 */
  118,120,122,124,126,                /* 10:14 */
  119,121,123,125,127                 /* 15:19 */
  };

const int octawpfillmap =             /* octaword parity */
  117;
```

```
const int predpfillmap =              /* predecode parity */
  116;

const int tagfillmap[30] = {          /* tag bits 13:42 -- tagfillmap[0:29] */
  29,28,27,26,25,24,23,22,21,20,      /* 13:22 */
  19,18,17,16,15,14,13,12,11,10,      /* 23:32 */
  09,08,07,06,05,04,03,02,01,00       /* 33:42 */
  };

const int asnfillmap[7] = {           /* asn 0:6 -- asnfillmap[0:6] */
  37,36,35,34,33,32,31                /* 0:6 */
  };

const int asmfillmap =                /* asm -- asmfillmap */
  30;

const int tagphysfillmap =            /* tagphysical address -- tagphysfillmap */
  38;

const int tagvalfillmap[2] = {        /* tag valid bits 0:1 -- tagvalfillmap */
  40,39                               /* 0:1 */
  };

const int tagparfillmap =             /* tag parity -- tagparfillmap */
  41;


main(argc, argv)
int argc;
char *argv[];
{
  int i,j,k,t;
  int status,instatus, instr_count;
  char filename[256],ofilename[256],hfilename[256];
  char *charptr;
  int instr[4], outvector[7];
  FILE* infile, outfile, hexfile;
  int base, asm, asn, tag, predecodes,owparity,pdparity,tparity,
      tvalids,tphysical,bhtvector, offset, chksum;

  strcpy (filename ,"loadfile.exe");
  strcpy(ofilename,"loadfile.srom");
  base = 0;
  tag = 0;
  asn = 0;
  asm = 1;
  tphysical= 1;
  bhtvector = 0;
  offset = 0;

  if (argc > 1)
    strcpy(filename, argv[1]);

  if (argc > 2)
    strcpy(ofilename, argv[2]);

  if (argc > 3)
    {
      base = strtol(argv[3],NULL,16) & (0xffffffff << 13);
      tag = base >> 13;
    }
```

```
 if (argc > 4)
   asn = strtol(argv[4],NULL,16) & 0x7f;

 if (argc > 5)
   asm = strtol(argv[5],NULL,16) & 1;

 if (argc > 6)
   tphysical = strtol(argv[6],NULL,16) & 1;

 if (argc > 7)
   bhtvector = strtol(argv[7],NULL,16) & 0xff;

 if (NULL == (infile = fopen(filename,"rb")))
   {
     printf("input file open error: %s\n", filename);
     exit(0);
   }

 if (NULL == (outfile = fopen(ofilename, "wb")))
   {
     printf("binary output file open error: %s\n", ofilename);
     exit(0);
   }

 strcpy(hfilename,ofilename);
 charptr = strpbrk(hfilename,".;");
 if (charptr != NULL) *charptr = 0;
 strcat(hfilename,".hex");

 if (NULL == (hexfile = fopen(hfilename, "w")))
   {
     printf("hex output file open error: %s\n", hfilename);
     exit(0);
   }

 fprintf(hexfile,":020000020000FC\n");

 tparity = eparity(tag) ^ eparity(tphysical) ^ eparity(asn);
 tvalids = 3;
 instatus = 0;
 instr_count = 0;

 for (i=0; i<512; i++)
   {
     for (j=0;j<4;j++) instr[j] = 0;
     for (j=0;j<7;j++) outvector[j]=0;


     if (instatus == 0)
{
  if (16 > (status = fread(&instr[0],1,16,infile)))
    instatus = 1;
  instr_count += status/4;
}
```

```
    predecodes=0;
    owparity = 0;
    for (j=0;j<4;j++)
{
  predecodes |= (4 ^ instrpredecode(instr[j])) << (j*5);
  /* invert bit 2 to match fill scan chain attribute */
  owparity ^= eparity(instr[j]);
}

    pdparity = eparity(predecodes);

    /* bhtvector */
    for (j=0;j<8;j++)
{
  t = BHTfillmap[j];
  outvector[t>>5] |= ((bhtvector >> j) & 1) << (t&0x1f);
}

    /* instructions */
    for (k=0;k<4;k++)
{
  for (j=0;j<32;j++)
    {
     t = dfillmap[j+k*32];
     outvector[t>>5] |= ((instr[k] >> j) & 1) << (t&0x1f);
    }
}

    /* predecodes */
    for (j=0;j<20;j++)
{
  t = predfillmap[j];
  outvector[t>>5] |= ((predecodes >> j) & 1) << (t&0x1f);
}

    /* owparity */
    outvector[octawpfillmap>>5] |= owparity << (octawpfillmap&0x1f);

    /* pdparity */
    outvector[predpfillmap>>5] |= pdparity << (predpfillmap&0x1f);

    /* tparity */
    outvector[tagparfillmap>>5] |= tparity << (tagparfillmap&0x1f);

    /* tvalids */
    for (j=0;j<2;j++)
{
  t =tagvalfillmap[j];
  outvector[t>>5] |= ((tvalids >> j) & 1) << (t&0x1f);
}

    /* tphysical */
    outvector[tagphysfillmap>>5] |=  tphysical << (tagphysfillmap&0x1f);

    /* asn */
    for (j=0;j<7;j++)
{
  t = asnfillmap[j];
  outvector[t>>5] |= ((asn >> j) & 1) << (t&0x1f);
}
```

```
      /* asm */
      outvector[asmfillmap>>5] |=  asm << (asmfillmap&0x1f);

      /* tag */
      for (j=0;j<30;j++)
 {
   t = tagfillmap[j];
   outvector[t>>5] |= ((tag >> j) & 1) << (t&0x1f);
 }

      fwrite(&outvector[0],1,25,outfile);
      fprintf(hexfile,":19%04X00",offset);
      chksum = (offset & 0xff) + (offset >> 8) + 0x19;
      for (j=0; j<25; j++)
 {
   charptr = ((char*) &outvector[0]) + j;
   fprintf(hexfile,"%02X", (0xff& *charptr));
   chksum += *charptr;
 }
      offset += 25;
      fprintf(hexfile,"%02X\n", (-chksum) & 0xff);
    }
  fprintf(hexfile,":00000001FF\n");
  if (instatus == 0)
    if (fread(&instr[0],1,16,infile))
      {
 printf("There are more instructions in the input file than can");
 printf("be fit in the output file: \n");
 printf(" truncated the input file after 8K of instructions!!!\n");
      }
  printf("\n");
  printf("Total intructions processed = %d\n", instr_count);
  fclose(infile);
  fclose(outfile);
  fclose(hexfile);
  exit(0);

}

int eparity(int x)
{

  x = x ^ (x >> 16);
  x = x ^ (x >> 8);
  x = x ^ (x >> 4);
  x = x ^ (x >> 2);
  x = x ^ (x >> 1);

  return (x&1);

}
#define EXT(data, bit)\
    (((data) & ((unsigned) 1 << (bit))) != 0)
#define EXTV(data, hbit, lbit)\
    (((data) >> (lbit)) & \
     ((((hbit) - (lbit) + 1) == 32) ? ((unsigned)0xffffffff) : \
      (~((unsigned)0xffffffff << ((hbit) - (lbit) + 1)))))
#define INS(name, bit, data)\
  (name) = (((name) & ~((unsigned) 1 << (bit))) | \
     (((unsigned) (data) << (bit)) & ((unsigned) 1 << (bit))))
```

```
int instrpredecode(int inst)
{

int result;
int opcode;
int func;
int jsr_type;
int ra;

int out0;
int out1;
int out2;
int out3;
int out4;
int e0_only;
int  e1_only;
int  ee;
int lnoop;
int  fadd;
int  fmul;
int  fe;
int  br_type;
int  ld;
int  store;
int  br;
int  call_pal;
int  bsr;
int  ret_rei;
int  jmp;
int  jsr_cor;
int  jsr;
int  cond_br;

 opcode = EXTV(inst, 31, 26 );
 func   = EXTV(inst, 12, 5);
 jsr_type = EXTV(inst, 15,14);
 ra = EXTV(inst,25,21);
```

