Alpha 21066 and Alpha 21066A Microprocessors

Hardware Reference Manual

Order Number: EC–QC4GB–TE

Revision/Update Information: This document supersedes the *Alpha 21066, 21066A, and 21068 Microprocessors Hardware Reference Manual* (EC–QC4GA–TE).

Digital Equipment Corporation
Maynard, Massachusetts

January 1996

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

Tables

Preface

This manual describes the architecture, internal design, and external interface of the Alpha 21066 and Alpha 21066A microprocessors. The *Alpha 21066/21066A Microprocessors Data Sheet* describes electrical, mechanical, and thermal characteristics.

Audience

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

Manual Organization

This manual contains the following chapters and appendixes:

- Chapter 1, Introduction, provides an overview of the Alpha architecture. It also briefly describes the features of the Alpha 21066 and Alpha 21066A microprocessors.
- Chapter 2, Internal Architecture, describes the Alpha 21066 and Alpha 21066A microprocessor architecture. It also describes the instruction pipeline and defines the scheduling and dual-issue rules.
- Chapter 3, Privileged Architecture Library Code, describes the microprocessor’s privileged architecture library code (PALcode).
- Chapter 4, Internal Processor Registers, describes the instruction fetch and decode unit (IDU) registers, load and store unit (LSU) registers, and PAL_TEMP registers.
- Chapter 5, Memory Controller, describes the memory controller, supported memories, memory operations, error handling, graphics operations, registers, cycle timing, and interface signals.
- Chapter 6, I/O Controller, describes the I/O controller (IOC), I/O operations, and IOC error handling, registers, and interface signals.

- Chapter 7, Clocks, describes the clock interface, phase-locked loop (PLL), clock frequency selection, and noise reduction requirements.
- Chapter 8, JTAG Test Port, describes the implementation of the joint testing action group (JTAG) test port.
- Chapter 9, SROM Interface and Icache Initialization, describes the instruction cache (Icache) initialization, using the serial ROM (SROM). It also describes how the SROM port can be used as a serial terminal port after the Icache is initialized.
- Appendix A, 21066A Differences, describes how the Alpha 21066A microprocessor differs from the Alpha 21066 microprocessor.
- Appendix B, Pin Summary, summarizes the following pins: memory controller; IOC; clock; JTAG; and instruction reference, interrupt, and SROM interface.
- Appendix C, Internal Register Summary, summarizes the Alpha 21066 internal registers.
- Appendix D, Technical Support and Ordering Information, describes how to obtain information and technical support, and order products and associated literature.

Conventions

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

Abbreviations

The following abbreviations apply to binary multiples and register access:

- **Binary multiples**

  The abbreviations K, M, and G (kilo, mega, and giga) represent binary multiples and have the following values:

K =	2^{10} (1024)
M =	2^{20} (1,048,576)
G =	2^{30} (1,073,741,824)

  For example:

2 KB =	2 kilobytes =	2×2^{10} bytes
4 MB =	4 megabytes =	4×2^{20} bytes
8 GB =	8 gigabytes =	8×2^{30} bytes

- **Register Access**

  The abbreviations used to indicate the type of access to register fields and bits have the following definitions:

  IGN — Ignore

  Bits and fields specified as IGN are ignored when written.

  MBZ — Must Be Zero

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

  RAX — Read As UNDEFINED

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

  RAZ — Read As Zero

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

  RC — Read Clears

  Bits and fields specified as RC are cleared when read. Unless otherwise specified, such fields cannot be written.[1]

  RES — Reserved

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

  RO — Read Only

  Bits and fields specified as RO can be read and are ignored (not written) on write operations.[1]

  RW — Read and Write

  Bits and fields specified as RW can be read and written. Writing a zero clears the bits.[1]

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

R/W1C — Read and Write One to Clear

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

WA — Write Always Clears

Bits and fields specified as WA can be read. Any write clears the bits.

WO — Write Only

Bits and fields specified as WO can be written but not read. In the I/O controller the results of reading such registers are UNPREDICTABLE. Writing a zero clears the bits.[1]

X

Bits and fields specified as X return UNPREDICTABLE results when read and are ignored on write operations.[1]

Aligned and Unaligned

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

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

Bit Notation

Multiple-bit fields can include contiguous and noncontiguous bits contained in angle brackets (<>). Multiple contiguous bits are indicated by a pair of numbers separated by a colon (:). For example, <9:7,5,2:0> specifies bits 9,8,7,5,2,1, and 0. Similarly, single bit positions are frequently indicated with angle brackets. For example, "The PAL bit (<0>) indicates PALmode to the hardware," indicates that the PAL bit is bit 0 in a register.[1]

Caution

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

Chip

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

Data Units

The following data unit terminology is used throughout this manual.

Term	Words	Bytes	Bits	Other
Nibble	¼	½	4	—
Byte	½	1	8	—
Tribyte	1½	3	24	—
Word	1	2	16	—
Longword	2	4	32	—
Quadword	4	8	64	Single read fill—The cache space that can be filled in a single read access. It takes four read accesses to fill an onchip (I or D) cache line.
Octaword	8	16	128	—
Hexword	16	32	256	Cache block, cache line—The space allocated to a single Bcache block.

External

Unless otherwise stated, throughout this manual, the term external means not contained in the Alpha 21066 or Alpha 21066A microprocessor.

Microprocessor

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

Note

Notes emphasize particularly important information.[1]

Numbering

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

Ranges and Extents

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

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

Security Holes

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

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

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

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

Signal Names

Signal names are printed in lowercase, boldfaced type. The names of low-asserted signals carry the suffix **_l**. The names of high-asserted signals have no suffix. For example, **pll_clk_in** is a high-asserted signal, and **pll_clk_in_l** is a low-asserted signal.

UNPREDICTABLE and UNDEFINED

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

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

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

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

A result specified as UNPREDICTABLE might acquire an arbitrary value subject to a few constraints. Such a result might 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 might be unchanged from their previous values. Operations that produce UNPREDICTABLE results might also produce exceptions.

UNPREDICTABLE results must not be security holes.

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

Also, operations that might produce UNPREDICTABLE results must not:

- Write or modify the contents of memory locations or registers that are inaccessible to the current process in the current access mode.
- Halt or hang the system or any of its components.

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

- The value of a register in another process
- The contents of processor temporary registers left behind by some previously running process
- A sequence of actions of different processes

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

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

Operations specified as UNDEFINED might vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. The operation might vary in effect from nothing to stopping system operation. UNDEFINED operations must not cause the processor to hang; that is, reach an unhalted state from which there is no transition to a normal state in which the machine executes instructions.

1 Introduction

This chapter is an overview of the Alpha architecture as it is implemented in the Alpha 21066 microprocessor (21066) and the Alpha 21066A microprocessor (21066A). It also lists the major features of the microprocessors.

Note

The 21066–166 and 21066–100 are functionally identical—they differ only in clock frequency (21066–166 at 166 MHz and 21066–100 at 100 MHz).

The upgraded 21066A versions (21066A–266 at 266 MHz, 21066A–233 at 233 MHz, and 21066A–100 at 100 MHz) are functionally identical and differ only in clock frequency.

The functionality described for the 21066 is true for the 21066A, except where the differences are explicitly detailed.

1.1 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 multiple operating system support.

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 instructions or store instructions. 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 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. Using resources in this way makes it easy to build processors that issue multiple instructions every CPU cycle.

The microprocessor 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 machine-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 does byte shifting and masking with normal 64-bit register-to-register instructions; it does not include single byte load and store instructions. The software developer must determine the precision of arithmetic traps.

For more information about the Alpha architecture, see the *Alpha Architecture Reference Manual*.

1.2 Microprocessor Features

The 21066 implementation of the Alpha architecture is a 0.65-micron, CMOS-based, superscalar, superpipelined processor using dual-instruction issue. The 21066A is implemented in the same way, with 0.50-micron technology.

The chip incorporates a high level of system integration for best-in-class system performance in cost-focused (21066–166, 21066A–233, and 21066A–266) or embedded (21066–100 and 21066A–100) applications. It integrates onchip, fully pipelined integer and floating-point processors, a high-bandwidth memory controller, an industry-standard I/O controller (IOC), an embedded graphics accelerator, internal instruction and data caches, and external cache control.

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

Microprocessor features include the following:

- Fully pipelined, 64-bit RISC architecture
- Supported by multiple operating systems:
  - Microsoft Windows NT
  - OSF/1
  - OpenVMS
  - VxWorks for Alpha
- Best-in-class performance:
  - 266 MHz (21066A–266), 233 MHz (21066A–233), 166 MHz (21066–166), or 100 MHz (21066–100 and 21066A–100) operation
  - Superscalar, superpipelined (dual issue)
  - Peak instruction execution rate of 532 MIPS (21066A–266), 466 MIPS (21066A–233), 330 MIPS (21066-166), or 200 MIPS (21066–100 and 21066A–100)
  - 0.50-micron (21066A) and 0.65-micron (21066) CMOS technology
- Pipelined onchip floating-point unit. IEEE single-precision and double-precision, VAX F_floating and G_floating, longword and quadword data types. Limited support for VAX D_floating data type.
- Onchip demand-paged memory management unit:
  - 12-entry instruction stream translation buffer (ITB) with eight entries for 8-KB pages and four entries for 4-MB pages
  - 32-entry data stream translation buffer (DTB) with each entry able to map a single 8-KB, 64-KB, 512-KB, or 4-MB page
  - Superpage mapping
- Onchip high-bandwidth memory controller:
  - Full 64-bit memory data path
  - Dynamic RAM (DRAM) controller with fully programmable timing
  - Optional error correction code (ECC)
  - Support for up to four banks of memory with programmable timing for each bank
  - RAS/CAS memory bus to industry-standard, single inline memory modules (SIMMs)

- Embedded graphics accelerator:
  - Onchip support for direct connection to a video RAM (VRAM) frame buffer
  - High-bandwidth memory data path to VRAM
- PCI IOC:
  - 32-bit multiplexed address and data
  - Industry-standard interface
  - No glue logic needed to connect PCI-compliant chips
  - Usable bandwidth in excess of 100 MB/s
  - Burst mode read and write operations
  - Asynchronous operation to CPU
  - Multimaster with peer-to-peer access
  - Alpha I/O-compliant
- Onchip, 8-KB, direct-mapped, write-through data cache (Dcache)
- Onchip, 8-KB, direct-mapped, instruction cache (Icache)
- Onchip control for optional, external, write-back backup cache (Bcache):
  - Direct connection to external static RAMs (SRAMs) without any external logic
  - Programmable cache size and speed
- Built-in phase-locked loop (PLL):
  - Frequency multiplier for low cost input clock
  - Programmable multiplier values
- Serial ROM (SROM) interface:
  - Loads instruction cache after reset
  - Supports software-controlled serial port after initialization
- Chip- and module-level test support — JTAG (IEEE 1149)
- 3.3-V supply voltage interfaces directly to 5-V logic

In addition, the 21066A features:

- Software-controllable power management
  - Programmable clock divider
  - Programmable override for interrupts, DMA, or both
- Parity protection on internal caches

2 Internal Architecture

This chapter describes the chip's microarchitecture. Much of the hardware design is based on specific PAL functionality. The combination of microarchitecture and PALcode (described in Chapter 3) defines the chip's implementation of the Alpha architecture. This chapter also describes the instruction pipeline and defines the scheduling and dual-issue rules. This chapter is an overview of the internal architecture and does not describe the chip in great detail.

2.1 Overview

The chip consists of a core central processing unit (CPU), a memory controller, and an I/O controller (IOC). The chip also contains an instruction cache (Icache), a data cache (Dcache), and a serial read-only memory (SROM) interface. Figure 2–1 is a block diagram of the chip.

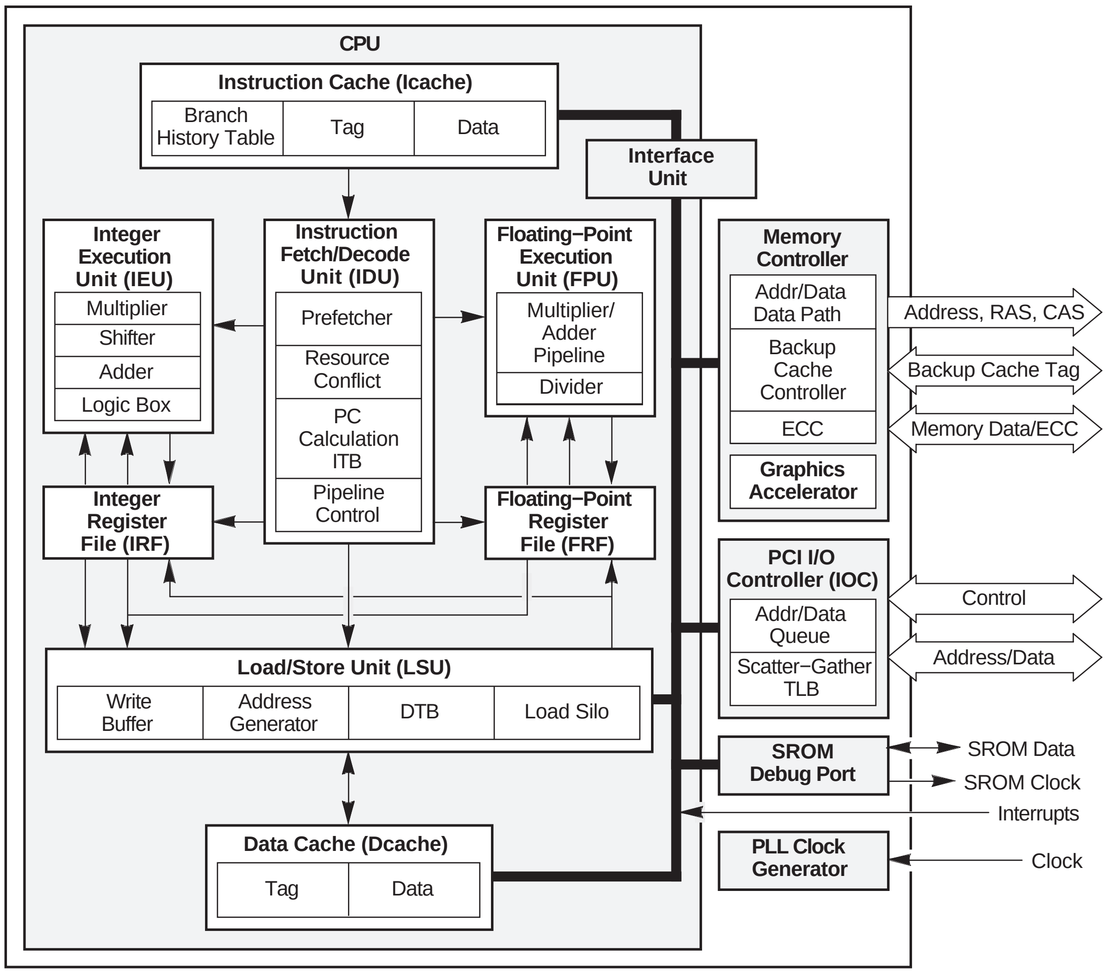
The memory controller interfaces to the system memory and the optional, external, backup cache (Bcache). It also contains the embedded graphics accelerator. The memory controller and interface are described in Chapter 5.

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

The SROM interface provides the initialization data load path from the SROM to the Icache. Following initialization, this interface can be used as a diagnostic port through the use of PALcode. The SROM interface and operation are described in Chapter 9.

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

Figure 2–1 Microprocessor Block Diagram



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

- IEU
- LSU
- FPU
- Branch unit

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

2.2 Instruction Fetch and Decode Unit

The primary function of the IDU is to issue instructions to the IEU, LSU, and FPU. The IDU includes the following functions:

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

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

- If resources are available, both instructions are issued.
- If resources are available only for the second instruction, neither the first nor the second instruction is issued.
- If the IDU issues only the first of a pair of instructions, it does not advance another instruction to attempt a dual issue; dual issue is attempted only on aligned quadword (8-byte) pairs.