```
e0_only  = (opcode == 0x24) ||      /* STF */
    (opcode == 0x25) ||       /* STG */
    (opcode == 0x26) ||       /* STS */
    (opcode == 0x27) ||       /* STT */
    (opcode == 0x0F) ||       /* STQ_U */
    (opcode == 0x2A) ||       /* LDL_L */
    (opcode == 0x2B) ||       /* LDQ_L */
    (opcode == 0x2C) ||       /* STL */
    (opcode == 0x2D) ||       /* STQ */
    (opcode == 0x2E) ||       /* STL_C */
    (opcode == 0x2F) ||       /* STQ_C */
    (opcode == 0x1F) ||       /* HW_ST*/
    (opcode == 0x18) ||
      /* MISC mem format: FETCH/_M, RS, RC, RPCC, TRAPB, MB) */
    (opcode == 0x12) ||       /* EXT,MSK,INS,SRX,SLX,ZAP*/
    (opcode == 0x13) ||       /* MULX */
    ((opcode == 0x1D) && (EXT(inst,8) == 0)) || /* MBOX HW_MTPR */
    ((opcode == 0x19) && (EXT(inst,8) == 0)) || /* MBOX HW_MFPR */
    (opcode == 0x01) ||               /* RESDEC's */
    (opcode == 0x02) ||               /* RESDEC's */
    (opcode == 0x03) ||               /* RESDEC's */
    (opcode == 0x04) ||               /* RESDEC's */
    (opcode == 0x05) ||               /* RESDEC's */
    (opcode == 0x06) ||               /* RESDEC's */
    (opcode == 0x07) ||               /* RESDEC's */
    (opcode == 0x0a) ||               /* RESDEC's */
    (opcode == 0x0c) ||               /* RESDEC's */
    (opcode == 0x0d) ||       /* RESDEC's */
    (opcode == 0x0e) ||       /* RESDEC's */
    (opcode == 0x14) ||       /* RESDEC's */
    (opcode == 0x1c);      /* RESDEC's */
```

```
e1_only =  (opcode == 0x30) ||      /* BR */
    (opcode == 0x34) ||       /* BSR */
    (opcode == 0x38) ||       /* BLBC */
    (opcode == 0x39) ||       /* BEQ */
    (opcode == 0x3A) ||       /* BLT */
    (opcode == 0x3B) ||       /* BLE */
    (opcode == 0x3C) ||       /* BLBS */
    (opcode == 0x3D) ||       /* BNE */
    (opcode == 0x3E) ||       /* BGE */
    (opcode == 0x3F) ||       /* BGT */
    (opcode == 0x1A) ||       /* JMP,JSR,RET,JSR_COROT */
    (opcode == 0x1E) ||       /* HW_REI */
    (opcode == 0x00) ||       /* CALL_PAL */
    ((opcode == 0x1D) && (EXT(inst,8) == 1)) || /* IBOX HW_MTPR */
    ((opcode == 0x19) && (EXT(inst,8) == 1));   /* IBOX HW_MTPR */
ee =       (opcode == 0x10) ||      /* ADD, SUB, CMP */
    (opcode == 0x11) ||       /* AND, BIC etc. logicals */
    (opcode == 0x28) ||       /* LDL */
    (opcode == 0x29) ||       /* LDQ */
    (opcode == 0x0B)&(ra != 0x1F) ||  /* LDQ_U */
    (opcode == 0x08) ||       /* LDA */
    (opcode == 0x09) ||       /* LDAH */
    (opcode == 0x20) ||       /* LDF */
    (opcode == 0x21) ||       /* LDG */
    (opcode == 0x22) ||       /* LDS */
    (opcode == 0x23) ||       /* LDT */
    (opcode == 0x1B);       /* HW_LD */


lnoop =    (opcode == 0x0B)&(ra == 0x1F);    /* LDQ_U R31, x(y) - NOOP*/

fadd =    ((opcode == 0x17) && (func != 0x20)) ||
                                            /* Flt, datatype indep excl CPYS */
    ((opcode == 0x15) && ((func & 0xf) != 0x2)) ||
                                     /* VAX excl MUL's */
    ((opcode == 0x16) && ((func & 0xf) != 0x2)) ||
                                     /* IEEE excl MUL's */
    (opcode == 0x31) ||       /* FBEQ */
    (opcode == 0x32) ||       /* FBLT */
    (opcode == 0x33) ||       /* FBLE */
    (opcode == 0x35) ||       /* FBNE */
    (opcode == 0x36) ||       /* FBGE */
    (opcode == 0x37);       /* FBGT */

fmul =     ((opcode == 0x15) && ((func & 0xf) == 0x2)) ||   /* VAX MUL's */
    ((opcode == 0x16) && ((func & 0xf) == 0x2));     /* IEEE MUL's */

fe =       ((opcode == 0x17) && (func == 0x20));    /*  CPYS */

br_type =  ((opcode & 0x30) == 0x30) ||   /* all branches */
    (opcode == 0x1A) ||     /* JMP's */
    (opcode == 0x00) ||     /* CALL PAL */
    (opcode == 0x1E);    /* HW_REI */
```

```
ld =    (opcode == 0x28) ||    /* LDL */
    (opcode == 0x29) ||    /* LDQ */
/*    (opcode == 0x2A) ||       LDL_L */
/*    (opcode == 0x2B) ||       LDQ_L */
    (opcode == 0x0B) ||    /* LDQ_U */
    (opcode == 0x20) ||    /* LDF */
    (opcode == 0x21) ||    /* LDG */
    (opcode == 0x22) ||    /* LDS */
    (opcode == 0x23) ||    /* LDT */
    (opcode == 0x1B);    /* HW_LD */

store =    (opcode == 0x24) ||    /* STF */
    (opcode == 0x25) ||    /* STG */
    (opcode == 0x26) ||    /* STS */
    (opcode == 0x27) ||    /* STT */
    (opcode == 0x0F) ||    /* STQ_U */
    (opcode == 0x2C) ||    /* STL */
    (opcode == 0x2D) ||    /* STQ */
    (opcode == 0x2E) ||    /* STL_C */
    (opcode == 0x2F) ||    /* STQ_C */
    (opcode == 0x18) ||
      /* Misc: TRAPB, MB, RS, RC, RPCC etc. */
          (opcode == 0x1F) ||    /* HW_ST */
    (opcode == 0x2A) ||    /* LDL_L */
    (opcode == 0x2B);    /* LDQ_L */


br =    (opcode == 0x30);    /* all branches */

call_pal = (opcode == 0x00);    /* call PAL */

bsr =    (opcode == 0x34);

ret_rei =  ((opcode == 0x1A) && (jsr_type == 0x2)) ||
    ((opcode == 0x1E) && (jsr_type != 0x3));

jmp = ((opcode == 0x1A) && (jsr_type == 0x0));

jsr_cor = ((opcode == 0x1A) && (jsr_type == 0x3));

jsr = ((opcode == 0x1A) && (jsr_type == 0x1));

cond_br = (opcode == 0x31) ||
   (opcode == 0x32) ||
   (opcode == 0x33) ||
   (opcode == 0x35) ||
   (opcode == 0x36) ||
   (opcode == 0x37) ||
   (opcode == 0x38) ||
   (opcode == 0x39) ||
   (opcode == 0x3A) ||
   (opcode == 0x3B) ||
   (opcode == 0x3C) ||
   (opcode == 0x3D) ||
   (opcode == 0x3E) ||
   (opcode == 0x3F);
```

```
out0 = br || bsr || jmp || jsr || (ee && !ld) || (e0_only && !store);
out1 = ret_rei ||(e1_only && !br_type)|| jmp ||jsr_cor|| jsr || lnoop ||
        (fadd && !br_type) || fe;;
out2 = call_pal || bsr || jsr_cor || e0_only ||jsr ||fmul || fe;
out3 = (e1_only && cond_br) || (e1_only && !br_type) || fadd || fmul || fe;
out4 = ee || lnoop || e0_only || fadd || fmul || fe;

 result = 0;
 INS( result, 0, out0 );
 INS( result, 1, out1 );
 INS( result, 2, out2 );
 INS( result, 3, out3 );
 INS( result, 4, out4 );

return (result);

}
```

D

Errata Sheet

Table D–1 lists the revision history for this document.

Table D–1 Document Revision History

Date	Revision
March 28, 1994	First draft.
May 16, 1994	Second draft.
July 20, 1994	First Preliminary version.
September 12, 1994	Second Preliminary version. (First printing.)
December 22, 1994	Final draft.
April 3, 1995	Third Preliminary version. (Second printing.)

E

Technical Support and Ordering Information

E.1 Technical Support

If you need technical support or help deciding which literature best meets your needs, call the Semiconductor Information Line:

United States and Canada	**1–800–332–2717**
TTY (United States only)	**1–800–332–2515**
Outside North America	**+1–508–568–6868**

E.2 Ordering Digital Semiconductor Products

To order the Alpha 21164 microprocessor and evaluation boards, contact your local distributor.

You can order the following semiconductor products from Digital:

Product	Order Number
Alpha 21164 300 MHz Microprocessor	21164–300MHZ
Alpha 21164 266 MHz Microprocessor	21164–266MHZ
Alpha 21164 Microprocessor Evaluation Board 266 MHz Kit (Supports OSF/1, OpenVMS, and Windows NT operating systems.)	21A04–01

E.3 Ordering Digital Semiconductor Sample Kits

To order an Alpha 21164 Microprocessor Sample Kit, which contains one Alpha 21164 microprocessor, one heat sink, and supporting documentation, call **1–800–DIGITAL**. You will need a purchase order number or credit card to order the following products:

E.3 Ordering Digital Semiconductor Sample Kits

Product	Order Number
Alpha 21164–266 Sample Kit	21164–SA

E.4 Ordering Associated Literature

The following table lists some of the available Digital Semiconductor literature. For a complete list, contact the Digital Semiconductor Information Line.

Title	Order Number
Alpha Architecture Reference Manual[1]	EY–L520E–DP–YCH
Alpha AXP Architecture Handbook	EC–QD2KA–TE
Alpha 21164 Microprocessor Data Sheet	EC–QAEPB–TE
Alpha 21164 Microprocessor Product Brief	EC–QAENB–TE
Alpha 21164 Evaluation Board Read Me First	EC–QD2VA–TE
Alpha 21164 Evaluation Board Product Brief	EC–QCZZB–TE
Alpha 21164 Evaluation Board User's Guide	EC–QD2UA–TE
DECchip 21171 Core Logic Chipset Product Brief	EC–QC3EB–TE
DECchip 21171 Core Logic Chipset Technical Reference Manual	EC–QE18A–TE
Answers to Common Questions about PALcode for Alpha AXP Systems	EC–N0647–72
PALcode for Alpha AXP Microprocessors System Design Guide	EC–N0660–72
Alpha Microprocessor Evaluation Board System Configuration and Windows NT 3.5 Installation Guide	EC–QD2YA–TE
SPICE Models for Alpha Microprocessors and Peripheral Chips: An Application Note	EC–QA4XB–TE
Alpha Microprocessors SROM Mini-Debugger User's Guide	EC–QHUXA–TE
Alpha Microprocessors Evaluation Board Debug Monitor User's Guide	EC–QHUVA–TE
Alpha Microprocessors Evaluation Board Software Design Tools User's Guide	EC–QHUWA–TE

[1]To order and purchase the *Alpha Architecture Reference Manual*, call **1–800–DIGITAL** from the U.S. or Canada, or contact your local Digital office, or technical or reference bookstore where Digital Press books are distributed by Prentice Hall.

E.5 Ordering Associated Third-Party Literature

You can order the following third-party literature directly from the vendor:

Title	Vendor
PCI System Design Guide	PCI Special Interest Group N/S HH3–15A 5200 N.E. Elam Young Pkwy Hillsboro, Oregon 97124–6497 1–503–696–2000
PCI Local Bus Specification, Rev 2.0	See previous entry.
IEEE Standard 754, *Standard for Binary Floating-Point Arithmetic*	IEEE Service Center 445 Hoes Lane P.O. Box 1331 Piscataway, NJ 08855–1331 1–800–678–IEEE (U.S. and Canada) 908–562–3805 (Outside U.S. and Canada)
IEEE Standard 1149.1, *A Test Access Port and Boundary Scan Architecture*	See previous entry.

Glossary

The glossary provides definitions for specific terms and acronyms associated with the Alpha 21164 microprocessor and chips in general.

abort

The unit stops the operation it is performing, without saving status, to perform some other operation.

ABT

Advanced bipolar/CMOS technology.

address space number (ASN)

An optionally implemented register used to reduce the need for invalidation of cached address translations for process-specific addresses when a context switch occurs. ASNs are processor specific; the hardware makes no attempt to maintain coherency across multiple processors.

address translation

The process of mapping addresses from one address space to another.

ALIGNED

A 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).

ALU

Arithmetic logic unit.

ANSI

American National Standards Institute. An organization that develops and publishes standards for the computer industry.

ASIC

Application-specific integrated circuit.

ASN

See address space number.

assert

To cause a signal to change to its logical true state.

AST

See asynchronous system trap.

asynchronous system trap (AST)

A software-simulated interrupt to a user-defined routine. ASTs enable a user process to be notified asynchronously, with respect to that process, of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes execution of the process at the point where it was interrupted.

backmap

A memory unit that is used to note addresses of valid entries within a cache.

bandwidth

Bandwidth is often used to express ‘‘high rate of data transfer’’ in a bus or an I/O channel. This usage assumes that a wide bandwidth may contain a high frequency, which can accommodate a high rate of data transfer.

Bcache

See external cache.

barrier transaction

A transaction on the external interface as a result of an MB (memory barrier) instruction.

BCT

Bipolar/CMOS technology.

BiCMOS

Bipolar/CMOS. The combination of bipolar and MOSFET transistors in a common integrated circuit.

bidirectional

Flowing in two directions. The buses are bidirectional; they carry both input and output signals.

BiSr

Built-in self-repair.

BiSt

Built-in self-test.

bit

Binary digit. The smallest unit of data in a binary notation system, designated as 0 or 1.

BIU

Bus interface unit. *See* Cbox.

block exchange

Memory feature that improves bus bandwidth by paralleling a cache victim write-back with a cache miss fill.

board-level cache

See external cache.

boot

Short for bootstrap. Loading an operating system into memory is called booting.

BSR

Boundary scan register.

buffer

An internal memory area used for temporary storage of data records during input or output operations.

bugcheck

A software condition, usually the response to software's detection of an ''internal inconsistency,'' which results in the execution of the system bugcheck code.

bus

A group of signals that consists of many transmission lines or wires. It interconnects computer system components to provide communications paths for addresses, data, and control information.

byte

Eight contiguous bits starting on an addressable byte boundary. The bits are numbered right to left, 0 through 7.

byte granularity

Memory systems are said to have byte granularity if adjacent bytes can be written concurrently and independently by different processes or processors.

cache

See cache memory.

cache block

The smallest unit of storage that can be allocated or manipulated in a cache. Also known as a cache line.

cache coherence