Section 2.8.4 describes instructions that can be dual issued.

2.2.1 Branch Prediction Logic

The branch unit, or prediction logic, is also part of the IDU. The microprocessor offers a register-selectable choice of branch prediction strategies (Section 4.1.7). Each instruction location in the Icache includes a single history bit to record the outcome of branch instructions. This information can be used to predict the result when the branch instruction is next executed. The prediction for the first execution of a branch instruction is based on the sign of the displacement field within the branch instruction, as follows:

- If the sign is negative, the instruction prefetcher predicts the conditional branches to be taken.
- If the sign is positive, the instruction prefetcher predicts that the conditional branches are not to be taken.
- Alternatively, if the history table is disabled, all branch prediction can be based on the sign of the displacement field.

The microprocessor provides a 4-entry subroutine return stack that is controlled by the hint bits in the BSR, HW_REI, JMP, JSR, RET, and JSR_COROUTINE instructions. The chip also allows all branch prediction hardware to be disabled.

Note

The 21066A implements an improved branch prediction scheme. For details, see Section A.4.

2.2.2 Instruction Translation Buffers

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

- An 8-entry, small-page ITB for 8-KB pages
- A 4-entry, large-page ITB for 4-MB (512 × 8 KB) pages

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

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

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

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

The ITBs support a single address space number (ASN) through the PTE ASM bit. Each PTE entry in the ITB contains an ASM bit. Write operations to the ITBASM register invalidate all entries that do not have the ASM bit set. This is a simple way to preserve entries that map operating system regions while invalidating all other entries. (The ITBASM register is described in Section 4.1.5.)

For more information about TBs and PTEs, see the *Alpha Architecture Reference Manual*.

2.2.3 Interrupt Logic

The IDU exception logic supports three sources of interrupts:

- Hardware interrupts
  - Three level-sensitive hardware interrupts are sourced by pins **irq<2:0>**.
  - Two internally generated interrupts respond to external interface error conditions. These are sourced by the memory controller error status register (ESR, Section 5.6.6) and the I/O controller status register 0 (IOC_STAT0, Section 6.4.3).
- Software interrupts

  The 15 prioritized software interrupts are sourced by the software interrupt request register (SIRR, Section 4.1.13.2).
- Asynchronous system traps (ASTs)

  Four ASTs are available, one for each processor mode: user, supervisor, executive, and kernel. These traps are sourced by the asynchronous trap request register (ASTRR, Section 4.1.13.3).

In addition to the three level-sensitive hardware interrupts from the **irq<2:0>** pins, error conditions detected in the memory controller and the IOC can interrupt the CPU.

The memory controller generates an interrupt if it detects any error condition, such as a nonexistent memory access, a missed refresh cycle, an uncorrectable ECC error, and a Bcache parity error. To return the memory subsystem to correct operation, software must clear these error conditions by writing to the memory controller ESR.

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

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

All interrupts can be independently masked using onchip enable registers. In addition, AST interrupts are qualified by the current processor mode and the current state of software interrupt enable register bit 2 (SIER, Section 4.1.13.5). Providing distinct enable bits for each independent interrupt source allows a flexible, software-controlled interrupt priority scheme to be implemented in PALcode or the operating system.

For example, the microprocessor can support a 5-level interrupt priority scheme by defining a distinct state of the hardware interrupt enable register (HIER, Section 4.1.13.4) for each interrupt priority level (IPL). The state of the HIER determines the current interrupt priority. The lowest IPL is produced by enabling all five interrupts, for example bits <5:1>. The next IPL is produced by enabling bits <5:2> and so on, to the highest IPL that is produced by enabling only bit 5 and disabling bits <4:1>.

When all interrupt enable bits are cleared, the processor cannot be interrupted from the hardware interrupt request register (HIRR, Section 4.1.13.1). Each state (bits <5:1> enabled through only bit 5 enabled) represents an individual IPL. If these states are the only states allowed in the HIER, a 5-level hardware interrupt priority scheme can be controlled entirely by PAL software. The scheme is extensible to provide multiple interrupt sources at the same IPL by grouping enable bits. Groups of enable bits must be set and cleared together to support multiple interrupts of equal priority level. This method reduces the total available number of distinct levels.

Because enable bits are provided for all hardware, software, and AST interrupt requests, a priority scheme can span all sources of processor interrupts. Additional restrictions are placed on AST interrupt requests.

Four AST interrupts are provided; one for each processor mode. The AST interrupt requests are qualified such that AST requests that correspond to a given mode are blocked when the processor is in a higher mode, regardless of the state of the AST interrupt enable register (ASTER, Section 4.1.13.6). In addition, all AST interrupt requests are qualified by SIER bit 2.

When the processor receives an interrupt request and that request is enabled and the processor is not currently executing PALcode, the hardware reports or delivers an interrupt to the exception logic. Before vectoring to the interrupt-service PAL dispatch address, the pipeline is completely drained and all outstanding Dcache fills are completed. The restart address is saved in the exception address register (EXC_ADDR, Section 4.1.8) and the processor enters PALmode. The cause of the interrupt can be determined by examining the state of the interrupt request registers.

Usually, the interrupts from the **irq<2:0>** pins can be grouped as follows:

- Interval timer interrupts
- Halt, powerfail, ISA and EISA nonmaskable interrupts, and the PCI **serr_l** signal
- ISA and EISA interrupt requests, and PCI interrupt requests

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

2.2.4 Performance Counters

An onchip performance recording mechanism counts various hardware events and causes an interrupt upon counter overflow. Interrupts are triggered six cycles after the event, and therefore, the exception program counter (PC) might not reflect the exact instruction causing counter overflow.

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

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

In addition, two external interface events (such as direct memory access [DMA] transactions, Bcache accesses, and so on), can be counted by programming the memory controller error address register (EAR, Section 5.6.7).

See Section 4.1.14 for more information about the performance counters.

2.3 Integer Execution Unit

The IEU contains the 64-bit integer execution data path, which includes the following functions:

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

The integer multiplier retires 4 bits per cycle.

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

2.4 Load and Store Unit

The LSU contains four major sections:

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

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

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

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

The internal processor registers (IPRs) are described in Chapter 4.

2.4.1 Data Translation Buffer

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

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

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

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

Each PTE in the DTB contains an ASM bit. The DTB supports a single ASN with the PTE ASM bit. Write operations to the DTBASM register invalidate all entries that do not have the ASM bit set. This is a simple way to preserve entries that map operating system regions while invalidating all other entries. (The DTBASM register is described in Section 4.2.6.)

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

PALcode fills and maintains the DTB. (The flow for a DTB miss is described in Chapter 3.) Note that the DTB can be filled in kernel mode by setting the hardware enable (HWE) bit in the ICCSR (Section 4.1.7).

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

2.4.2 Load Silos

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

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

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

To improve performance, the IDU can restart the execution of LSU-directed instructions before the last pending Dcache fill is complete. Dcache fill transactions result in four data transfers from the memory controller to the Dcache. These transfers can each be separated by one or more cycles, depending on the characteristics of the Bcache and memory subsystems.

In the first of these four transfers, the memory controller sends the quadword of the fill block originally requested by the CPU (it can always do this for read operations that hit in the Bcache). Therefore, the pending load instruction that requested the Dcache fill can finish before the Dcache fill finishes.

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

2.4.3 Write Buffer

The LSU write buffer has two purposes:

- The CPU can generate store data faster than the Bcache subsystem can accept the data, which causes CPU stall cycles. The write buffer is a finite, high-bandwidth resource that receives store data and reduces the number of possible CPU stall cycles to a minimum.
- The write buffer also attempts to aggregate store data into aligned, 32-byte cache blocks to maximize the rate at which the microprocessor can write data into the Bcache.

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

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

Each of the four write buffer entries can store up to 32 bytes (four quadwords). The buffer has a *head* pointer and *tail* pointer. The buffer puts new commands into empty tail entries and takes commands out of nonempty head entries. The head pointer increments when an entry is unloaded to the memory controller, and the tail pointer increments when new data is put into the tail entry. The head and tail pointers point to the same entry only when either all four entries are empty or no entries are empty.

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

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

- The write buffer contains at least two valid entries.
- The write buffer contains one valid entry and at least 256 CPU cycles have elapsed since the execution of the last write-buffer-directed instruction.
- The write buffer contains an MB instruction.
- The write buffer contains an STQ/C or STL/C instruction.
- A load miss that requires the write buffer to be flushed is pending to an address that is currently valid in the write buffer. The write buffer is completely flushed regardless of which entry matches the address.

Note

The 21066A implements revised write buffer unload logic. For details, see Section A.5.

2.4.3.1 Memory Barrier Instruction

The CPU will not execute load or store instructions that occur after an MB instruction until all preceding load or store instructions have been completed. The instructions are completed when:

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

A load or store instruction is considered complete even if error status occurs. (See Chapter 6 for more information.)

2.5 Floating-Point Unit

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

The FPU includes the following functions:

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

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

The FPU can accept an instruction every cycle, with the exception of floating-point divide instructions. The latency for data-dependent, nondivide instructions is six cycles. (For more information about instruction timing, see Section 2.7.)

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

Note

The 21066A has new floating-point divide hardware that greatly reduces average divide latency. For more detailed information, refer to Section A.3.

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

2.5.1 IEEE Floating-Point Conformance

The microprocessor supports IEEE floating-point operations as specified by the Alpha architecture. Support for a complete implementation of the *IEEE Standard for Binary Floating-Point Arithmetic (754-1985)* is provided by a combination of hardware and software as described in the *Alpha Architecture Reference Manual*. Additional information that provides guidelines for writing code supporting precise exception handling (necessary for complete conformance to the standard) is in the *Alpha Architecture Reference Manual*.

The microprocessor implements the following functions:

- When operating without the /UNDERFLOW modifier, the microprocessor replaces underflow results with exact zero, whether or not the correct result would have been negative zero as defined in the IEEE standard. This Alpha architecture value-added behavior improves performance over either hardware or software denormal handling.

  When strict IEEE compliance is required, the /UNDERFLOW modifier is necessary and the software must provide the correct result (including negative zero).
- The microprocessor supports infinity operands only when used in the CMPT instruction. Other instructions using infinity operands cause invalid operation (INV) arithmetic traps.
- NaN, denormal, or infinity (except when used in CMPT) input operands produce INV arithmetic traps when used with arithmetic operation instructions. CPYSE, CPYSN, FCMOV, MF_FPCR, and MT_FPCR are not arithmetic operations and will pass NaN, denormal, and infinity values without initiating arithmetic traps. Input operand traps take precedence over arithmetic result traps.
- The microprocessor will not produce a NaN, denormal, or infinity result.
- The microprocessor supports IEEE normal and chopped rounding modes in hardware. Instructions designating plus infinity and minus infinity rounding modes cause precise exceptions to the OPCDEC PAL entry point. This implies that the EXC_ADDR register will be loaded with the address of the faulting instruction and all following instructions will be aborted. Table 3–5 describes the PALcode entry points.
- The DIVS and DIVT instructions with the /INEXACT modifier report an inexact (INE) arithmetic trap on all results of operations that do not involve NaN, infinity, or denormal input operands. Operations using NaN, infinity, and denormal input operands generate INV arithmetic traps.

- Floating-point exceptions generated by the microprocessor are recorded in the following places:
  - Whether or not the corresponding trap is enabled (through the instruction modifiers), the occurrence of all detected exceptions other than SWC are recorded in the FPCR (accessible by the MT_FPCR and MF_FPCR instructions) records. Because this register can be cleared only with an explicit clear command (MT_FPCR), the exception information it records is a summary of all exceptions that have occurred since the last clear operation.
  - If an exception is detected and the corresponding trap is enabled, the microprocessor records the condition in the exception summary register (EXC_SUM, Section 4.1.9) and initiates an arithmetic trap. As a special case, in order to support inexact exception behavior with the DIVS/I and DIVT/I instructions, the FPCR does not record an inexact exception, although the microprocessor always sets the INE bit in the EXC_SUM register during these instructions. This behavior allows software to emulate the division instructions with accurate reporting of potential inexact exceptions.

  For more information about the FPCR, see the *Alpha Architecture Reference Manual*.

2.6 Internal Cache Organization

All memory cells in the onchip Dcache and Icache are fully static, 6-transistor CMOS structures.

2.6.1 Data Cache

The 8-KB data cache (Dcache) is a write-through, direct-mapped, read-allocate physical cache with 32-byte blocks. When a PCI device writes to cacheable memory, the Dcache block corresponding to the memory address is set invalid to maintain cache coherency. The 21066A has longword parity protection.

2.6.2 Instruction Cache

The 8-KB instruction cache (Icache) is a physical direct-mapped cache. Each cache block (line) contains the following:

- 32 bytes of Istream data
- 21-bit associated tag
- 6-bit address space number (ASN) field
- 1-bit address space match (ASM) field

- 8-bit (21066) or 16-bit (21066A) branch history (BHT) field

The 21066A has longword parity protection.

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

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

Note

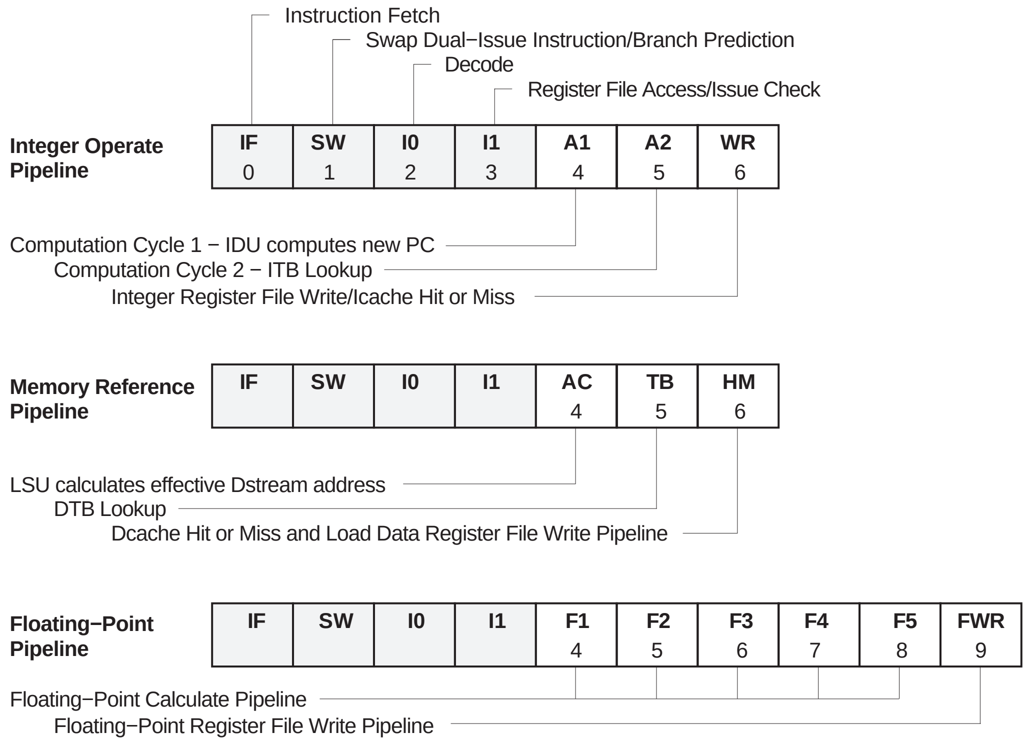
Both the Dcache and the Icache are longword parity protected in the 21066A. Refer to Section A.2 for details.