Maintaining cache coherence requires that when a processor accesses data cached in another processor, it must not receive incorrect data and when cached data is modified, all other processors that access that data receive modified data. Schemes for maintaining consistency can be implemented in hardware or software. Also called cache consistency.

cache fill

An operation that loads an entire cache block by using multiple read cycles from main memory.

cache flush

An operation that marks all cache blocks as invalid.

cache hit

The status returned when a logic unit probes a cache memory and finds a valid cache entry at the probed address.

cache interference

The result of an operation that adversely affects the mechanisms and procedures used to keep frequently used items in a cache. Such interference may cause frequently used items to be removed from a cache or incur significant overhead operations to ensure correct results. Either action hampers performance.

cache line

See cache block.

cache line buffer

A buffer used to store a block of cache memory.

cache memory

A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes. The Alpha 21164 microprocessor contains three onchip internal caches. *See also* write-through cache and write-back cache.

cache miss

The status returned when cache memory is probed with no valid cache entry at the probed address.

CALL_PAL Instructions

Special instructions used to invoke PALcode.

Cbox

The external interface control logic unit. Provides the 21164 microprocessor with an interface to the external data bus, board-level Bcache, and the onchip Scache.

central processing unit (CPU)

The unit of the computer that is responsible for interpreting and executing instructions.

CISC

Complex instruction set computing. An instruction set consisting of a large number of complex instructions that are managed by microcode. *Contrast with* RISC.

clean

In the cache of a system bus node, refers to a cache line that is valid but has not been written.

clock

A signal used to synchronize the circuits in a computer

CMOS

Complementary metal-oxide semiconductor. A silicon device formed by a process that combines PMOS and NMOS semiconductor material.

conditonal branch instructions

Instructions that test a register for positive/negative or for zero/non-zero. They can also test integer registers for even/odd.

control and status register (CSR)

A device or controller register that resides in the processor's I/O space. The CSR initiates device activity and records its status.

CPLD

Complex programmable logic device.

CPU

See central processing unit.

CSR

See control and status register.

cycle

One clock interval.

data bus

The bus used to carry data between the 21164 and external devices. Also called the pin bus.

Dcache

Data cache. A cache reserved for storage of data. The Dcache does not contain instructions.

DIP

Dual inline package.

direct-mapping cache

A cache organization in which only one address comparison is needed to locate any data in the cache, because any block of main memory data can be placed in only one possible position in the cache.

direct memory access (DMA)

Access to memory by an I/O device that does not require processor intervention.

dirty

One status item for a cache block. The cache block is valid and has been written so that it may differ from the copy in system main memory.

dirty victim

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict. The data must therefore be written to memory.

DRAM

Dynamic random-access memory. Read/write memory that must be refreshed (read from or written to) periodically to maintain the storage of information.

DTL

Diode-transistor logic.

dual issue

Two instructions are issued, in parallel, during the same microprocessor cycle. The instructions use different resources and so do not conflict.

EB164

An evaluation board. A hardware/software applications development platform for the Alpha program and a debug platform for the Alpha 21164 microprocessor.

Ebox

The Ebox contains the 64-bit integer execution data path.

ECC

Error correction code. Code and algorithms used by logic to facilitate error detection and correction. *See also* ECC error.

ECC error

An error detected by ECC logic, to indicate that data (or the protected ‘‘entity’’ has been corrupted. The error may be correctable (soft error) or uncorrectable (hard error).

ECL

Emitter-coupled logic.

EEPROM

Electrically erasable programmable read-only memory. A memory device that can be byte-erased, written to, and read from. *Contrast with* FEPROM.

EPLD

Erasable programmable logic device.

external cache

A cache memory provided outside of the microprocessor chip, usually located on the same module. Also called board-level or module-level cache.

Fbox

The unit within the 21164 microprocessor that performs floating-point calculations.

FEPROM

Flash-erasable programmable read-only memory. FEPROMs can be bank- or bulk-erased. *Contrast with* EEPROM.

FET

Field-effect transistor.

firmware

Machine instructions stored in hardware.

floating point

A number system in which the position of the radix point is indicated by the exponent part and another part represents the significant digits or fractional part.

flush

See cache flush.

FPGA

Field-programmable gate array.

FPLA

Field-programmable logic array.

granularity

A characteristic of storage systems that defines the amount of data that can be read and/or written with a single instruction, or read and/or written independently. VAX systems have byte or multibyte granularities, whereas disk systems typically have 512-byte or greater granularities. For a given storage device, a higher granularity generally yields a greater throughput.

hardware interrupt request (HIR)

An interrupt generated by a peripheral device.

high-impedance state

An electrical state of high resistance to current flow, which makes the device appear not physically connected to the circuit.

hit

See cache hit.

Ibox

A logic unit within the 21164 microprocessor that fetches, decodes, and issues instructions. It also controls the microprocessor pipeline.

Icache

Instruction cache. A cache reserved for storage of instructions. One of the three areas of primary cache (located on the 21164) used to store instructions. The Icache contains 8K bytes of memory space. It is a direct-mapped cache. Icache blocks, or lines, contain 32 bytes of instruction stream data with associated tag as well as a 6-bit ASM field and an 8-bit branch history field per block. Icache does not contain hardware for maintaining cache coherency with memory and is unaffected by the invalidate bus.

IEEE Standard 754

A set of formats and operations that apply to floating-point numbers. The formats cover 32-, 64-, and 80-bit operand sizes.

IEEE Standard 1149.1

A standard for the Test Access Port and Boundary Scan Architecture used in board-level manufacturing test procedures. Commonly referred to as the Joint Test Action Group (JTAG) standard.

INT*nn*

The term INT*nn*, where *nn* is one of 2, 4, 8, 16, 32, or 64, refers to a data field size of *nn* contiguous NATURALLY ALIGNED bytes. For example, INT4 refers to a NATURALLY ALIGNED longword.

internal processor register (IPR)

One of many registers internal to the Alpha 21164 microprocessor.

IPGA

Interstitial pin grid array.

JFET

Junction field-effect transistor.

latency

The amount of time it takes the system to respond to an event.

LCC

Leadless chip carrier.

LFSR

Linear feedback shift register.

load/store architecture

A characteristic of a machine architecture where data items are first loaded into a processor register, operated on, and then stored back to memory. No operations on memory other than load and store are provided by the instruction set.

longword

Four contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 31.

LSB

Least significant bit.

machine check

An operating system action triggered by certain system hardware-detected errors that can be fatal to system operation. Once triggered, machine check handler software analyzes the error.

MAF

Miss address file.

main memory

The large memory, external to the microprocessor, used for holding most instruction code and data. Usually built from cost-effective DRAM memory chips. May be used in connection with the microprocessor's internal caches and an optional external cache.

masked write

A write cycle that only updates a subset of a nominal data block.

MBO

See must be one.

Mbox

This section of the processor unit performs address translation, interfaces to the Dcache, and performs several other functions.

MBZ

See must be zero.

MESI protocol

A cache consistency protocol with full support for multiprocessing. The MESI protocol consists of four states that define whether a block is modified (M), exclusive (E), shared (S), or invalid (I).

MIPS

Millions of instructions per second.

miss

See cache miss.

module

A board on which logic devices (such as transistors, resistors, and memory chips) are mounted and connected to perform a specific system function.

module-level cache

See external cache.

MOS

Metal-oxide semiconductor.

MOSFET

Metal-oxide semiconductor field-effect transistor.

MSI

Medium-scale integration.

multiprocessing

A processing method that replicates the sequential computer and interconnects the collection so that each processor can execute the same or a different program at the same time.

Must be one (MBO)

A field that must be supplied as one.

Must be zero (MBZ)

A field that is reserved and must be supplied as zero. If examined, it must be assumed to be UNDEFINED.

NATURALLY ALIGNED

See ALIGNED.

NATURALLY ALIGNED data

Data stored in memory such that the address of the data is evenly divisible by the size of the data in bytes. For example, an ALIGNED longword is stored such that the address of the longword is evenly divisible by 4.

NMOS

N-type metal-oxide semiconductor.

NVRAM

Nonvolatile random-access memory.

OBL

Observability linear feedback shift register.

octaword

Sixteen contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 127.

OpenVMS Alpha operating system

Digital's open version of the VMS operating system, which runs on Alpha platforms.

operand

The data or register upon which an operation is performed.

PAL

Privileged architecture library. *See also* PALcode. *Also* Programmable array logic (hardware). A device that can be programmed by a process that blows individual fuses to create a circuit.

PALcode

Alpha privileged architecture library code, written to support Alpha microprocessors. PALcode implements architecturally defined behavior.

PALmode

A special environment for running PALcode routines.

parameter

A variable that is given a specific value that is passed to a program before execution.

parity

A method for checking the accuracy of data by calculating the sum of the number of ones in a piece of binary data. Even parity requires the correct sum to be an even number, odd parity requires the correct sum to be an odd number.

PGA

Pin grid array.

pipeline

A CPU design technique whereby multiple instructions are simultaneously overlapped in execution.

PLA

Programmable logic array.

PLCC

Plastic leadless chip carrier or plastic-leaded chip carrier.

PLD

Programmable logic device.

PLL

Phase-locked loop.

PMOS

P-type metal-oxide semiconductor.

PQFP

Plastic quad flat pack.

primary cache

The cache that is the fastest and closest to the processor. The first-level caches, located on the CPU chip, composed of the Dcache, Icache, and Scache.

program counter

That portion of the CPU that contains the virtual address of the next instruction to be executed. Most current CPUs implement the program counter (PC) as a register. This register may be visible to the programmer through the instruction set.

PROM

Programmable read-only memory.

pull-down resistor

A resistor placed between a signal line and a negative voltage.

pull-up resistor

A resistor placed between a signal line to a positive voltage.

quad issue

Four instructions are issued, in parallel, during the same microprocessor cycle. The instructions use different resources and so do not conflict.

quadword

Eight contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 63.

RAM

Random-access memory.

READ_BLOCK

A transaction where the 21164 requests that an external logic unit fetch read data.

read data wrapping

System feature that reduces apparent memory latency by allowing read data cycles to differ the usual low-to-high sequence. Requires cooperation between the 21164 and external hardware.

read stream buffers

Arrangement whereby each memory module independently prefetches DRAM data prior to an actual read request for that data. Reduces average memory latency while improving total memory bandwidth.

register

A temporary storage or control location in hardware logic.

reliability

The probability a device or system will not fail to perform its intended functions during a specified time interval when operated under stated conditions.

reset

An action that causes a logic unit to interrupt the task it is performing and go to its' initialized state.

RISC

Reduced instruction set computing. A computer with an instruction set that is paired down and reduced in complexity so that most can be performed in a single processor cycle. High-level compilers synthesize the more complex, least frequently used instructions by breaking them down into simpler instructions. This approach allows the RISC architecture to implement a small, hardware-assisted instruction set, thus eliminating the need for microcode.

ROM

Read-only memory.

RTL

Register-transfer logic.

SAM

Serial access memory.

SBO

Should be one.

SBZ

Should be zero.

Scache

Secondary cache. A 3-way set-associative, second-level cache located on the Alpha 21164 microprocessor.

scheduling

The process of ordering instruction execution to obtain optimum performance.

set-associative

A form of cache organization in which the location of a data block in main memory constrains, but does not completely determine, its location in the cache. Set-associative organization is a compromise between direct-mapped organization, in which data from a given address in main memory has only one possible cache location, and fully associative organization, in which data from anywhere in main memory can be put anywhere in the cache. An ''*n*-way set-associative'' cache allows data from a given address in main memory to be cached in any of *n* locations. The Scache in the 21164 microprocessor has a 3-way set-associative organization.