2.7 Instruction Pipeline Organization

The microprocessor has a 7-stage pipeline for integer operate and memory reference instructions and a 10-stage pipeline for floating-point operate instructions. The IDU maintains the state for all pipeline stages, to track outstanding register write operations, and to determine Icache hits and misses.

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

Figure 2–2 Instruction Pipelines


Instruction Fetch
Swap Dual−Issue Instruction/Branch Prediction
Decode
Register File Access/Issue Check
Integer Operate Pipeline
IF 0
SW 1
I0 2
I1 3
A1 4
A2 5
WR 6
Computation Cycle 1 − IDU computes new PC
Computation Cycle 2 − ITB Lookup
Integer Register File Write/Icache Hit or Miss
Memory Reference Pipeline
IF
SW
I0
I1
AC 4
TB 5
HM 6
LSU calculates effective Dstream address
DTB Lookup
Dcache Hit or Miss and Load Data Register File Write Pipeline
Floating−Point Pipeline
IF
SW
I0
I1
F1 4
F2 5
F3 6
F4 7
F5 8
FWR 9
Floating−Point Calculate Pipeline
Floating−Point Register File Write Pipeline

2.7.1 Instruction Pipeline Static and Dynamic Stages

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

- Instruction fetch
- Swap
- Decode
- Issue logic

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

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

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

2.7.2 Aborts

The causes for aborts are grouped into two classes:

- Exceptions (including interrupts)
- Nonexceptions

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

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

If an exception occurs, the processor aborts all instructions issued after the excepting instruction. Due to the nature of some error conditions, this can occur as late as the write cycle. The address of the excepting instruction is then latched in the exception address register (EXC_ADDR, Section 4.1.8). When the pipeline is fully drained, the processor begins to execute instructions at the address given by the PALcode dispatch. The pipeline is drained when:

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

2.7.3 Nonissue Conditions

Nonissue conditions occur for the following reasons:

- The pipeline freezes when a valid instruction or pair of instructions is prepared to issue but cannot because of a resource conflict. These types of nonissue cycles can be minimized through code scheduling.
- The existence of pipeline *bubbles* when the pipeline does not contain a valid instruction to issue.

Pipeline bubbles exist because of the abort conditions described in Section 2.7.2. In addition, a single pipeline bubble is produced when a branch-type instruction is predicted to be taken, including subroutine calls and returns. Pipeline bubbles are reduced directly by the hardware through *bubble squashing* but can also be effectively minimized with careful coding. Bubble squashing is the ability of any of the first four pipeline stages to advance when a bubble is detected in the pipeline stage immediately ahead of it while the pipeline is otherwise frozen.

2.8 Instruction Scheduling and Issuing Rules

Sections 2.8.1 through 2.8.5 describe the instruction scheduling and issuing rules.

2.8.1 Instruction Class Definition

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

Table 2–1 Producer–Consumer Classes

Class	Unit	Instructions
LD	Load and store	All loads
		HW_MFPR, RPCC, RS, RC, and STC producers only
		FETCH consumer only
ST	Load and store	All stores and HW_MTPR
IBR	Integer execution	Integer conditional branches

(continued on next page)

Table 2–1 (Cont.) Producer–Consumer Classes

Class	Unit	Instructions
FBR	Floating-point	Floating-point conditional branches
JSR	Integer execution	JMP, JSR, RET, and JSR_COROUTINE
		BSR and BR producer only
IADDLOG	Integer execution	ADDL, ADDL/V, ADDQ, and ADDQ/V
		LDA and LDAH
		SUBL, SUBL/V, SUBQ, SUBQ/V, S4ADDL, S4ADDQ, S8ADDL, S8ADDQ, S4SUBL, S4SUBQ, S8SUBL, and S8SUBQ
		AND, BIC, BIS, ORNOT, and XOR
SHIFTCM	Integer execution	CMOVEQ, CMOVNE, CMOVLT, CMOVLE, CMOVGT, CMOVGE, CMOVLBS, and CMOVLBC
		EXTQL, EXTLL, EXTWL, EXTBL, EXTQH, EXTLH, and EXTWH
		INSQL, INSLL, INSWL, INSBL, INSQH, INSLH, and INSWH
		MSKQL, MSKLL, MSKWL, MSKBL, MSKQH, MSKLH, and MSKWH
		SLL, SRL, and SRA
		ZAP and ZAPNOT
ICMP	Integer execution	CMPEQ, CMPLT, CMPLE, CMPULT, CMPULE, and CMPBGE
IMULL	Integer execution	MULL and MULL/V
IMULQ	Integer execution	MULQ, MULQ/V, and UMULH
FPOP	Floating-point	Floating-point operates except divide
FDIV	Floating-point	Floating-point divide

2.8.2 Producer–Consumer Latency

The microprocessor enforces scheduling rules regarding producer–consumer latencies. Table 2–2 shows these rules as a producer–consumer latency matrix. Each row and column in the matrix is a class of Alpha instructions.

Table 2–2 Producer–Consumer Latency Matrix

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

Notes for Table 2–2:

- The numbers in the matrix represent the cycles of latency. For example, one cycle of latency is indicated by a 1 in the matrix. One cycle of latency means that if instruction B uses the results of instruction A, then instruction B can be issued one cycle after instruction A is issued.
- LD indicates a Dcache hit is assumed for load instructions. The latency for Dcache miss depends on the system configuration.
- X indicates an impossible state or a state not normally encountered. For example, a floating-point branch would not follow an integer compare.
- For the ST consumer class, two latencies (for example, 2/0) are given with some producer classes. The first is the latency for base address of store, and the second is the latency for store data. Floating-point results cannot be used as the base address for load or store operations.
- Two latencies are given for IMUL followed by IMUL (for example, 21/19). The first is the latency with data dependency; in other words, the second IMUL uses the result from the first IMUL. The second is the multiply latency without data dependencies.

- Two latencies are given for FDIV followed by FDIV (for example, 34/30). The first is the latency with data dependency; in other words, the second FDIV uses the result from the first FDIV. The second is the divide latency without data dependencies.

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

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

The SRA instruction consumes the result (R3) produced by the ADDQ instruction. The matrix shows a latency of 1 associated with an IADD-SHIFT producer–consumer pair. In other words, if the ADDQ instruction is issued in cycle *n*, the SRA instruction can be issued in cycle *n*+1.

The SUBQ instruction consumes the result (R5) produced by the SRA instruction. The matrix shows a latency of 2 associated with a SHIFT-IADD producer–consumer pair. In other words, if the SRA instruction is issued in cycle *n*, the SUBQ instruction can be issued in cycle *n+2*. The IDU injects one no-operation (NOP) cycle in the pipeline for this case.

In the final case, the STQ instruction consumes the result (R7) produced by the SUBQ instruction. The matrix shows a latency of 0 associated with an IADD-ST producer–consumer pair when the result of the IADD instruction is the store data. In other words, the SUBQ and STQ instruction pair can be dual issued.

2.8.3 Producer–Producer Latency

Producer–producer latency (also known as write-after-write conflicts) is restricted only by the register write order. For most instructions, this is dictated by issue order; however, IMUL, FDIV, and LD instructions might require more time to complete than other instructions. Therefore, to preserve write ordering, IMUL, FDIV, and LD instructions must stall following instructions that write the same destination register.

In general, only cases involving an intervening producer–consumer conflict are of interest. They can commonly occur in a dual-issue situation when a register is reused. In these cases, producer–consumer latencies are equal to or greater than the required producer–producer latency as determined by write ordering and, therefore, dictate the overall latency. The following code is an example:

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

2.8.4 Instruction Issue Rules

The following conditions prevent instruction issue:

- No instruction can be issued until all of its source and destination registers are clean. In other words, all outstanding write operations to the destination register are guaranteed to complete in issue order, and either there are no outstanding write operations to the source registers or those write operations can be bypassed.
- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR, BSR, BR, or JSR (with destination other than R31) instruction can be issued after an MB instruction until the MB has been acknowledged on the external pin bus.
- No IMUL instructions can be issued if the integer multiplier is busy.
- No SHIFT, IADDLOG, ICMP, or ICMOV instruction can be issued exactly three cycles before an integer multiplication completes.
- No integer or floating-point conditional branch instruction can be issued in the cycle immediately following a JSR, JMP, RET, JSR_COROUTINE, or HW_REI instruction.
- No TRAPB instruction can be issued as the second instruction of a dual-issue pair.
- No LD instructions can be issued in the two cycles immediately following an STC instruction.
- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR, BSR, BR, or JSR (with destination other than R31) instruction can be issued when the LSU is busy due to a load miss or write buffer overflow. (For more information, see Section 2.4.3.)
- No FDIV instruction can be issued if the floating-point divider is busy.
- No floating-point operate instruction can be issued exactly five or six cycles before a floating-point divide completes.

2.8.5 Dual-Issue Table

Table 2–3 can be used to determine instruction pairs that can issue in a single cycle. Instructions are dispatched using two internal data paths or buses. In Table 2–3, the buses are referred to as IB0, IB1, and IBx.

In Table 2–3, any instruction identified with IB0 can be issued in the same cycle as any instruction identified with IB1. An instruction that is identified as IBx can be issued with either IB0 or IB1.

Dual issue is attempted if the input operands are available as defined by the producer–consumer latency matrix in Table 2–2 and the following requirements are met:

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

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

Table 2–3 Instruction Opcode Summary with Instruction Issue Bus

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

Key to abbreviations

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

Table 2–3 lists all Alpha opcodes from 00 (CALL_PAL) through 3F (BGT).

In the table, the column headings appearing over the instructions have a granularity of 8. The rows beneath the leftmost 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 leftmost 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 leftmost 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 leftmost 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 leftmost column.

The instruction format is listed under the instruction symbol.

See the *Alpha Architecture Reference Manual* for more information about instructions, their opcodes, and their definitions.

2.9 PALcode

Users and operating system developers require functions to be implemented consistently in a family of machines. When functions conform to a common interface, the code that uses those functions can be used on several different implementations without modification. PALcode allows many different physical implementations to coexist, each one adhering to the same programming interface specification.

2.9.1 Required PALcode Instructions

The PALcode instructions listed in Table 2–4 are described in the *Alpha Architecture Reference Manual*.

Table 2–4 Required PALcode Instructions

Mnemonic	Type	Operation
HALT	Privileged	Halt processor
IMB	Unprivileged	Istream memory barrier

2.9.2 Architecturally Reserved PALcode Instructions

The Alpha architecture provides five opcodes as implementation-specific privileged instructions. These instructions are defined independently for each hardware implementation of the Alpha architecture to give PALcode software routines access to specific hardware states and functions.

Table 2–5 lists the hardware-specific, privileged instructions that are executed in PALmode. They produce OPCDEC exceptions if executed while not in the PALcode environment. (See Section 3.5 for a definition of OPCDEC.) These instructions are mapped using the architecturally reserved opcodes PAL19, PAL1B, PAL1D, PAL1E, and PAL1F. They can be used only while executing chip-specific PALcode.

Table 2–5 Implementation-Specific PALcode Instructions

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

Programming Note

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

2.10 Physical Address Space

Internally, the chip uses a 34-bit address space divided into regions as shown in Table 2–6. Addresses in the cacheable memory region update the optional Bcache when referenced by the CPU. The graphics memory region supports simple graphics operations on data. In the PCI sparse memory region, addresses are shifted 5 bits to allow access size encodings. Addresses are not shifted in the PCI dense memory region. PCI peripherals can access only the cacheable and noncacheable memory regions because the IOC maps the 32-bit PCI address to the internal address that can access only those memory spaces. In other words, the PCI cannot access the IOC register, memory controller register, or graphics memory regions. (The translation mechanism is described in Chapter 6.)

Instructions cannot be executed from the upper three quadrants of memory; that is, when address bits <33:32> = 01, 10, or 11. Instructions can be executed from the cacheable and noncacheable regions of the lowest quadrant, in which case they will be cached in the Icache but not the Bcache.

Table 2–6 Physical Address Space

Address Region	Address Range (<33:0>)	Size
PCI dense memory	3 FFFF FFFF..3 0000 0000	4.0 GB
PCI sparse memory	2 FFFF FFFF..2 0000 0000	4.0 GB
PCI configuration	1 FFFF FFFF..1 E000 0000	0.5 GB
PCI I/O	1 DFFF FFFF..1 C000 0000	0.5 GB
PCI interrupt acknowledge and special cycle	1 BFFF FFFF..1 A000 0000	0.5 GB
IOC registers	1 9FFF FFFF..1 8000 0000	0.5 GB
Memory controller registers	1 7FFF FFFF..1 2000 0000	1.5 GB
Graphics memory*	1 1FFF FFFF..1 0000 0000	0.5 GB
Nonexistent memory (NXM)	0 FFFF FFFF..0 4000 0000	3.0 GB
Noncacheable memory	0 3FFF FFFF..0 2000 0000	0.5 GB
Cacheable memory†	0 1FFF FFFF..0 0000 0000	0.5 GB

*Only this region supports graphics functions on write operations.
†All other address regions are noncacheable.

2.11 Memory Controller

The onchip memory controller is the interface between the CPU and the system memory and optional Bcache. It has several memory-mapped control and status registers (CSRs) to program organization, timing, and size of DRAM, VRAM, and Bcache SRAM. It controls CPU requests and DMA requests (from the IOC) to and from memory and the Bcache. It also controls VRAM shift register load and memory refresh operations.

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

The memory controller can also perform the following graphics operations:

- Dumb frame buffer operation
- Transparent stipple operation
- Write-per-bit plane masking
- Byte write operations (with external gating)
- Full and split VRAM shift-register loads

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

2.12 I/O Controller

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

The IOC incorporates scatter-gather mapping logic to translate 32-bit addresses generated by PCI bus masters to the 34-bit CPU physical address space. The IOC implements an 8-entry translation lookaside buffer (TLB) for fast translations. Two programmable address windows, controlled by memory-mapped IOC registers, control access from PCI peripherals to system memory. (The IOC and its registers are described in Chapter 6.)

2.13 Lock Registers

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

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

- Execution of any STx_C instruction
- A DMA write operation to the locked range
- Execution of a CALL_PAL REI instruction
- The IOC is locked (Section 6.3.10)

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

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

3
Privileged Architecture Library Code

This chapter describes the microprocessor's privileged architecture library code (PALcode).

3.1 Overview

The Alpha architecture defines an innovative feature called PALcode that allows many different physical implementations to coexist, each one adhering to the same programming interface specification. PALcode has characteristics that make it appear to be a combination of microcode, ROM basic input/output operating system (BIOS), and system service routines, but it is not exactly analogous to any of them.

The following examples list the major reasons for using PALcode. In each case, PALcode routines provide the functions:

- Some necessary support functions are too complex to implement directly in a processor chip's hardware and cannot be handled by a normal operating system software routine. Such functions include routines to fill the translation buffer, acknowledge interrupts, and dispatch exceptions. In some architectures, these functions are handled by microcode, but the Alpha architecture is careful not to mandate the use of microcode for reasonable chip implementations.
- Some functions must run atomically and involve long sequences of instructions that may need complete access to all of the underlying computer hardware. An example is the sequence that returns from an exception or interrupt.
- Some instructions are necessary for backward compatibility or ease of programming; however, they 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 an Intel style interlocked memory access might be familiar to CISC programmers, but it 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.

The PALcode routines are simply programs that are invoked at specified times, and read in as instruction stream (Istream) code in the same way that all other Alpha code is read. After it is invoked, PALcode runs in a special environment.

3.2 PALmode Environment

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

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

One important aspect of PALcode is that it uses normal Alpha instructions for most of its operations; that is, the same instruction set used by nonprivileged Alpha programs. In addition, the following instructions are available only in PALmode:

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

These instructions will cause an OPCDEC exception if they are attempted while not in PALmode. The Alpha architecture allows some flexibility in the functions performed by these special PALmode instructions. In the microprocessor, the special PALmode 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).

See Section 3.4 for more information about these special PALmode instructions.

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

Caution

It is possible to cause unintended side effects by writing what appears to be perfectly acceptable PALcode. Digital recommends that most users do not attempt to change PALcode.

See Section 3.6 for more information about PALmode restrictions.

3.3 Invoking PALcode

PALcode is invoked at specific entry points, under certain well-defined conditions. PALcode can be thought of as a series of callable routines, with each routine indexed as an offset from a base address. The base address of the PALcode is programmable (stored in the PAL base address register—PAL_BASE, Section 4.1.16) and is normally set by the system reset code. See Section 3.5 for more information about PALcode entry points.

When an event that invokes PALcode occurs, the microprocessor first drains the pipeline. The current program counter (PC) is loaded into the exception address register (EXC_ADDR, Section 4.1.8), and the appropriate PALcode routine is dispatched. These operations are performed under direct control of the chip hardware, and place the machine 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 register. Bit 0 in the EXC_ADDR register indicates PALmode to the hardware. On return from a PALcode routine, EXC_ADDR register bit 0 is usually clear, and in that case the hardware loads the new PC, enables interrupts, enables memory mapping, and dispatches back to the user.

3.3.1 Hardware Dispatch to PALcode

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 the use of microcode in other architectures. There are four major categories of hardware-initiated PALcode:

- When the microprocessor is reset, it enters PALmode and executes the RESET PALcode. The system remains in PALmode until an HW_REI instruction is executed and EXC_ADDR register bit 0 is cleared. The system then continues execution in non-PALmode (native mode). While the initial RESET PALcode is executing, the remaining low-level system initialization is performed, including any modification to the PALcode base register.

- When the microprocessor detects a system hardware error, it invokes one of several PALcode routines, depending on the type of error. Errors handled in this way include machine checks, arithmetic exceptions, reserved or privileged instruction decode, and data fetch errors.
- When the microprocessor senses an interrupt, it dispatches the interrupt acknowledgment to a PALcode routine that gathers the necessary information, then handles the situation appropriately for the particular interrupt.
- When a data stream (Dstream) or Istream TB miss occurs, one of several PALcode routines is called to perform the TB fill. The memory-management algorithms or virtual-to-physical page mappings are flexible. In the simplest case, this can be an automatic one-to-one translation from virtual-to-physical address. In a typical operating system, these routines reference page tables and perform the translation and fill based on the page table contents.

These basic hardware-related functions are difficult to implement efficiently with typical operating system service routines.

3.3.2 CALL_PAL Instruction

PALcode can also be invoked with the CALL_PAL instruction. This special instruction dispatches to PALcode at a specific entry point by using the same steps as hardware-activated PALcode; that is, the pipeline is drained, the PC is saved, and the appropriate dispatch is made to an offset from the PALcode base. The difference is that the dispatch is controlled by the program through an instruction, rather than through a hardware event or error. Also, CALL_PAL instructions place PC+4 in the EXC_ADDR register.

The CALL_PAL instruction format includes the OPCODE (bits <31:26>) and the function field (bits <25:0>). The function field specifies the CALL_PAL routine to be invoked. The microprocessor hardware dispatches support only a subset of the possible CALL_PAL function values (Section 3.5). CALL_PAL routines can perform different functions for different operating systems running on the microprocessor. Unlike the basic hardware-generated PALcode, the CALL_PAL operations are largely optional and are based on the needs of the system implementation.

A subtle difference exists between hardware-dispatched and CALL_PAL-dispatched PALcode. Some form of the hardware-invoked PALcode functions are necessary in most computer systems. For example, when the microprocessor detects a serious system error, it dispatches to the machine check (MCHK) PALcode entry point. The exact PALcode that resides at this entry point can do what is reasonable, based on the needs of the system.

On the other hand, the CALL_PAL instruction dispatch is completely under control of the executing program. If the program never executes one of the instructions in the CALL_PAL list, that PALcode will never be run. Any PALcode that does run, executes in PALmode under the same restrictions as hardware-activated PALcode.

The microprocessor supports hardware dispatch for both privileged and nonprivileged CALL_PAL instructions. That is, some of the functions that are passed to the CALL_PAL instruction are considered special. A CALL_PAL instruction is privileged or nonprivileged depending on whether the user can call that particular instruction, not on the mode in which the instruction eventually runs. Every CALL_PAL instruction dispatches to PALcode that runs in PALmode.

The difference between privileged and nonprivileged CALL_PAL instructions is that privileged CALL_PAL instructions can be executed only in kernel mode; otherwise, they are vectored to 13E0 (OPCDEC, Table 3–5).

Privileged and nonprivileged CALL_PAL instructions are dispatched in exactly the same way and, when executed, enter PALmode, perform their function, and return to the user. The user mode is checked before execution. If a user attempts to execute a privileged CALL_PAL instruction in any mode other than kernel mode, an OPCDEC PALcode routine, rather than the CALL_PAL function, is run. An OPCDEC exception is also taken if a CALL_PAL function code that is not supported by the microprocessor hardware dispatch is attempted.

3.4 Reserved Opcode Instructions

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

- Special load and store instructions (HW_LD, HW_ST)
- Move-to and a move-from processor register instructions (HW_MTPR, HW_MFPR)
- Return from a PALmode exception or interrupt instruction (HW_REI)

These instructions produce an OPCDEC exception if executed while not in the PALmode environment. These instructions can be executed in kernel mode if the hardware enable (HWE) bit is set in the Icache control and status register (ICCSR, Section 4.1.7).

Register checking and bypassing logic is provided for PALcode instructions and for non-PALcode instructions, when using general-purpose registers.

Note

Explicit software timing is required for accessing the hardware-specific internal processor registers (IPRs) and the PALcode temporary (PAL_TEMP) registers. These constraints are described in Section 3.6 and Chapter 4.

3.4.1 HW_MFPR and HW_MTPR Instructions

PALcode uses the HW_MFPR and HW_MTPR instructions to move data from and to the IPRs. The IPR specified by the PAL, ABX, IBX, and index field is written or read with the data from the specified integer register.

Caution

Writing or reading IPRs can have side effects.

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

- LSU registers
- IDU registers
- PAL_TEMP registers

It is possible for an HW_MTPR instruction to write multiple registers in parallel if they both have the same index. (See Table 3–1.)

Figure 3–1 shows the HW_MFPR and HW_MTPR instruction format, and Table 3–1 describes its fields.

Figure 3–1 HW_MFPR and HW_MTPR Instructions Format

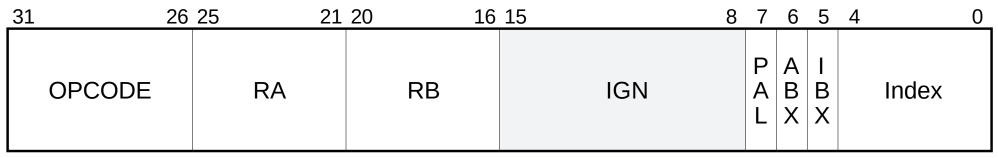


Table 3–1 HW_MFPR and HW_MTPR Format Description

Bits	Field	Description
31:26	OPCODE	The opcode is either 25 (HW_MFPR) or 29 (HW_MTPR).
25:21 20:16	RA RB	These fields contain the source (HW_MTPR) or destination (HW_MFPR) register number. RA and RB must be identical.
15:8	IGN	These bits are ignored.
7	PAL	When this bit is set, the instruction is referencing a PAL temporary (PAL_TEMP) register.*
6	ABX	When this bit is set, the instruction is referencing a register in the LSU.*
5	IBX	When this bit is set, the instruction is referencing a register in the IDU.*
4:0	Index	This bit specifies the index within the group.*

*See Table 3–2.

Table 3–2 indicates how the PAL, ABX, IBX, and INDEX fields are set to access the registers. Setting the PAL, ABX, and IBX fields to zero generates an NOP.

Table 3–2 Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	Index	Access	Comments
TB_TAG	—	—	1	0	WO	PALmode only
ITB_PTE	—	—	1	1	RW	PALmode only
ICCSR	—	—	1	2	RW	—
ITB_PTE_TEMP	—	—	1	3	RO	PALmode only

(continued on next page)

Table 3–2 (Cont.) Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	Index	Access	Comments
EXC_ADDR	—	—	1	4	RW	—
SL_RCV	—	—	1	5	RO	—
ITBZAP	—	—	1	6	WO	PALmode only
ITBASM	—	—	1	7	WO	PALmode only
ITBIS	—	—	1	8	WO	PALmode only
PS	—	—	1	9	RW	—
EXC_SUM	—	—	1	10	RW	—
PAL_BASE	—	—	1	11	RW	—
HIRR	—	—	1	12	RO	—
SIRR	—	—	1	13	RW	—
ASTRR	—	—	1	14	RW	—
HIER	—	—	1	16	RW	—
SIER	—	—	1	17	RW	—
ASTER	—	—	1	18	RW	—
SL_CLR	—	—	1	19	WO	—
SL_XMIT	—	—	1	22	WO	—
TB_CTL	—	1	—	0	WO	—
DTB_PTE	—	1	—	2	RW	—
DTB_PTE_TEMP	—	1	—	3	RO	—
MM_CSR	—	1	—	4	RO	—
VA	—	1	—	5	RO	—
DTBZAP	—	1	—	6	WO	—
DTBASM	—	1	—	7	WO	—
DTBIS	—	1	—	8	WO	—
DC_STAT	—	1	—	12	RO	—
ABOX_CTL	—	1	—	14	WO	—
ALT_MODE	—	1	—	15	WO	—
CC	—	1	—	16	RW	—
CC_CTL	—	1	—	17	WO	—

(continued on next page)

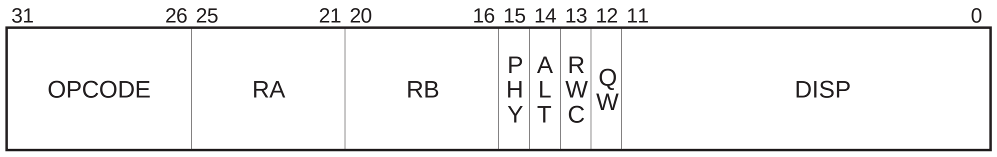
Table 3–2 (Cont.) Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	Index	Access	Comments
FLUSH_IC	—	1	—	21	WO	—
FLUSH_IC_ASM	—	1	—	23	WO	—
PAL_TEMP<31:0>	1	—	—	31..0	RW	—

3.4.2 HW_LD and HW_ST Instructions

PALcode uses the HW_LD and HW_ST instructions to access memory outside the realm of normal Alpha memory management. Figure 3–2 shows the HW_LD and HW_ST instructions format, and Table 3–3 describes its fields.

Figure 3–2 HW_LD and HW_ST Instructions Format



The effective address of these instructions is calculated as follows:

addr ⇐ (SEXT(DISP) + RB) AND NOT (QW | 11_2)

Table 3–3 HW_LD and HW_ST Format Description

Bits	Field	Description
31:26	OPCODE	The opcode is either 27 (HW_LD) or 31 (HW_ST).
25:21 20:16	RA RB	These fields specify register numbers, interpreted as is usual for load and store instructions.
15	PHY	Physical—When this bit is set, the effective address of the instruction is a physical address; otherwise, it is a virtual address.*

*Physical-mode load-lock (HW_LD/L) and store-conditional (HW_ST/C) variants of the HW_LD and HW_ST instructions can be created by setting both the PHY and ALT bits.

(continued on next page)

Table 3–3 (Cont.) HW_LD and HW_ST Format Description

Bits	Field	Description
14	ALT	Alternate mode—For virtual-mode HW_LD and HW_ST instructions, this bit selects the processor mode bits that are used for memory-management checks. When this bit is set, the mode bits in the ALT_MODE register are used; otherwise, the current mode bits of the processor status (PS) register are used.*
13	RWC	Read-with-write check—When set, this bit enables both read and write access checks on virtual HW_LD instructions.
12	QW	Quadword—When set, this bit specifies the data length is quadword; otherwise, the data length is longword.
11:0	DISP	This field holds a 12-bit, signed, byte displacement.

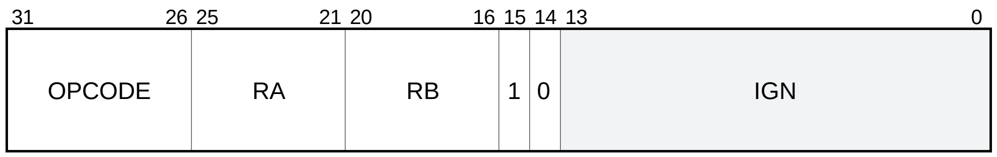
*Physical-mode load-lock (HW_LD/L) and store-conditional (HW_ST/C) variants of the HW_LD and HW_ST instructions can be created by setting both the PHY and ALT bits.

3.4.3 HW_REI Instruction

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

Figure 3–3 shows the HW_REI instruction format, and Table 3–4 describes its fields.

Figure 3–3 HW_REI Instruction Format



The next address and PALmode bit are calculated as follows:

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

Table 3–4 HW_REI Format Description

Bits	Field	Description
31:26	OPCODE	The opcode is 30.
25:21 20:16	RA RB	The register numbers contained in these fields should be 1F (R31); otherwise, a stall may occur.
15:14	10	These bits contain the branch prediction hint. The microprocessor pushes the contents of the EXC_ADDR register on the JSR prediction stack. Bit 15 must be set to pop the stack to avoid misalignment.
13:0	IGN	These bits are ignored.

3.5 PALcode Entry Points

Table 3–5 lists the PALcode entry points from the highest priority (RESET) to the lowest priority (FEN). The table defines only the entry point offset, bits <13:0>. The new PC bits <33:14> come from the PAL_BASE register. At power up, the PAL_BASE register value is zero.

Note

PALcode at PAL entry points of higher priority than DTB_MISS must unlock possible locks on the memory-management control and status register (MM_CSR, Section 4.2.9) and virtual address register (VA, Section 4.2.8).

Table 3–5 PALcode Entry Points

Entry Name	Time	Offset	Cause
RESET	Anytime	0000	Reset.
MCHK	Anytime	0020	Uncorrected hardware error.
ARITH	Anytime	0060	Arithmetic exception.
INTERRUPT	Anytime	00E0	Interrupt, including corrected hardware error.
Dstream Errors	Pipe stage 6	01E0, 08E0, 09E0, 11E0	See Table 3–6.

(continued on next page)

Table 3–5 (Cont.) PALcode Entry Points

Entry Name	Time	Offset	Cause
ITB_MISS	Pipe stage 5	03E0	ITB miss.
ITB_ACV	Pipe stage 5	07E0	Istream access violation.
CALL_PAL	Pipe stage 5	2000, 2040, 2080, 20C0 through 3FC0	128 locations based on instruction.
OPCDEC	Pipe stage 5	13E0	Reserved or privileged opcode.
FEN	Pipe stage 5	17E0	Floating-point (FP) operation attempted with: • FP instructions disabled by the ICCSR FPE bit • FP IEEE round to ± infinity • FP IEEE with data type field other than S, T, Q

PALcode functions are implemented through the CALL_PAL instruction. CALL_PAL instructions cause exceptions in the hardware. As with all exceptions, hardware loads the EXC_ADDR register with a possible return address.

CALL_PAL exceptions load the EXC_ADDR register with the address of the instruction following the CALL_PAL instruction, not the address of the CALL_PAL instruction. PALcode that supports the desired PALmode function need not increment the EXC_ADDR register before executing an HW_REI instruction to return to native mode. This feature requires special handling in the arithmetic trap and machine check PALcode flows. (See Section 4.1.8 for more information.)