SIMM

Single inline memory module.

SIP

Single inline package.

SIPP

Single inline pin package.

SMD

Surface mount device.

SRAM

Static random-access memory.

SROM

Serial read-only memory.

SSI

Small-scale integration.

SSRAM

Synchronous static random-access memory.

stack

An area of memory set aside for temporary data storage or for procedure and interrupt service linkages. A stack uses the last-in/first-out concept. As items are added to (pushed on) the stack, the stack pointer decrements. As items are retrieved from (popped off) the stack, the stack pointer increments.

STRAM

Self-timed random-access memory.

superpipelined

Describes a pipelined machine that has a larger number of pipe stages and more complex scheduling and control. *See also* pipeline.

superscalar

Describes a machine architecture that allows multiple independent instructions to be issued in parallel during a given clock cycle.

tag

The part of a cache block that holds the address information used to determine if a memory operation is a hit or a miss on that cache block.

TB

Translation buffer.

tristate

Refers to a bused line that has three states: high, low, and high-impedance.

TTL

Transistor–transistor logic.

UART

Universal asynchronous receiver-transmitter.

UNALIGNED

A datum of size 2**N stored at a byte address that is not a multiple of 2**N.

unconditional branch instructions

Instructions that write a return address into a register.

UNDEFINED

An operation that may halt the processor or cause it to lose information. Only privileged software (that is, software running in kernel mode) can trigger an UNDEFINED operation.

UNPREDICTABLE

Results or occurrences that do not disrupt the basic operation of the processor; the processor continues to execute instructions in its normal manner. Privileged or unprivileged software can trigger UNPREDICTABLE results or occurrences.

UVPROM

Ultraviolet (erasable) programmable read-only memory.

valid

Allocated. Valid cache blocks have been loaded with data and may return cache hits when accessed.

victim

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict.

virtual cache

A cache that is addressed with virtual addresses. The tag of the cache is a virtual address. This process allows direct addressing of the cache without having to go through the translation buffer making cache hit times faster.

VHSIC

Very-high-speed integrated circuit.

VLSI

Very-large-scale integration.

VRAM

Video random-access memory.

word

Two contiguous bytes (16 bits) starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 15.

write data wrapping

System feature that reduces apparent memory latency by allowing write data cycles to differ the usual low-to-high sequence. Requires cooperation between the 21164 and external hardware.

write-back

A cache management technique in which write operation data is written into cache but is not written into main memory in the same operation. This may result in temporary differences between cache data and main memory data. Some logic unit must maintain coherency between cache and main memory.

write-back cache

Copies are kept of any data in the region; read and write operations may use the copies, and write operations use additional state to determine whether there are other copies to invalidate or update.

write-through

A cache management technique in which a write operation to cache also causes the same data to be written in main memory during the same operation.

write-through cache

Copies are kept of any data in the region; read operations may use the copies, but write operations update the actual data location and either update or invalidate all copies.

WRITE_BLOCK

A transaction where the 21164 requests that an external logic unit process write data.

Index

A

Aborts, 2–18
Absolute Maximum Rating, 9–1
ac coupling, 9–6
Addressing, 1–2
Address regions, physical, 4–12
Address translation, 2–10
addr_bus_req_h
  description, 3–3
  operation, 4–44, 4–53, 4–70, 5–81, 7–4, 9–11
addr_cmd_par_h
  description, 3–3
  operation, 3–3, 4–70, 4–71, 4–95, 7–4, 9–17, 9–18
addr_h<39:4>
  description, 3–3
  operation, 3–3, 4–13, 4–14, 4–15, 4–16, 4–39, 4–52, 4–70, 4–71, 4–75, 4–95, 7–3, 9–10, 9–11, 9–13
addr_res_h<2:0>
  description, 3–4
  operation, 4–56, 4–57, 4–58, 4–64, 4–65, 7–4, 9–17
Alpha documentation, E–2
ALT_MODE register, 5–60
Architecture, 1–1 to 1–3
Associated literature, E–2
AST, 2–9
ASTER register, 5–26
ASTRR register, 5–25
Asynchronous system trap
  See AST

B

Bcache, 2–14
  block size, 4–15
  errors, 4–92
  hit under READ MISS example, 4–90
  interface, 4–4
    introduction, 4–2 to 4–4
  selecting options, 4–35
  structure, 4–15
  systems without, 4–19, 4–81
  timing, 4–31
  victim buffers, 4–18
Bcache read transaction
  private read operation, 4–32
BCACHE VICTIM command, 4–38
Bcache write transaction
  private write operation, 4–34
BC_CONFIG register, 5–84
BC_CONTROL register, 5–78
BC_TAG_ADDR register, 5–89
BIU, 4–2, 4–14, 4–30, 4–31, 4–44, 4–53, 4–83
  See also Cbox
  buffer, 4–4
Block diagram, 21164, 2–2
Boundaries
  data wrap order, 4–13
Boundary scan register, 12–7

Branch prediction, 2–5, 2–20
Bubble cycle, 2–32
Bubble squashing, 2–20
Bus contention
  command/address bus, 4–70 to 4–80
  data bus, 4–70 to 4–80
Bus interface unit
  See BIU

C

Cache coherency, 4–19 to 4–29
  basics, 4–19
  flush protocol, 4–21
    state machines, 4–27
    systems, 4–25
  transaction conflicts, 4–28
  write invalidate protocol, 4–21
    state machines, 4–24
    states, 4–23
    systems, 4–22
Cache control and bus interface unit
  See Cbox
Cache organization, 2–13
cack_h
  description, 3–4
  operation, 3–4, 4–31, 4–36, 4–37, 4–40, 4–41, 4–44, 4–46, 4–48, 4–50, 4–52, 4–81, 4–82, 4–83, 4–84, 4–86, 4–90, 4–95, 5–13, 5–21, 5–81, 7–4, 8–10, 8–11, 9–11, 9–13, 9–14, 12–7
Cbox, 2–12
  IPR PALcode restrictions, 5–100
  IPRs, 5–68 to 5–98
  read requests, 2–31
  write buffer data store, 2–35
  write ordering, 2–37
CC register, 5–61
CC_CTL register, 5–62
cfail_h
  description, 3–4
  operation, 4–31, 4–38, 4–82, 4–95, 5–13, 5–21, 7–4, 8–10, 8–11, 9–16, 12–7
clk_mode_h<1:0>
  description, 3–4
  operation, 4–5, 7–3, 9–16, 9–23
Clocks, 4–5 to 4–12
  CPU, 4–5
  reference, 4–8, 4–9
  system, 4–6
cmd_h<3:0>
  description, 3–4
  operation, 3–3, 4–37, 4–41, 4–48, 4–53, 4–55, 4–64, 4–71, 4–75, 4–83, 4–95, 7–4, 9–10, 9–17, 9–18
Coherency, caches, 4–19
Command/address
  driving bus, 4–70
  errors, 4–92
Commands
  21164 initiated, 4–37
  BCACHE VICTIM, 4–38
  FETCH, 4–37
  FETCH_M, 4–37
  FLUSH, 4–64
  INVALIDATE, 4–55
  LOCK, 4–37
  MEMORY BARRIER, 4–37
  NOP, 4–37, 4–55, 4–64
  READ, 4–64
  READ DIRTY, 4–55
  READ DIRTY/INVALIDATE, 4–56
  READ MISS0, 4–38
  READ MISS1, 4–38
  READ MISS MOD0, 4–38
  READ MISS MOD1, 4–38
  READ MISS MOD STC0, 4–39
  READ MISS MOD STC1, 4–39
  SET DIRTY, 4–37
  SET SHARED, 4–55
  WRITE BLOCK, 4–37
  WRITE BLOCK LOCK, 4–38
Commands, sending to 21164, 4–53
Conventions, xxii to xxvii
CPU
  clock, 4–5
  microarchitecture, 2–2

cpu_clk_out_h
description, 3–6
operation, 3–11, 4–5, 7–3, 9–4

D

dack_h
description, 3–6
operation, 3–7, 4–31, 4–36, 4–37, 4–39, 4–41, 4–43, 4–44, 4–46, 4–48, 4–58, 4–66, 4–76, 4–77, 4–78, 4–80, 4–81, 4–82, 4–83, 4–84, 4–86, 4–90, 4–95, 5–81, 7–4, 8–9, 9–11, 9–13, 9–14