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

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

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

CALL_PAL instructions that fall outside the ranges of 00000000..0000003F and 00000080..000000BF result in an OPCDEC exception. CALL_PAL instructions that fall within the range of 00000000..0000003F while the microprocessor is not executing in kernel mode also result in an OPCDEC exception.

The hardware recognizes four classes of Dstream memory management errors:

- Bad virtual address (VA) (incorrect sign extension)
- Data translation buffer (DTB) miss (native mode and PALmode)
- Alignment error
- Access violation (ACV), fault on read (FOR), fault on write (FOW)

The following errors get mapped into four PAL entry points:

- UNALIGN
- DTB_MISS PALmode
- DTB_MISS native mode
- D_FAULT

Table 3–6 lists the priority of these entry points as a group. (Also see Dstream errors, Table 3–5). A particular Dstream memory reference can generate errors that fall into more than one of the four error classes recognized by the hardware.

Table 3–6 Dstream Error PAL Entry Points

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

*ACV, FOR, FOW

1 = error
0 = no error
x = ignored

See Section 3.3 for more information about PALcode entry points.

3.6 PALmode Restrictions

Many of the PALmode restrictions involve waiting *n* cycles before using the results of a PAL instruction. The typical way to wait for *n* cycles is to insert *n* instructions between two time-sensitive instructions. Because the microprocessor can dual issue instructions, it is possible to write code that requires $2 \times n + 1$ instructions to wait *n* cycles. Due to the resource requirements of individual instructions and the microprocessor's hardware design, multiple copies of the same instruction cannot be dual issued. Following are a few examples of the PALmode restrictions:

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

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

  - Multiple read instructions
  - Multiple write instructions
  - A read instruction followed by write instruction

  These cycles can be guaranteed by either of the following:

  - Including seven instructions that do not use the register in transition
  - Proving, through the dual-issue rules or machine state, that at least three cycles of delay will occur

  Multiple copies of an HW_MFPR instruction (used as an NOP instruction) can be used to pad cycles after the original HW_MTPR instruction. Multiple copies of the same instruction will never dual-issue; therefore, a maximum of three instructions are necessary to ensure at least three cycles of delay.

  Example:

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

  The HW_REI instruction uses the instruction translation buffer (ITB) if the EXC_ADDR register contains a non-PALmode virtual program counter (VPC bit 0 = 0). The previous rule implies that at least three cycles of delay must be included after writing the ITB before executing an HW_REI instruction to exit PALmode.

Exceptions:

– HW_MFPR instructions reading a PAL_TEMP register can never occur exactly two cycles after an HW_MTPR instruction writing a PAL_TEMP register. The following code solves this problem:

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

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

– The EXC_ADDR register can be read by an HW_REI instruction two cycles after the HW_MTPR instruction. This is equivalent to one intervening cycle of delay. Use the following code to perform this function:

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

• An HW_MTPR operation to the data translation buffer invalidate single register (DTBIS, Section 4.2.7) cannot be sourced from a bypassed path. All data being moved to the DTBIS register must be sourced directly from the register file. One way to ensure this is to provide at least three cycles of delay before using the result of any integer operation (except MUL) as the source of an HW_MTPR DTBIS instruction.

Note

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

Use the following code to perform this function:

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

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

  JSR
  JMP
  RET
  JSR_COROUTINE
  BSR
  HW_REI
  All Bxx opcodes except BR
- Table 3–7 lists the number of cycles required after an HW_MTPR instruction and before an HW_REI instruction for the specified registers. These cycles can be ensured by inserting one HW_MFPR R31,0 instruction or other appropriate instructions for each cycle of delay required after the HW_MTPR instruction.

Table 3–7 HW_MTPR Restrictions

Register	Cycles Between HW_MTPR and HW_REI
DTBIS, ASM, ZAP	0
ITBIS, ASM, ZAP	2
HIER, SIER	3
HIRR, SIRR	3
ICCSR (FPE bit)	4
PS	5

- When loading the cycle counter register (CC, Section 4.2.13), bits <3:0> must be loaded with zero. Loading non-zero values in these bits can cause an inaccurate count.
- An HW_MTPR DTBIS instruction cannot be combined with an HW_MTPR ITBIS instruction. The hardware will not clear the ITB if both the instruction fetch and decode unit registers and the LSU registers are simultaneously selected. Two instructions are needed to clear each TB individually. Use the following code to perform this function:

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

- Three cycles of delay are required between the following instructions:

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

- The following operations are disabled in the cycle immediately following an HW_REI instruction (MxPR = MTPR or MFPR):
  - HW_MxPR ITB_TAG
  - HW_MxPR ITB_PTE
  - HW_MxPR ITB_PTE_TEMP

    This rule implies that it is never a good idea to allow exceptions while updating the ITB. The ITB register will not be written if:

    * An exception interrupts the flow of the ITB miss routine and attempts to execute an HW_REI instruction. The return address begins with an HW_MxPR instruction to an ITB register.
    * The return from the exception interrupt is predicted correctly to avoid any delay between the two instructions.

    The following code demonstrates this situation:

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

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

  - ITB_PTE_TEMP

  If the microprocessor is not in PALmode because the hardware enable (HWE) bit is set in the ICCSR, an HW_MTPR or HW_MFPR instruction to or from these registers is ignored.

- When writing the PAL_BASE register, exceptions must not occur. An exception occurring simultaneously with a write instruction to the PAL_BASE register can leave the register in a metastable state. All asynchronous exceptions except reset can be avoided under the following conditions:
  - PALmode—Blocks all interrupts
  - Machine checks disabled—Blocks I/O error exceptions (through the ABOX_CTL register or MB instruction isolation)
  - Not under trap shadow—Avoids arithmetic traps

    The trap shadow is defined as less than:
    * 3 cycles after a non-MUL integer operate that may overflow
    * 22 cycles after an MULL/V instruction
    * 24 cycles after an MULQ/V instruction
    * 6 cycles after a non-DIV floating-point operation that may cause a trap
    * 34 cycles after a DIVF or DIVS instruction that may cause a trap
    * 63 cycles after a DIVG or DIVT instruction that may cause a trap
- The sequence HW_MTPR PTE, HW_MTPR TAG is not allowed. At least two delay cycles must occur between HW_MTPR PTE and HW_MTPR TAG instructions. This applies to both the ITB and DTB.
- The MCHK exception service routine must check the EXC_SUM register for simultaneous arithmetic errors. Arithmetic traps will not trigger exceptions a second time after returning from exception service for the machine check.
- Three cycles of delay must be inserted between HW_MFPR DTB_PTE and HW_MFPR DTB_PTE_TEMP instructions. Use the following code to perform this function:

```
HW_MFPR Rx,DTB_PTE          ; Reads DTB_PTE into DTB_PTE_TEMP register
HW_MFPR R31,0               ; First cycle of delay
HW_MFPR R31,0               ; Second cycle of delay
HW_MFPR R31,0               ; Third cycle of delay
HW_MFPR Ry,DTB_PTE_TEMP     ; Read DTB_PTE_TEMP into register file Ry
```

- Three cycles of delay must be inserted between HW_MFPR ITB_PTE and HW_MFPR ITB_PTE_TEMP instructions. Use the following code to perform this function:

```
HW_MFPR Rx,ITB_PTE          ; Reads ITB_PTE into ITB_PTE_TEMP register
HW_MFPR R31,0               ; First cycle of delay
HW_MFPR R31,0               ; Second cycle of delay
HW_MFPR R31,0               ; Third cycle of delay
HW_MFPR Ry,ITB_PTE_TEMP     ; Read ITB_PTE_TEMP into register file Ry
```

- The content of the destination register for HW_MFPR Rx,DTB_PTE and HW_MFPR Rx,ITB_PTE instructions is UNPREDICTABLE.
- Two HW_MFPR DTB_PTE instructions cannot be issued in consecutive cycles. This implies that more than one instruction might be necessary between the HW_MFPR instructions if dual issue is possible. Similar restrictions apply to the ITB_PTE register.
- Reading the EXC_SUM register requires special timing. (See Section 4.1.9 for specific information.)
- Dstream memory management (DMM) errors occurring one cycle before HW_MxPR instructions to the ITB_PTE will not stop the TB pointer from incrementing to the next TB entry even though the HW_MxPR instruction will be aborted by the DMM error. This behavior affects performance but not functionality.
- PALcode that writes multiple ITB entries must write the entry that maps the address contained in the EXC_ADDR register last.
- HW_ST instructions cannot be followed for two cycles by any load instruction that may miss in the Dcache.
- Updates to the ICCSR ASN field require at least 10 cycles of delay before entering native mode that can reference the ASN during Icache access. If the ICCSR HWE bit is used to update the ASN field in kernel mode, it is sufficient during this time to make all Istream references to pages with the ASM bit set to avoid use of the ASN field.
- HW_MTPR instructions that update the TB_CTL register cannot follow by one cycle an HW_MTPR instruction that updates the DTB_PTE or ITB_PTE register.

- Table 3–8 shows the delays required for HW_MTPR instructions that update the ICCSR (ASN field), FLUSH_IC, and FLUSH_IC_ASM registers.

Table 3–8 HW_MTPR Cycle Delay

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

  The delay ensures that the update occurs before the first instruction fetch in native mode, because the pipeline can contain instructions that were fetched before the update (which would remain valid during a pipeline stall). At least one instruction must be issued during each cycle of the delay to ensure that the pipeline is cleared of all instructions fetched prior to the update.

  If the update is performed in kernel mode using the ICCSR HWE bit, it is sufficient during this time to make all Istream references to pages with the ASM bit set to avoid use of the ASN field.

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

  The mechanism that stops instruction flow during machine check exceptions causes the machine check exception to appear as a Dstream fault on the following instruction in the hardware pipeline. If the following instruction is an HW_MxPR instruction, a Dstream fault will not abort execution in all cases. The EXC_ADDR register will be loaded with the address of the HW_MxPR instruction as if it were aborted. An HW_REI instruction to this restart address will incorrectly re-execute this instruction.

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

3.6.1 Translation Buffer Miss Flows

Sections 3.6.1.1 and 3.6.1.2 describe hardware-specific details that are helpful when writing ITB and DTB fill routines. The flows highlight the trade-offs and restrictions between PALcode and hardware. The PALcode source that is released with the chip should be referenced before any new flows are written. The following descriptions assume the reader has a working knowledge of the Alpha memory-management architecture.

3.6.1.1 Instruction Translation Buffer Miss

When the IDU encounters an ITB miss, it performs the following sequence:

1. Latches the VPC of the target Istream reference in the EXC_ADDR register.
2. Flushes the pipeline of any instructions that follow the instruction that caused the ITB miss.
3. Waits for the completion of any other instructions that may be in progress.
4. Enters PALmode.
5. Jumps to the ITB miss PALcode entry point.

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

1. Create some scratch area in the integer register file (IRF) by writing the content of a few integer registers to the PAL_TEMP register file.
2. Read the target virtual address from the EXC_ADDR register.
3. Fetch the page table entry (PTE) (this can take multiple read operations) using a physical-mode HW_LD instruction. If this PTE valid bit is clear, either report translation not valid (TNV) or ACV, as appropriate.
4. Check the PTE valid bit because translation buffers cannot contain invalid PTEs. Also check the ITB PTE RAM to ensure it does not hold the fault on execute (FOE) bit. If the PTE valid bit is set and the FOE bit is clear, PALcode can fill an ITB entry.
5. Write the original virtual address to the TB_TAG register, using an HW_MTPR instruction. This writes the tag into a temporary register and not into the ITB TAG field.
6. Write the PTE to the TB_CTL register to select either the large-page or small-page TB regions. Wait at least one cycle before executing the next step.

7. Write the PTE to the ITB_PTE register, using an HW_MTPR instruction. This HW_MTPR instruction causes the ITB TAG and PTE fields to be written.

Note

The HW_MTPR instruction to the ITB_PTE register can be issued without delay after the HW_MTPR instruction to the TB_TAG register is issued.

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

3.6.1.2 Data Translation Buffer Miss

When the LSU encounters a DTB miss, it performs the following sequence:

1. Latches the referenced virtual address in the virtual address register (VA, Section 4.2.8) and latches other information about the reference in the memory management control and status register (MM_CSR, Section 4.2.9).
2. Locks the VA register and MM_CSR against further modifications and latches the PC of the instruction that generated the reference in the EXC_ADDR register.
3. Drains the machine (Section 3.6.1.1).
4. Jumps to the DTB miss PALcode entry point.

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

1. Create some scratch area in the IRF by writing the contents of a few integer registers to the PAL_TEMP register file.
2. Read the requested virtual address from the VA register. Reading this register unlocks the VA register and the MM_CSR. The MM_CSR is updated only when Dstream memory-management errors occur. It retains information about the instruction that generated the DTB miss for later use.

3. Fetch the PTE. This operation can require multiple read operations. If the valid bit of the PTE is clear, either a TNV or an ACV must be reported, unless a FETCH or FETCH_M instruction caused the DTB miss. This can be checked in the MM_CSR OPCODE field. If the value in this field is 18_{16}, a FETCH or FETCH_M instruction caused this DTB miss, and the subsequent TNV or ACV should not be reported. Therefore, PALcode should do the following:

   a. Read the value in the EXC_ADDR register.

   b. Increment the value by four.

   c. Write the value back to the EXC_ADDR register.

   d. Execute an HW_REI instruction.

4. Write the register that holds the contents of the PTE to the DTB_CTL register. This effectively selects one of the four possible granularity hint sizes.

5. Write the original virtual address to the TB_TAG register. This writes the tag into a temporary register and not into the DTB TAG field.

6. Write the PTE to the DTB_PTE register. This HW_MTPR instruction causes the DTB TAG and PTE fields to be written.

   Note

   It is not necessary to delay issuing the HW_MTPR instruction to the DTB_PTE register after the HW_MTPR instruction to the TB_TAG register is issued.

7. Restore the contents of any modified integer registers.

8. Restart the Istream with an HW_REI instruction.

4 Internal Processor Registers

This chapter describes the following internal processor registers (IPRs):

- IDU registers
- LSU registers
- PAL_TEMP registers

Note

The abbreviations in the Type column of the register field description tables indicate field access behavior. The abbreviations are defined in the Conventions section of the Preface.

The memory controller and I/O controller (IOC) control and status registers (CSRs) are described in Chapters 5 and 6.

4.1 IDU Registers

Sections 4.1.1 through 4.1.16 describe the IDU registers.

4.1.1 Translation Buffer Tag Register

The write-only translation buffer tag (TB_TAG) register holds the tag for the next translation buffer update operation in the instruction translation buffer (ITB) or the data translation buffer (DTB).

The tag is written to a temporary register and not transferred to the ITB or DTB until the ITB page table entry register (ITB_PTE, Section 4.1.2) or the DTB page table entry register (DTB_PTE, Section 4.2.3) is written. The entry to be written is chosen at the time of the ITB_PTE or DTB_PTE write operation. The DTB replacement algorithm (a not-last-used algorithm or a round-robin algorithm) is specified by the DTB round-robin enable (DTBRR) bit in the LSU control register (ABOX_CTL, Section 4.2.1).

The page format is determined by the granularity hints (GH) field in the translation buffer control register (TB_CTL, Section 4.2.2).

Figure 4–1 shows the TB_TAG register formats, and Tables 4–1 and 4–2 describe its fields.

Note

The TB_TAG register is written only while in PALmode, regardless of the state of the hardware enable (HWE) bit in the instruction cache control and status register (ICCSR, Section 4.1.7).

Figure 4–1 TB_TAG Register Formats

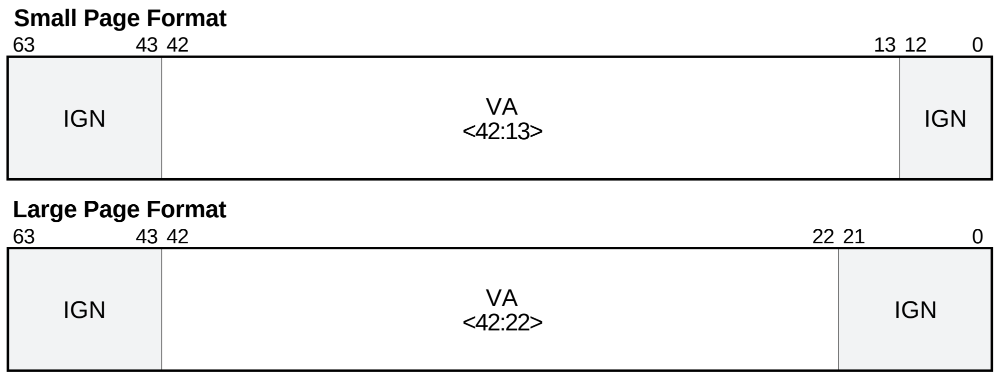


Table 4–1 TB_TAG Register Small-Page Format Field Description

Bits	Field	Type	Description
63:43	IGN	WO	Ignored when written.
42:13	VA<42:13>	WO	Virtual address bits <42:13>.
12:0	IGN	WO	Ignored when written.

Table 4–2 TB_TAG Register Large-Page Format Field Description

Bits	Field	Type	Description
63:43	IGN	WO	Ignored when written.
42:22	VA<42:22>	WO	Virtual address bits <42:22>.
21:0	IGN	WO	Ignored when written.

4.1.2 Instruction Translation Buffer Page Table Entry Register

The read and write instruction translation buffer page table entry (ITB_PTE) register represents 12 page table entries split into two distinct arrays. The first eight page table entries provide small-page (8 KB) translations and the remaining four provide large-page (4 MB) translations. The entry to be written is chosen by a not-last-used algorithm (implemented in hardware independently for each array) and the granularity hints (GH) field in the translation buffer control register (TB_CTL, Section 4.2.2).

The ITB_PTE register is written using the format described in the *Alpha Architecture Reference Manual*, but some fields are ignored.

The ITB tag array is updated from the TB_TAG register (Section 4.1.1) when the ITB_PTE register is written. Two instructions are required to read the ITB_PTE register:

1. The first instruction sends the PTE data to the instruction translation buffer page table entry temporary register (ITB_PTE_TEMP, Section 4.1.3).
2. The second instruction reads the ITB_PTE_TEMP register and returns the PTE to the register file.

To access the complete set of ITB_PTE register entries, the TB entry pointer that corresponds to the large- or small-page format (selected by the TB_CTL register) is incremented when the ITB_PTE register is read or written.

Figure 4–2 shows the ITB_PTE register formats, and Tables 4–3 and 4–4 describe its fields.

Note

The ITB_PTE register is written only while in PALmode, regardless of the state of the hardware enable (HWE) bit in the instruction cache control and status register (ICCSR, Section 4.1.7).

Figure 4–2 ITB_PTE Register Formats

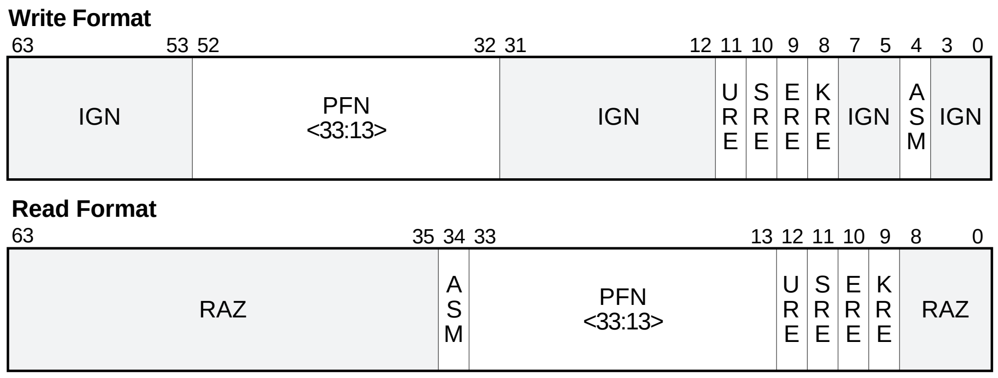


Table 4–3 ITB_PTE Register Write-Format Field Description

Bits	Field	Type	Description[*]
63:53	IGN	RW	Ignored when written.
52:32	PFN<33:13>	RW	Page frame number—This field always points to a page boundary.
31:12	IGN	RW	Ignored when written.
11	URE	RW	User read-enable—When set, this bit enables user mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in user mode.
10	SRE	RW	Supervisor read-enable—When set, this bit enables supervisor mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in supervisor mode.
9	ERE	RW	Executive read-enable—When set, this bit enables executive mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in executive mode.
8	KRE	RW	Kernel read-enable—When set, this bit enables kernel mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in kernel mode.

[*] For more information about the fields, see the *Alpha Architecture Reference Manual*.

(continued on next page)

Table 4–3 (Cont.) ITB_PTE Register Write-Format Field Description

Bits	Field	Type	Description[*]
7:5	IGN	RW	Ignored when written.
4	ASM	RW	Address space match—When this bit is set, this PTE matches all address space numbers (ASNs). For a given virtual address, this bit must be set consistently in all processes; otherwise, the address mapping is UNPREDICTABLE.
3:0	IGN	RW	Ignored when written.

[*]For more information about the fields, see the *Alpha Architecture Reference Manual*.

Table 4–4 ITB_PTE Register Read-Format Field Description

Bits	Field	Type	Description[*]
63:35	RAZ	RW	Read as zero.
34	ASM	RW	See the corresponding write-format field description (Table 4–3).
33:13	PFN<33:13>	RW	See the corresponding write-format field description (Table 4–3).
12	URE	RW	See the corresponding write-format field description (Table 4–3).
11	SRE	RW	See the corresponding write-format field description (Table 4–3).
10	ERE	RW	See the corresponding write-format field description (Table 4–3).
9	KRE	RW	See the corresponding write-format field description (Table 4–3).
8:0	RAZ	RW	Read as zero.

[*]For more information about the fields, see the *Alpha Architecture Reference Manual*.

4.1.3 Instruction Translation Buffer Page Table Entry Temporary Register

The instruction translation buffer page table entry temporary (ITP_PTE_TEMP) register is a read-only holding register for ITB_PTE read data. Two instructions are required to read the ITB_PTE register (Section 4.1.2) and to return data to the integer register file:

1. The first instruction reads the ITB_PTE register data to the ITB_PTE_TEMP register.
2. The second instruction reads the ITB_PTE_TEMP register data to the integer register file.

The ITB_PTE_TEMP register is updated on all ITB read accesses and write accesses. A read of the ITB_PTE register to the ITB_PTE_TEMP register should be followed closely by a read of the ITB_PTE_TEMP register to the register file.

Note

The ITB_PTE_TEMP register is read only while in PALmode, regardless of the state of the hardware enable (HWE) bit in the instruction cache control and status register (ICCSR, Section 4.1.7).

The ITB_PTE_TEMP register format and fields are the same as the ITB_PTE register read format shown in Figure 4–2 and described in Table 4–4.

4.1.4 Instruction Translation Buffer ZAP Register

A write operation to the instruction translation buffer ZAP (ITBZAP) register invalidates all 12 ITB entries. It also resets both not-last-used (NLU) pointers to their initial state. The ITBZAP register is written only in PALmode.

4.1.5 Instruction Translation Buffer ASM Register

A write operation to the instruction translation buffer ASM (ITBASM) register invalidates all ITB entries in which the address space match (ASM) bit equals zero. The ITBASM register is written only in PALmode.

4.1.6 Instruction Translation Buffer Invalidate Single Register

A write operation to the instruction translation buffer invalidate single (ITBIS) register invalidates the ITB entry that maps the virtual address held in the integer register. The integer register is identified by the Rb field of the HW_MTPR instruction used to perform the write operation.

4.1.7 Instruction Cache Control and Status Register

The instruction cache control and status register (ICCSR) contains various IDU hardware enable bits and fields.

The only architecturally defined bit in this register is the floating-point enable (FPE) bit, which enables floating-point instructions. Hardware clears all of the bits in this register at reset, except for the ASN field and performance counter interrupt enable bits PC0 and PC1.

The HWE bit allows the special PALcode instructions to execute in kernel mode. This bit is only for diagnostic routines or operating system alternative PALcode routines. It does not allow access to the ITB registers while not in PALmode.

Figure 4–3 shows the ICCSR formats, and Tables 4–5 and 4–6 describe its fields.

Figure 4–3 ICCSR Formats

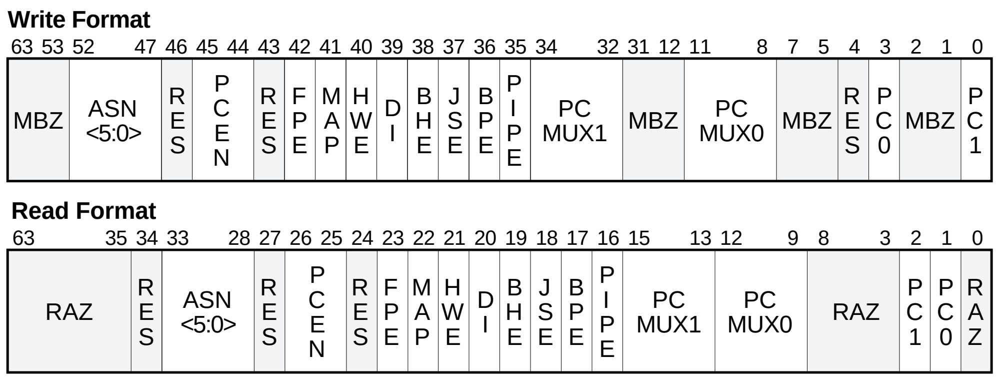


Table 4–5 ICCSR Write-Format Field Description

Bits	Field	Type	Description
63:53	MBZ	RW	Must be zero.
52:47	ASN<5:0>	RW	Address space number—This field is used in conjunction with the Icache to further qualify cache entries and avoid some cache flushes. The ASN is written to the Icache during fill operations and compared with the Istream data on fetch operations. Mismatches invalidate the fetch without affecting the Icache.
46	RES	RW	Reserved by Digital and should not be used.
45:44	PCEN	RW	Performance counter enable—When either bit is set, both performance counters increment. When both bits are clear, both performance counters are disabled.
43	RES	RW	Reserved by Digital and should not be used.
42	FPE	RW	Floating-point enable—When this bit is set, floating-point instructions can be issued. When this bit is clear, floating-point instructions cause FEN exceptions (Section 3.5).
41	MAP	RW	Superpage map enable—When set, this bit enables superpage Istream memory mapping of VPC<33:13> directly to physical PC<33:13>, essentially bypassing the ITB when VPC<42:41> = 2. Superpage mapping is allowed only in kernel mode. The Icache ASM bit is always set. When clear, this bit disables superpage mapping.
40	HWE	RW	Hardware enable—When this bit is set, the five PALRES instructions (Section 3.4) can be issued in kernel mode. When this bit is clear, attempts to execute PALRES instructions while not in PALmode result in OPCDEC exceptions (Section 3.5).
39	DI	RW	Dual issue enable—When set, this bit enables instruction dual issue. When this bit is clear, only one instruction can be issued for each CPU cycle.
38	BHE	RW	Branch history enable—This bit is used in conjunction with the BPE bit (<36>) (Table 4–7).

(continued on next page)

Table 4–5 (Cont.) ICCSR Write-Format Field Description

Bits	Field	Type	Description
37	JSE	RW	JSR stack enable—When set, this bit enables the JSR stack to push return addresses. When this bit is clear, the JSR stack is disabled. See the *Alpha Architecture Reference Manual* for more information about the JSR stack.
36	BPE	RW	Branch predict enable—This bit is used in conjunction with the BHE bit (<38>) (Table 4–7).
35	PIPE	RW	When this bit is set, the pipeline proceeds normally. When this bit is clear, all hardware-interlocked instructions drain the machine and wait for the write buffer to empty before issuing the next instruction. Instructions that do not cause the pipe to drain include HW_MTPR, HW_REI, conditional branches, and instructions that specify R31 as a destination register.
34:32	PCMUX1	RW	Performance counter 1 multiplexer—See Section 4.1.14 for more information.
31:12	MBZ	RW	Must be zero.
11:8	PCMUX0	RW	Performance counter 0 multiplexer—See Section 4.1.14 for more information.
7:5	MBZ	RW	Must be zero.
4	RES	RW	Reserved by Digital and should not be used.
3	PC0	RW	Performance counter 0 interrupt request enable—The PC0 interrupt request is enabled after 2^{12} events are counted when this bit is set, and after 2^{16} events are counted when this bit is clear.
2:1	MBZ	RW	Must be zero.
0	PC1	RW	Performance counter 1 interrupt request enable—The PC1 interrupt request is enabled after 2^{8} events are counted when this bit is set, and after 2^{12} events are counted when this bit is clear.

Table 4–6 ICCSR Read-Format Field Description

Bits	Field	Type	Description
63:35	RAZ	RW	Read as zero.
34	RES	RW	Reserved by Digital and should not be used.
33:28	ASN<5:0>	RW	See the corresponding write-format field description (Table 4–5).
27	RES	RW	Reserved by Digital and should not be used.
26:25	PCEN	RW	See the corresponding write-format field description (Table 4–5).
24	RES	RW	Reserved by Digital and should not be used.
23	FPE	RW	See the corresponding write-format field description (Table 4–5).
22	MAP	RW	See the corresponding write-format field description (Table 4–5).
21	HWE	RW	See the corresponding write-format field description (Table 4–5).
20	DI	RW	See the corresponding write-format field description (Table 4–5).
19	BHE	RW	See the corresponding write-format field description (Table 4–5).
18	JSE	RW	See the corresponding write-format field description (Table 4–5).
17	BPE	RW	See the corresponding write-format field description (Table 4–5).
16	PIPE	RW	See the corresponding write-format field description (Table 4–5).
15:13	PCMUX1	RW	See the corresponding write-format field description (Table 4–5).
12:9	PCMUX0	RW	See the corresponding write-format field description (Table 4–5).
8:3	RAZ	RW	Read as zero.
2	PC1	RW	See the corresponding write-format field description (Table 4–5).
1	PC0	RW	See the corresponding write-format field description (Table 4–5).

(continued on next page)

Table 4–6 (Cont.) ICCSR Read-Format Field Description

Bits	Field	Type	Description
0	RAZ	RW	Read as zero.

Table 4–7 defines the BPE and BHE bit branch selection.

Table 4–7 BPE and BHE Branch Prediction Selection

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

4.1.8 Exception Address Register

The read and write exception address (EXC_ADDR) register is used to restart the system after exceptions or interrupts. Software can use the HW_MTPR instruction to read and write this register, and it can also be written directly by the hardware.

When an event that invokes PALcode occurs, the current program counter (PC) value is loaded into the EXC_ADDR register, and the appropriate PALcode routine is dispatched. 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 register. On return from a PALcode routine, EXC_ADDR register bit 0 (which indicates PALmode to the hardware) is usually clear and, in that case, the hardware loads the new PC, enables interrupts, enables memory mapping, and dispatches back to the user in native (non-PAL) mode.

Caution

The HW_MTPR instruction can be used to update the EXC_ADDR register while in native mode only when the PAL bit value is zero. The combination of native mode and a PAL bit value of one causes UNDEFINED behavior.