Data cache
See Dcache
Data integrity, 4–92
address and command parity, 4–95
Bcache tag control parity, 4–94
Bcache tag data parity, 4–94
ECC and parity, 4–92
force correction, 4–94
Data translation buffer
See DTB
Data types, 1–1
floating-point, 1–3, 2–10
integer, 1–2
Data wrap order, 4–13
data_bus_req_h
description, 3–6
operation, 4–43, 4–72, 4–74, 4–80, 7–4, 9–11, 9–13, 9–14
data_check_h<15:0>
description, 3–7
operation, 4–70, 4–92, 7–3, 9–17, 9–18
data_h<127:0>
description, 3–6
operation, 3–8, 4–43, 4–46, 4–55, 4–64, 4–70, 4–71, 4–75, 4–92, 4–93, 7–3, 9–9, 9–10, 9–11, 9–13
data_ram_oe_h
description, 3–7
operation, 4–32, 4–43, 4–76, 4–77, 4–78, 4–79, 4–80, 7–3, 9–19
data_ram_we_h
description, 3–7
operation, 4–34, 7–3, 9–19
Dcache, 2–13
control, 2–12
DC_FLUSH register, 5–60
DC_MODE register, 5–56
dc_ok_h
description, 3–7
operation, 3–11, 4–5, 7–1, 7–2, 7–3, 7–5, 7–13, 9–4, 9–5, 9–16, 12–2, 12–3
DC_PERR_STAT register, 5–50
DC_TEST_CTL register, 5–63
DC_TEST_TAG register, 5–64
DC_TEST_TAG_TEMP register, 5–66
Decoupling, 9–24
Delayed system clock, 4–8
Design examples, 2–40
Documentation, E–2
DTB, 2–11
DTB_ASN register, 5–38
DTB_CM register, 5–39
DTB_IAP register, 5–52
DTB_IA register, 5–52
DTB_IS register, 5–53
DTB_PTE register, 5–41
DTB_PTE_TEMP register, 5–43
DTB_TAG register, 5–40
Duplicate tag store, 4–15
algorithm, 4–17
full, 4–16
partial Scache, 4–18

E

Ebox, 2–9
registers, 2–9, 5–99
ECC, 4–92 to 4–94
EI_ADDR register, 5–94
EI_STAT register, 5–91
Entry-pointer queues, 2–36
Environment instructions
PALcode, 6–7

Error correction code
  See ECC
Exceptions, 2–18
EXC_ADDR register, 5–14
EXC_MASK register, 5–17
EXC_SUM register, 5–15
External cache
  See Bcache
External interface
  rules for use, 4–83
External interface introduction, 4–2 to 4–4

F

Features, 1–3 to 1–4
FETCH command, 4–37, 4–52
FETCH_M command, 4–37, 4–52
Fill, 2–32
FILL, after other transactions, 4–81
FILL error, 4–95
FILL transaction, 4–43
fill_error_h
  description, 3–7
  operation, 4–43, 4–95, 7–4, 8–9, 8–12, 9–16
fill_h
  description, 3–7
  operation, 3–7, 4–36, 4–43, 4–72, 4–73, 4–78, 4–81, 4–83, 4–95, 7–4, 8–9, 9–16
fill_id_h
  description, 3–7
  operation, 3–7, 4–41, 4–43, 4–95, 7–4, 8–9, 9–16
fill_nocheck_h
  description, 3–7
  operation, 7–4, 9–16
FILL_SYN register, 5–95
Floating data types, 2–10
Floating-point unit
  See FPU
FLUSH command, 4–64
Flush protocol, 4–21, 4–25, 4–26, 4–27
  commands, 4–64
  state machines, 4–27
FLUSH timing diagram, 4–66
FLUSH transaction, 4–66
FPU, 2–10
Free-entry queue, 2–36

H

Heat sink, 10–3
Hint bits, 2–11
HWINT_CLR register, 5–28
HW_LD instruction, 6–3
HW_MFPR instruction, 6–3
HW_MTPR instruction, 6–3
HW_REI instruction, 6–3
HW_ST instruction, 6–3

I

Ibox, 2–2, 2–4
  branch prediction, 2–5
  instruction
    decode, 2–4
    issue, 2–4
  instruction translation buffer, 2–7
  interrupts, 2–8
  IPRs, 5–5 to 5–37
    encoding, 5–2
  slotting, 2–22
Icache, 2–13
ICM register, 5–19
ICPERR_STAT register, 5–13
ICSR register, 5–20
IC_FLUSH_CTL register, 5–13
idle_bc_h
  description, 3–8
  operation, 3–6, 4–19, 4–43, 4–44, 4–72, 4–73, 4–74, 4–75, 4–80, 4–83, 4–84, 4–86, 4–88, 7–4, 9–16
IEEE floating-point conformance, A–12
IFAULT_VA_FORM register, 5–11
index_h<25:4>
  description, 3–8
  operation, 3–11, 4–4, 4–15, 4–72, 4–73, 4–89, 7–3, 9–9

Initialization
role of interrupt signals, 4–96
Input clock
ac coupling, 9–6
impedance levels, 9–6
termination, 9–6
Input clocks, 9–4
Instruction
decode, 2–4
issue, 2–4
Instruction cache
See Icache
Instruction fetch/decode unit and branch unit
See Ibox
Instruction issue, 1–3, 2–18
Instructions
classes, 2–20
issue rules, 2–28
latencies, 2–24
MB, 2–12
slotting, 2–20, 2–22
WMB, 2–12, 2–35
Instruction translation buffer, 2–7
See ITB
int4_valid_h<3:0>
description, 3–8
operation, 4–14, 4–41, 4–48, 7–4, 9–17
Integer execution unit
See Ebox
Integer register file
See IRF
Interface restrictions, 4–81
Interface transactions
21164 initiated, 4–36 to 4–52
system initiated, 4–53 to 4–69
Internal processor registers
See IPRs
Interrupts, 4–96 to 4–98
ASTs, 2–9
disabling, 2–9
hardware, 2–8
initialization, 4–96
normal operation, 4–96
priority level, 4–96
Interrupts (cont'd)
software, 2–8
Interrupt signals, 4–96
INTID register, 5–24
INT*nn*, xxiv
INVALIDATE command, 4–55
INVALIDATE timing diagram, 4–60
INVALIDATE transaction, 4–60
IPLR register, 5–23
IPRs
accessibility, 5–1
ALT_MODE, 5–60
ASTER, 5–26
ASTRR, 5–25
BC_CONFIG, 5–84
BC_CONTROL, 5–78
BC_TAG_ADDR, 5–89
CC, 5–61
CC_CTL, 5–62
DC_FLUSH, 5–60
DC_MODE, 5–56
DC_PERR_STAT, 5–50
DC_TEST_CTL, 5–63
DC_TEST_TAG, 5–64
DC_TEST_TAG_TEMP, 5–66
DTB_ASN, 5–38
DTB_CM, 5–39
DTB_IA, 5–52
DTB_IAP, 5–52
DTB_IS, 5–53
DTB_PTE, 5–41
DTB_PTE_TEMP, 5–43
DTB_TAG, 5–40
EI_ADDR, 5–94
EI_STAT, 5–91
EXC_ADDR, 2–19, 5–14
EXC_MASK, 5–17
EXC_SUM, 5–15
FILL_SYN, 5–95
HWINT_CLR, 5–28
ICM, 5–19
ICPERR_STAT, 5–13
ICSR, 2–9, 5–20
IC_FLUSH_CTL, 5–13
IFAULT_VA_FORM, 5–11

IPRs (cont'd)
INTID, 5–24
IPLR, 2–9, 5–23
ISR, 5–29
ITB_ASN, 5–8
ITB_IA, 5–9
ITB_IAP, 5–9
ITB_IS, 5–10
ITB_PTE, 5–6
ITB_PTE_TEMP, 5–9
ITB_TAG, 5–5
IVPTBR, 5–12
MAF_MODE, 5–58
MCSR, 5–54
MM_STAT, 5–44
MVPTBR, 5–49
PAL_BASE, 5–18, 6–3
PMCTR, 5–33
reset state, 7–10
SC_ADDR, 5–75
SC_CTL, 5–69
SC_STAT, 5–72
SIRR, 5–27
SL_RCV, 5–32
SL_XMIT, 5–31
VA, 5–46
VA_FORM, 5–47
IRF, 2–9
irq_h<3:0>
description, 3–9
operation, 2–8, 2–9, 4–6, 4–9, 4–97, 5–30, 7–4, 9–16
ISR register, 5–29
Issue rules, 2–28
Issuing rules, 2–20 to 2–29
ITB, 2–7
ITB_ASN register, 5–8
ITB_IAP register, 5–9
ITB_IA register, 5–9
ITB_IS register, 5–10
ITB_PTE register, 5–6
ITB_PTE_TEMP register, 5–9
ITB_TAG register, 5–5
IVPTBR register, 5–12

L

Latencies, 2–24
Literature, E–2
Live lock
cache conflict, 4–28
Load-after-store trap, 2–29
Load instructions
noncacheable space, 2–31
Load miss, 2–30
LOCK command, 4–37
Lock mechanisms, 4–30
LOCK timing diagram, 4–50
LOCK transaction, 4–50
Logic symbol, 3–1

M

MAF, 2–11, 2–30 to 2–33, 4–14
entries, 2–32
entry, 2–33
rules, 2–30
MAF_MODE register, 5–58
MB instruction, 2–12, 4–52
Mbox, 2–2, 2–10
address translation, 2–10
data translation buffer, 2–11
IPRs, 5–38 to 5–67
encoding, 5–3
load instruction, 2–11
miss address file, 2–11
store execution, 2–33 to 2–34
store instructions, 2–12
write buffer, 2–12
write buffer address file, 2–35
mch_hlt_irq_h
description, 3–9
operation, 2–8, 4–8, 4–97, 7–4, 9–16
MCSR register, 5–54
Memory address translation unit
See Mbox

MEMORY BARRIER command, 4–37
when to use, 4–52
Memory regions, physical, 4–12
Merge
write buffer, 4–14
Merging
loads to noncacheable space, 2–31
rules, 2–30
Microarchitecture, 2–2 to 2–14
Miss address file
See MAF
MM_STAT register, 5–44
Multiple instruction issue, 2–4
MVPTBR register, 5–49

N

Noncached read operations, 4–14
Noncached write operations, 4–14
Nonissue conditions, 2–20
NOP command, 4–37, 4–55, 4–64

O

Operating temperature, 10–1
Ordering products, E–1
osc_clk_in_h,l
description, 3–9
operation, 3–4, 4–5, 4–11, 7–3, 7–5, 9–2, 9–4, 9–5, 9–6, 9–15, 9–23, 12–3

P

Page table entry
See PTE
PAL
restrictions, 5–101
PALcode, 1–2
environment instructions, 6–7
invoke, 6–3
PALmode, 6–2
environment, 6–2
PALshadow registers, 5–99
PALtemp IPRs, 5–99
encoding, 5–2
PAL_BASE register, 5–18, 6–3
Parity, 4–92
Parts
ordering, E–1