CALL_PAL exceptions load the EXC_ADDR register with the PC of the instruction following the CALL_PAL instruction. This allows CALL_PAL service routines to return without needing to increment the value in the EXC_ADDR register. This feature requires careful treatment in PALcode. Arithmetic traps and machine check exceptions can prompt CALL_PAL exceptions resulting in an incorrect value being saved in the EXC_ADDR register.

EXC_ADDR register bit 1 takes on special meaning only in the cases of an arithmetic trap (ARITH) or a machine check (MCHK) exception. PALcode servicing these two exceptions must interpret bit 1 as follows:

- A zero in bit 1 indicates that the PC in <63:2> is too large by a value of 4 bytes; subtract 4 before executing an HW_REI instruction from this address.
- A one in bit 1 indicates that the PC in <63:2> is correct; clear bit 1.

All other PALcode entry points except RESET can expect EXC_ADDR register bit 1 to be zero. (See Section 3.5 for more information about PALcode entry points.)

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

```
HW_MFPR Rx, EXC_ADDR    ;Read EXC_ADDR into GPR
SUBQ    Rx, 2,Rx        ;Subtract 2 causing borrow if bit [1]=0
BIC     Rx, 2,Rx        ;Clear bit [1]
HW_MTPR Rx, EXC_ADDR    ;Write-back to EXC_ADDR
```

Figure 4–4 shows the EXC_ADDR register format, and Table 4–8 describes its fields.

Figure 4–4 EXC_ADDR Format

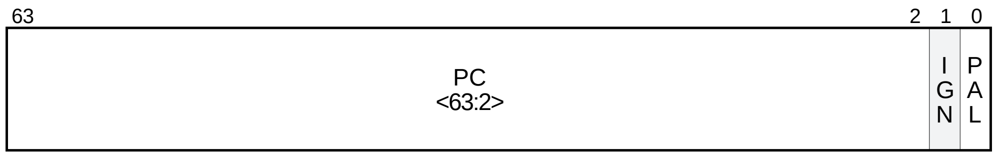


Table 4–8 EXC_ADDR Field Description

Bits	Field	Type	Description
63:2	PC<63:2>	RW	This field holds the program counter value for the instruction that did not complete its execution when an exception or interrupt occurred.
1	IGN	RW	This bit is normally ignored. It has special meaning only for arithmetic trap and machine check exceptions.
0	PAL	RW	When set, this bit indicates PALmode; when clear, it indicates native mode.

4.1.9 Exception Summary Register

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

The register containing the result of an arithmetic operation is recorded in the *exception register write mask*. The exception register write mask is a 64-bit parameter in which each bit corresponds to a floating-point or integer general-purpose register (F31..F0 and R31..R0). The exception register write mask is visible only through the 1-bit window (MSK) in the EXC_SUM register. Each read of the EXC_SUM register reads one exception register write mask bit in the sequence F31..F0 then R31..R0. The read also clears the exception register write mask bit. The EXC_SUM register must be read 64 times to extract and clear the complete exception register write mask.

Any write operation to the EXC_SUM register clears bits <8:2> but does not affect the MSK bit.

An exception register write mask bit is cleared three cycles after it is read. Code that reads the EXC_SUM register must allow at least three cycles between reads. This allows the exception register write mask clear and shift operations to complete and ensures that successive bits are read.

Figure 4–5 shows the EXC_SUM register format, and Table 4–9 describes its fields.

Figure 4–5 EXC_SUM Register Format

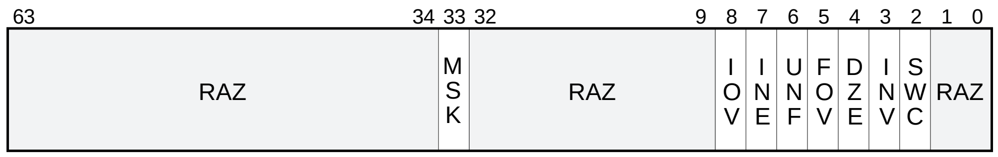


Table 4–9 EXC_SUM Register Field Description

Bits	Field	Type	Description
63:34	RAZ	RW	Read as zero.
33	MSK	RC	Exception register write mask window.
32:9	RAZ	RW	Read as zero.
8	IOV	WA	When set, this bit indicates an overflow or integer arithmetic overflow on an FPU convert to integer operation.
7	INE	WA	When set, this bit indicates a floating-point inexact error.
6	UNF	WA	When set, this bit indicates a floating-point underflow.
5	FOV	WA	When set, this bit indicates a floating-point overflow.
4	DZE	WA	When set, this bit indicates a divide-by-zero operation.
3	INV	WA	When set, this bit indicates an invalid operation.
2	SWC	WA	Software completion—When set, this bit indicates that software completion is possible. It is set after a floating-point instruction containing the /S modifier completes with an arithmetic trap and all previous floating-point instructions that trapped since the last HW_MTPR instruction to the EXC_SUM register also contained the /S modifier. This bit is cleared when 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 on any HW_MTPR instruction to the EXC_SUM register.
1:0	RAZ	RW	Read as zero.

4.1.10 Clear Serial Line Interrupt Register

The write-only clear serial line interrupt (SL_CLR) register clears serial line interrupt requests and performance counter interrupt requests.

The SLC, PC1, and PC0 bits are written with a zero to clear the corresponding interrupt request. These interrupts are enabled in the hardware interrupt enable register (HIER, Section 4.1.13.4).

Figure 4–6 shows the SL_CLR register format, and Table 4–10 describes its fields.

Figure 4–6 SL_CLR Register Format

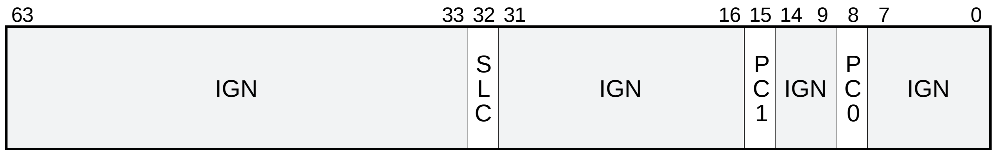


Table 4–10 SL_CLR Register Field Description

Bits	Field	Type	Description
63:33	IGN	WO	Ignored when written.
32	SLC	WO	When written with a zero, this bit clears the serial line interrupt request.
31:16	IGN	WO	Ignored when written.
15	PC1	WO	When written with a zero, this bit clears the performance counter 1 interrupt request.
14:9	IGN	WO	Ignored when written.
8	PC0	WO	When written with a zero, this bit clears the performance counter 0 interrupt request.
7:0	IGN	WO	Ignored when written.

4.1.11 Serial Line Receive Register

The serial line receive (SL_RCV) register contains a single read-only bit (RCV). This bit and the TMT bit in the serial line transmit register (SL_XMIT, Section 4.1.12) are used with the interrupt control registers and the **sromd** and **sromclk** pins to provide an onchip serial line function. The RCV bit is functionally connected to the **sromd** pin after the external SROM loads the Icache. Using a software timing loop, the RCV bit can be read to receive external data, one bit at a time. (See Chapter 9 for more information.)

When a transition is detected on the receive line, the serial line request (SLR) bit in the hardware interrupt request register (HIRR, Section 4.1.13.1) is set to request a serial line interrupt. The serial line interrupt is cleared by the SLC bit in the serial line clear register (SL_CLR, Section 4.1.10) and can be disabled by clearing the serial line enable (SLE) bit in the hardware interrupt enable register (HIER, Section 4.1.13.4).

Figure 4–7 shows the serial line receive register format, and Table 4–11 describes its fields.

Figure 4–7 SL_RCV Register Format

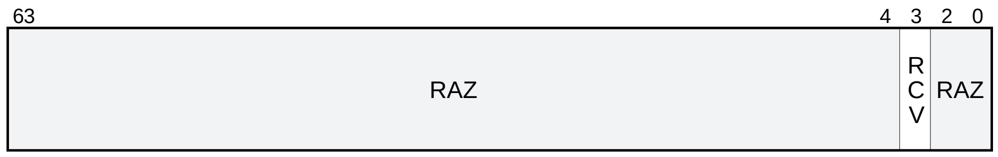


Table 4–11 SL_RCV Register Field Description

Bits	Field	Type	Description
63:4	RAZ	RO	Read as zero.
3	RCV	RO	Receive—This bit is functionally connected to the **sromd** pin after the Icache is initially loaded from the external SROM.
2:0	RAZ	RO	Read as zero.

4.1.12 Serial Line Transmit Register

The serial line transmit (SL_XMIT) register contains a single write-only bit (TMT). This bit and the RCV bit in the serial line receive register (SL_RCV, Section 4.1.11) are used with the interrupt control registers and the **sromd** and **sromclk** pins to provide an onchip serial line function. The TMT bit is functionally connected to the **sromclk** pin after the external SROM loads the Icache. Using a software timing loop, the TMT bit can be written to transmit data externally, one bit at a time. (See Chapter 9 for more information.)

Figure 4–8 shows the SL_XMIT register format, and Table 4–12 describes its fields.

Figure 4–8 SL_XMIT Register Format

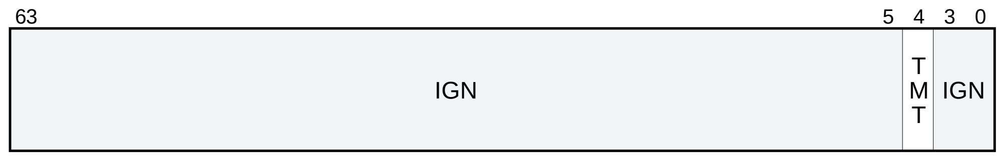


Table 4–12 SL_XMIT Register Field Description

Bits	Field	Type	Description
63:5	IGN	WO	Ignored when written.
4	TMT	WO	Transmit—This bit is functionally connected to the **sromclk** pin after the Icache is initially loaded from the external SROM.
3:0	IGN	WO	Ignored when written.

4.1.13 Interrupt Request and Enable Registers

The interrupt request and enable registers comprise two sets of three registers each. The read formats for the three interrupt request registers are identical, as are the read formats for the three interrupt enable registers. Each read/write register has a unique write format.

Table 4–13 lists the interrupt request and enable register and the sections and figures that describe them.

Table 4–13 Interrupt Request and Enable Registers

Register	Mnemonic	Access	Description Section	Read-Format Figure	Write-Format Figure
Hardware interrupt request register	HIRR	RO	4.1.13.1	4–9	None
Software interrupt request register	SIRR	RW	4.1.13.2	4–9	4–11
Asynchronous trap interrupt request register	ASTRR	RW	4.1.13.3	4–9	4–12
Hardware interrupt enable register	HIER	RW	4.1.13.4	4–10	4–13
Software interrupt enable register	SIER	RW	4.1.13.5	4–10	4–14
Asynchronous trap interrupt enable register	ASTER	RW	4.1.13.6	4–10	4–15

Figure 4–9 shows the HIRR, SIRR, and ASTRR read format, and Figure 4–10 shows the HIER, SIER, and ASTER read format.

Figure 4–9 HIRR, SIRR, and ASTRR Read Format

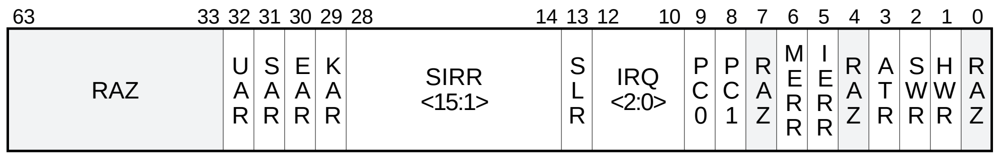


Figure 4–10 HIER, SIER, and ASTER Read Format

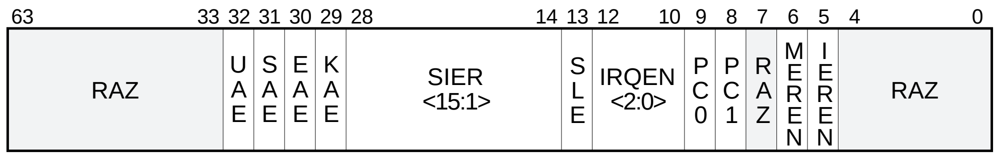


Table 4–14 describes the HIRR, SIRR, and ASTRR read-format fields.

Table 4–14 HIRR, SIRR, and ASTRR Read-Format Field Description

Bits	Field	Type*	Description
63:33	RAZ	RW	Read as zero.
32	UAR	RW	User AST request—When set, this bit indicates a user mode asynchronous trap (AST) interrupt request. (See Table 4–18.)
31	SAR	RW	Supervisor AST request—When set, this bit indicates a supervisor mode AST interrupt request. (See Table 4–18.)
30	EAR	RW	Executive AST request—When set, this bit indicates an executive mode AST interrupt request. (See Table 4–18.)
29	KAR	RW	Kernel AST request—When set, this bit indicates a kernel mode AST interrupt request. (See Table 4–18.)
28:14	SIRR <15:1>	RW	Software interrupt request—When any of these bits is set, it indicates a corresponding software interrupt request. (See Table 4–17.)
13	SLR	RW	Serial line interrupt request—When set, this bit indicates a serial line interrupt is requested (Section 4.1.11). The SL_CLR register (Section 4.1.10) clears this interrupt.
12:10	IRQ <2:0>	RW	Interrupt request—When any of these bits is set, it indicates an interrupt request on the corresponding **irq<2:0>** pin.
9	PC0	RW	Performance counter 0 interrupt request—When set, this bit indicates a performance counter 0 interrupt is requested (Section 4.1.7). The SL_CLR register (Section 4.1.10) clears this interrupt.
8	PC1	RW	Performance counter 1 interrupt request—When set, this bit indicates a performance counter 1 interrupt is requested (Section 4.1.7). The SL_CLR register (Section 4.1.10) clears this interrupt.
7	RAZ	RW	Read as zero.
6	MERR	RW	Memory controller error interrupt request—When set, this bit indicates a memory controller error interrupt request. The memory controller error status register (ESR, Section 5.6.6) clears this interrupt.

*The HIRR is read-only (RO).

(continued on next page)

Table 4–14 (Cont.) HIRR, SIRR, and ASTRR Read-Format Field Description

Bits	Field	Type*	Description
5	IERR	RW	IOC error interrupt request—When set, this bit indicates an IOC error interrupt request. The IOC status register 0 (IOC_STAT0, Section 6.4.3) clears this interrupt.
4	RAZ	RW	Read as zero.
3	ATR	RW	AST interrupt request—This bit is set if any of the AST request (bits <32:29>) and the corresponding enable bit are set. This bit also requires that the processor mode be equal to or higher than the request mode. Bit SIER2 must be set in the software interrupt enable register (SIER, Section 4.1.13.5) to enable AST interrupt requests.
2	SWR	RW	Software interrupt request—This bit is set if any software interrupt request (bits <28:14>) and the corresponding enable bit are set.
1	HWR	RW	Hardware interrupt request—This bit is set if any hardware interrupt request (bits <13:8,6,5>) and the corresponding enable bits are set.
0	RAZ	RW	Read as zero.

*The HIRR is read-only (RO).

Table 4–15 describes the HIER, SIER, and ASTER read-format fields.

Table 4–15 HIER, SIER, and ASTER Read-Format Field Description

Bits	Field	Type	Description
63:33	RAZ	RW	Read as zero.
32	UAE	RW	User AST enable—When set, this bit indicates that user mode AST interrupts are enabled.
31	SAE	RW	Supervisor AST enable—When set, this bit indicates that supervisor mode AST interrupts are enabled.
30	EAE	RW	Executive AST enable—When set, this bit indicates that executive mode AST interrupts are enabled.
29	KAE	RW	Kernel AST enable—When set, this bit indicates that kernel mode AST interrupts are enabled.