Pending-request queue, 2–36
Performance counters, 2–38
perf_mon_h
description, 3–10
operation, 2–38, 5–36, 7–5, 9–16
Physical address considerations, 4–12
Physical address regions, 4–12
Physical memory regions, 4–12
Pipeline, wave, 4–33
Pipeline organization, 2–14 to 2–20
Pipelines, 2–9
bubbles, 2–20
examples, 2–14
floating add, 2–16
integer add, 2–16
load (Dcache hit), 2–16
load (Dcache miss), 2–17
store (Dcache hit), 2–17
instruction issue, 2–18
stages, 2–14, 2–18
stall, 2–18, 2–20
PMCTR register, 5–33
port_mode_h<1:0>
description, 3–10
operation, 7–5, 9–16, 12–1, 12–2
Power supply
considerations, 9–24
decoupling, 9–24
sequencing, 9–25
Private Bcache transactions
21164 to Bcache, 4–31 to 4–35
Privileged architecture library code
See PALcode
Producer–consumer dependencies, 2–24
Producer–producer dependencies, 2–24
Producer–producer latency, 2–27
PTE, 2–8, 2–11

pwr_fail_irq_h
description, 3–10
operation, 2–8, 4–8, 4–97, 7–4, 9–16

Q

Queues
entry-pointer, 2–36

R

Race condition
21164 and system, 4–83
Race example
idle_bc_h and **cack_h**, 4–86
READ command, 4–64
READ DIRTY command, 4–55
READ DIRTY/INVALIDATE command, 4–56
READ DIRTY/INVALIDATE transaction, 4–58
READ DIRTY timing diagram, 4–58
READ DIRTY transaction, 4–58
READ MISS0 command, 4–38
READ MISS1 command, 4–38
READ MISS MOD0 command, 4–38
READ MISS MOD1 command, 4–38
READ MISS MOD STC0 command, 4–39
READ MISS MOD STC1 command, 4–39
READ MISS no Bcache timing diagram, 4–40
READ MISS timing diagram, 4–41
READ MISS transaction, 4–41
READ MISS transaction (no Bcache), 4–40
READ MISS with **idle_bc_h** asserted example, 4–88
READ MISS with victim abort example, 4–89
READ MISS with victim example, 4–84
READ MISS with victim timing diagram, 4–45, 4–46
READ MISS with victim transaction, 4–43
READ timing diagram, 4–68
READ transaction, 4–68
Read/write spacing
data bus contention, 4–71
Reference clock, 4–8, 4–9
example 1, 4–10
example 2, 4–11
examples, 4–9
ref_clk_in_h
description, 3–10
operation, 4–5, 4–8, 4–9, 4–10, 4–11, 4–12, 7–3, 9–4, 9–10, 9–13, 9–15, 9–16, 9–23
Registers
See also IPRs
accessibility, 5–1
integer, 2–9
PALshadow, 2–9, 5–99
PALtemp, 5–99
Related documentation, E–2
Replay traps, 2–29 to 2–30
as aborts, 2–19
load instruction, 2–12, 2–33
load-miss-and-use, 2–19
Reset
forcing, 4–95
Resource conflict, 2–20
Restrictions
interface, 4–81

S

Scache, 2–13
block size, 4–15
scache_set_h<1:0>
description, 3–10
operation, 4–16, 4–18, 7–4, 9–17
Scheduling rules, 2–20 to 2–29
SC_ADDR register, 5–75
SC_CTL register, 5–69
SC_STAT register, 5–72
Second-level cache
See Scache
Semiconductor Information Line, E–1
Serial read-only memory
See SROM

SET DIRTY command, 4–37
SET DIRTY timing diagram, 4–50
SET DIRTY transaction, 4–50
SET SHARED command, 4–55
SET SHARED timing diagram, 4–62
SET SHARED transaction, 4–62
shared_h
  description, 3–10
  operation, 7–4, 9–16
Signal descriptions, 3–3 to 3–15
SIRR register, 5–27
Slotting, 2–22
SL_RCV register, 5–32
SL_XMIT register, 5–31
Specifications
  mechanical, 11–1
SROM, 2–14
srom_clk_h
  description, 3–10
  operation, 5–31, 7–5, 7–6, 9–17, 9–21, 9–22, 12–1
srom_data_h
  description, 3–10
  operation, 5–32, 7–5, 7–6, 7–7, 9–16, 9–22, 12–1
srom_oe_l
  description, 3–10
  operation, 7–5, 7–6, 9–17, 12–1
srom_present_l
  description, 3–10
  operation, 7–5, 7–6, 9–16, 9–20, 9–21, 12–1
Store instruction, 2–12
  execution, 2–33
st_clk_h
  description, 3–11
Superpages, 2–8
System clock, 4–6
  delayed, 4–8
System clock delay, 4–8
System interface, 4–2
  addresses, 4–4
  commands, 4–4
System interface introduction, 4–2 to 4–4
system_lock_flag_h
  description, 3–11
  operation, 4–30, 7–4, 9–16
sys_clk_out1_h,l
  description, 3–11
  operation, 3–9, 3–11, 4–2, 4–5, 4–6, 4–8, 4–9, 4–11, 4–57, 4–65, 5–87, 7–3, 9–4, 9–10, 9–11, 9–13, 9–15, 9–23
sys_clk_out2_h,l
  description, 3–11
  operation, 3–9, 3–10, 3–11, 4–5, 5–87, 7–3, 7–5, 9–5
sys_mch_chk_irq_h
  description, 3–11
  operation, 2–8, 4–8, 4–97, 7–4, 9–16
sys_reset_l
  description, 3–11
  operation, 4–96, 7–1, 7–2, 7–3, 7–5, 7–6, 7–13, 9–16, 9–19, 9–20

T

Tag store, duplicate, 4–15
tag_ctl_par_h
  description, 3–11
  operation, 4–79, 4–94, 7–3, 9–17, 9–18
tag_data_h<38:20>
  description, 3–11
  operation, 4–15, 4–19, 4–75, 4–94, 7–3, 9–19
tag_data_par_h
  description, 3–11
  operation, 4–19, 4–43, 4–94, 7–3, 9–19
tag_dirty_h
  description, 3–11
  operation, 3–11, 4–19, 4–41, 4–43, 4–79, 4–94, 7–3, 9–17, 9–18
tag_ram_oe_h
  description, 3–11
  operation, 4–32, 4–79, 7–3, 9–19
tag_ram_we_h
  description, 3–12
  operation, 4–34, 7–3, 9–19

tag_shared_h
description, 3–12
operation, 3–11, 4–19, 4–43, 4–57, 4–65, 4–79, 4–94, 7–3, 9–17, 9–18
tag_valid_h
description, 3–12
operation, 3–11, 4–19, 4–43, 4–94, 7–3, 9–18, 9–19
tck_h
description, 3–12
operation, 7–5, 9–24, 12–1, 12–2
tdi_h
description, 3–12
operation, 7–5, 9–4, 9–24, 12–1, 12–2, 12–7
tdo_h
description, 3–12
operation, 7–5, 9–24, 12–1, 12–2, 12–7
Technical support, E–1
Temperature, 10–1
temp_sense
description, 3–12
operation, 7–5, 9–4
Terminology, xxii to xxvii
test_status_h<1:0>
description, 3–12
operation, 5–22, 7–5, 7–6, 9–19, 12–1, 12–6, 12–7
Thermal design considerations, 10–4
Thermal heat sink, 10–3
Thermal management, 10–1
Thermal operating temperature, 10–1
Timing diagrams
Bcache hit under READ MISS, 4–90
Bcache read, 4–32
Bcache write, 4–34
bus contention, 4–70
FILL, 4–78, 4–79
FILL to private read or write, 4–80
FLUSH, 4–66
idle_bc_h and **cack_h** race, 4–86
INVALIDATE, 4–60
LOCK, 4–50
READ, 4–68
READ DIRTY, 4–58
Timing diagrams (cont'd)
READ MISS, 4–41
READ MISS completed first—victim buffer, 4–76
READ MISS—no Bcache, 4–40
READ MISS second—no victim buffer, 4–77
READ MISS with **idle_bc_h** asserted, 4–88
READ MISS with victim, 4–45, 4–46, 4–84
READ MISS with victim abort, 4–89
SET DIRTY, 4–50
SET SHARED, 4–62
using **data_bus_req_h**, 4–74
using **idle_bc_h** and **fill_h**, 4–73
wave pipeline, 4–33
WRITE BLOCK, 4–49
tms_h
description, 3–12
operation, 7–5, 9–4, 9–24, 12–1, 12–2, 12–4
Transactions
FILL, 4–43
FLUSH, 4–66
INVALIDATE, 4–60
LOCK, 4–50
READ, 4–68
READ DIRTY, 4–58
READ DIRTY/INVALIDATE, 4–58
READ MISS, 4–41
READ MISS (no Bcache), 4–40
READ MISS with victim, 4–43
SET DIRTY, 4–50
SET SHARED, 4–62
system initiated, 4–53
WRITE BLOCK, 4–48
WRITE BLOCK LOCK, 4–48
Traps
load-after-store, 2–29
load-miss-and-use, 2–28
replay, 2–19, 2–29, 2–33
Tristate
BCACHE VICTIM to fill, 4–75
FILL to private Bcache read or write, 4–80

Tristate (cont'd)
overlap, 4–70, 4–75
READ or WRITE to fill, 4–75
system Bcache command to fill, 4–78
trst_l
description, 3–12
operation, 7–5, 7–13, 9–24, 12–1, 12–2, 12–3

V

VA register, 5–46
VA_FORM register, 5–47
Victim buffers, 4–18, 4–44
victim_pending_h
description, 3–12
operation, 4–16, 4–18, 4–38, 4–44, 7–4, 9–17

W

Wave pipeline, 4–33
WMB instruction, 2–12, 2–35
Write-after-write conflicts
See Producer–producer dependencies
See Producer–producer latency
WRITE BLOCK command, 4–37
WRITE BLOCK command acknowledge, 4–81
WRITE BLOCK LOCK command, 4–38
WRITE BLOCK LOCK restriction, 4–82
WRITE BLOCK LOCK transaction, 4–48
WRITE BLOCK timing diagram, 4–49
WRITE BLOCK transaction, 4–48
Write buffer, 2–12, 2–35 to 2–37
entry processing, 2–36
Write invalidate protocol, 4–21, 4–22
commands, 4–55
states, 4–23
systems, 4–22
Write ordering, 2–37