(continued on next page)

Table 4–15 (Cont.) HIER, SIER, and ASTER Read-Format Field Description

Bits	Field	Type	Description
28:14	SIER <15:1>	RW	Software interrupt enable—When any of these bits is set, it indicates that the corresponding software interrupt is enabled.
13	SLE	RW	Serial line interrupt enable—When set, this bit indicates that serial line interrupts are enabled.
12:10	IRQEN <2:0>	RW	Interrupt request enable—When any of these bits is set, it indicates that interrupts are enabled on the corresponding **irq<2:0>** pin.
9	PC0	RW	Performance counter 0 interrupt enable—When set, this bit indicates that performance counter 0 interrupts are enabled.
8	PC1	RW	Performance counter 1 interrupt enable—When set, this bit indicates that performance counter 1 interrupts are enabled.
7	RAZ	RW	Read as zero.
6	MEREN	RW	Memory controller error interrupt enable—When set, this bit indicates that memory controller error interrupts are enabled.
5	IEREN	RW	IOC error interrupt enable—When set, this bit indicates that IOC error interrupts are enabled.
4:0	RAZ	RW	Read as zero.

Table 4–16 summarizes how the interrupt request and enable bits are mapped to the registers on read and write operations.

Table 4–16 Interrupt Request and Enable Registers Bit Map

Field	Read* Request Bits	Read† Enable Bits	Write Bits
UAR	32	—	ASTRR 51
UAE	—	32	ASTER 51
SAR	31	—	ASTRR 50
SAE	—	31	ASTER 50
EAR	30	—	ASTRR 49
EAE	—	30	ASTER 49
KAR	29	—	ASTRR 48
KAE	—	29	ASTER 48
SIRR<15:0>	28:14	—	SIRR 47:33
SIER<15:0>	—	28:14	SIER 47:33
SLR	13	—	—
SLE	—	13	HIER 32
IRQ<2:0>	12:10	—	—
IRQEN<2:0>	—	12:10	—
PC0	9	9	HIER 15
PC1	8	8	HIER 8
MERR	6	—	—
MERREN	—	6	HIER 13
IERR	5	—	—
IERREN	—	5	HIER 12
ATR	3	—	—
SWR	2	—	—
HWR	1	—	—

*HIRR, SIRR, ASTRR bits

†HIER, SIER, ASTER bits

The HIRR, SIRR, SIER, ASTRR, and ASTER write formats are shown in Figures 4–11 through 4–15 and the fields are described in Tables 4–17 through 4–21.

4.1.13.1 Hardware Interrupt Request Register

The read-only hardware interrupt request register (HIRR) indicates all outstanding interrupt requests and summary bits at the time it is read. Each bit in the HIRR corresponds to a bit that must be set in the hardware interrupt enable register (HIER, Section 4.1.13.4) to enable an interrupt request. When it is read, the HIRR also returns the state of software and AST interrupt requests.

Note

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

All interrupt requests are blocked while executing in PALmode. (For more information about interrupt operations, see Section 2.2.3.)

Figure 4–9 shows the HIRR format, and Table 4–14 describes its fields.

4.1.13.2 Software Interrupt Request Register

The read and write software interrupt request register (SIRR) controls software interrupt requests. Each bit in the SIRR corresponds to a bit that must be set in the software interrupt enable register (SIER, Section 4.1.13.5) to enable an interrupt request. When it is read, the SIRR also returns the state of hardware and AST interrupt requests.

All interrupt requests are blocked while executing in PALmode. (For more information about interrupt operations, see Section 2.2.3.)

Figure 4–9 shows the SIRR read format, and Table 4–14 describes its fields. Figure 4–11 shows the SIRR write format, and Table 4–17 describes its fields.

Figure 4–11 SIRR Write Format

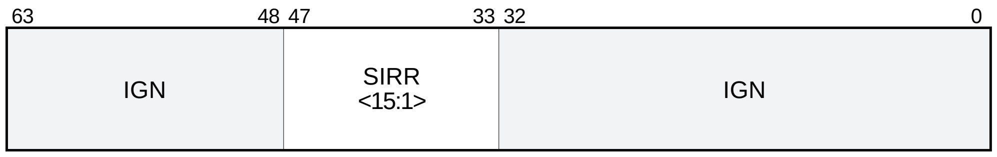


Table 4–17 SIRR Write-Format Field Description

Bits	Field	Type	Description
63:48	IGN	RW	Ignored when written.
47:33	SIRR <15:1>	RW	Software interrupt request—These bits correspond to software interrupt enable 15 through 1 in the SIER (Section 4.1.13.5). Writing a one to any of these bits when the corresponding enable bit is set, requests the corresponding interrupt. Writing a zero to any of these bits clears the corresponding interrupt request.
32:0	IGN	RW	Ignored when written.

4.1.13.3 Asynchronous Trap Interrupt Request Register

The read and write asynchronous trap interrupt request register (ASTRR) contains AST interrupt request bits for each processor mode. To generate an AST interrupt, the following must occur:

- The corresponding enable bit must be set in the asynchronous trap enable register (ASTER, Section 4.1.13.6).
- The processor must be in the selected processor mode or a higher privileged mode according to the current value of current mode (CM) bits in the processor status register (PS, Section 4.1.15).

In addition, to enable AST interrupt requests, bit SIER2 must be set in the software interrupt enable register (SIER, Section 4.1.13.5). This provides a mechanism to lock out AST requests over certain interrupt priority levels (IPLs).

When it is read, the ASTRR also returns the state of software and hardware interrupt requests.

All interrupt requests are blocked while executing in PALmode. (For more information about interrupt operations, see Section 2.2.3.)

Figure 4–9 shows the ASTRR read format, and Table 4–14 describes its fields. Figure 4–12 shows the ASTRR write format, and Table 4–18 describes its fields.

Figure 4–12 ASTRR Write Format

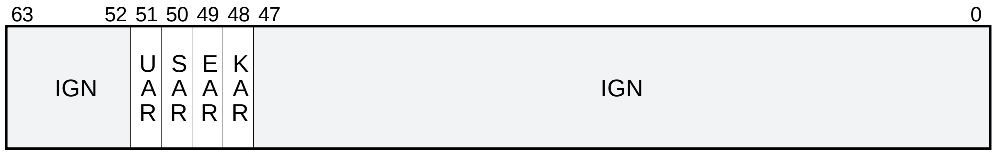


Table 4–18 ASTRR Write-Format Field Description

Bits	Field	Type	Description
63:52	IGN	RW	Ignored when written.
51	UAR	RW	User mode AST request—This bit corresponds to user mode AST interrupt enable. Writing a one to this bit when the corresponding enable bit is set, requests the interrupt. Writing a zero to this bit clears the interrupt.
50	SAR	RW	Supervisor mode AST request—This bit corresponds to supervisor mode AST interrupt enable. Writing a one to this bit when the corresponding enable bit is set, requests the interrupt. Writing a zero to this bit clears the interrupt.
49	EAR	RW	Executive mode AST request—This bit corresponds to executive mode AST interrupt enable. Writing a one to this bit when the corresponding enable bit is set, requests the interrupt. Writing a zero to this bit clears the interrupt.
48	KAR	RW	Kernel mode AST request—This bit corresponds to kernel mode AST interrupt enable. Writing a one to this bit when the corresponding enable bit is set, requests the interrupt. Writing a zero to this bit clears the interrupt.
47:0	IGN	RW	Ignored when written.

4.1.13.4 Hardware Interrupt Enable Register

The read and write hardware interrupt enable register (HIER) enables the hardware interrupt requests described in Table 4–19. There is a one-to-one correspondence between the interrupt request and enable bits. When set, a bit in this register enables the corresponding interrupt request in the HIRR (Section 4.1.13.1); otherwise, the request is disabled.

When read, the HIER returns all the interrupt enables described in Table 4–15.

Figure 4–10 shows the HIER read format, and Table 4–15 describes its fields. Figure 4–13 shows the HIER write format, and Table 4–19 describes its fields.

Figure 4–13 HIER Write Format

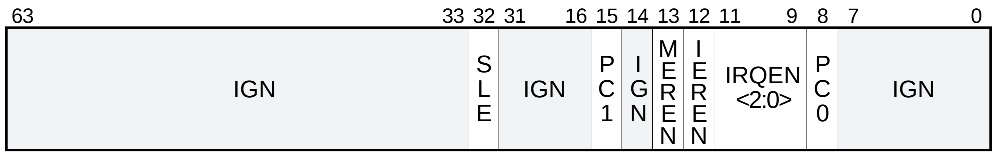


Table 4–19 HIER Write-Format Field Description

Bits	Field	Type	Description
63:33	IGN	RW	Ignored when written.
32	SLE	RW	Serial line interrupt enable—When set, this bit enables serial line interrupts.
31:16	IGN	RW	Ignored when written.
15	PC1	RW	Performance counter 1 interrupt enable—When set, this bit enables performance counter 1 interrupts.
14	IGN	RW	Ignored when written.
13	MEREN	RW	Memory controller error interrupt enable—When set, this bit enables memory controller error interrupts.
12	IEREN	RW	IOC error interrupt enable—When set, this bit enables IOC error interrupts.
11:9	IRQEN <2:0>	RW	Interrupt request enable—When any of these bits is set, interrupts are enabled on the corresponding **irq<2:0>** pin.
8	PC0	RW	Performance counter 0 interrupt enable—When set, this bit enables performance counter 0 interrupts.
7:0	IGN	RW	Ignored when written.

4.1.13.5 Software Interrupt Enable Register

The read and write software interrupt enable register (SIER) enables the software interrupt requests described in Table 4–20. There is a one-to-one correspondence between the interrupt request and enable bits. When set, a bit in this register enables the corresponding interrupt request in the SIRR (Section 4.1.13.2); otherwise, the request is disabled.

When read, the SIER returns all the interrupt enables described in Table 4–15.

Figure 4–10 shows the SIER read format, and Table 4–15 describes its fields. Figure 4–14 shows the SIER write format, and Table 4–20 describes its fields.

Figure 4–14 SIER Write Format

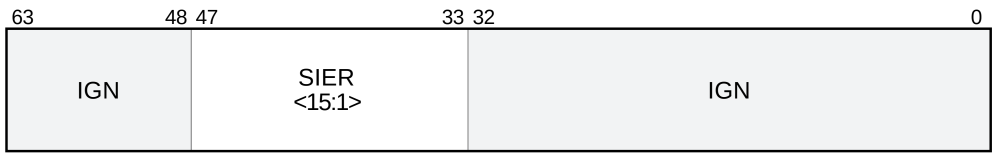


Table 4–20 SIER Write-Format Field Description

Bits	Field	Type	Description
63:48	IGN	RW	Ignored when written.
47:33	SIER <15:1>	RW	Software interrupt enable—When any of these bits is set, the corresponding software interrupt is enabled.
32:0	IGN	RW	Ignored when written.

4.1.13.6 Asynchronous Trap Interrupt Enable Register

The read and write asynchronous trap interrupt enable register (ASTER) enables the AST interrupt requests described in Table 4–21. There is a one-to-one correspondence between the interrupt request and enable bits. When set, a bit in this register enables the corresponding interrupt request in the ASTRR (Section 4.1.13.3); otherwise, the request is disabled.

When read, the ASTER returns all the interrupt enables described in Table 4–15.

Figure 4–10 shows the ASTER read format, and Table 4–15 describes its fields. Figure 4–15 shows the ASTER write format, and Table 4–21 describes its fields.

Figure 4–15 ASTER Write Format

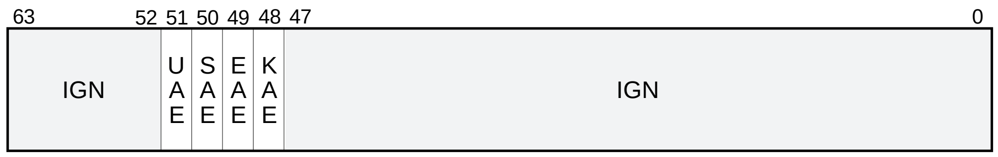


Table 4–21 ASTER Write-Format Field Description

Bits	Field	Type	Description
63:52	IGN	RW	Ignored when written.
51	UAE	RW	User AST enable—When set, this bit enables user mode AST interrupts.
50	SAE	RW	Supervisor AST enable—When set, this bit enables supervisor mode AST interrupts.
49	EAE	RW	Executive AST enable—When set, this bit enables executive mode AST interrupts.
48	KAE	RW	Kernel AST enable—When set, this bit enables kernel mode AST interrupts.
47:0	IGN	RW	Ignored when written.

4.1.14 Performance Counters

The performance counters are reset to zero at power up; otherwise, they are never cleared. The counters count events over a long period of time relative to the event frequency; intermediate counter values cannot be extracted.

Because the counters continuously accumulate a count of selected events, despite interrupts being enabled, the first interrupt after selecting a new counter input has an error boundary as large as the specified overflow range. The overflow range is specified in bits 3 and 0 of the ICCSR (Section 4.1.7).

Some inputs can overcount events that occur simultaneously with Dstream errors that abort the event late in the pipeline. For example, when counting load instructions, an attempt to execute a load instruction that results in an ITB_MISS exception increments the performance counter after the first aborted execution attempt, and then increments the counter again after the TB-fill routine when the load instruction reissues and completes.

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

The six cycles before an interrupt is triggered implies that as many as 12 instructions might have completed before the start of the interrupt service routine.

When counting Icache misses, intervening instructions cannot complete and the exception PC (EXC_ADDR <63:2>, Section 4.1.8) contains the address of the last Icache miss. Branch mispredictions allow no more than two instructions to complete before start of the interrupt service routine.

Table 4–22 lists performance counter 0 inputs and Table 4–23 lists performance counter 1 inputs.

Table 4–22 Counter 0 Input Selection

PCMUX0 <3:0>	Input	Events Counted
000X	Total issues÷2	Total issues divided by 2; a dual issue increments the count by 1.
000X	Pipeline dry	Cycles where nothing issued due to lack of valid Istream data. Causes include Icache fill, mispredictions, branch delay slots, and pipeline drain for exception.
010X	Load instructions	All load instructions.
011X	Pipeline frozen	Cycles where nothing issued due to resource conflict.
100X	Branch instructions	All conditional branches, unconditional branches, JSR, and HW_REI instructions.
1011	PALmode	Cycles while executing in PALmode.
1010	Total cycles	Total cycles.
110X	Total nonissues÷2	Total nonissues divided by 2; no issue increments the count by 1.

X = 1 or 0

(continued on next page)

Table 4–22 (Cont.) Counter 0 Input Selection

PCMUX0 <3:0>	Input	Events Counted
111X	Memory controller EAR	External interface events specified in the memory controller EAR. (See Section 5.6.7 for more information about the events that can be monitored.)

X = 1 or 0

Table 4–23 Counter 1 Input Selection

PCMUX1 <2:0>	Input	Events Counted
000	Dcache miss	Total Dcache misses.
001	Icache miss	Total Icache misses.
010	Dual issues	Cycles of dual issue.
011	Branch mispredic-tions	Both conditional branch mispredictions and JSR or HW_REI mispredictions. Conditional branch mispredictions cause four cycles and others cause five cycles of dry pipeline delay.
100	FP instructions	Total floating-point operate instructions; that is, not floating-point branch, load, or store instructions.
101	Integer operate	Integer operate instructions including LDA and LDAH with destination other than R31.
110	Store instructions	Total store instructions.
111	Memory controller EAR	External interface events specified in the memory controller EAR. (See Section 5.6.7 for more information about the events that can be monitored.)

4.1.15 Processor Status Register

The read and write processor status (PS) register implements the current mode bits of the architecturally defined processor status (PS) register.

Figure 4–16 shows the register formats, and Tables 4–24 and 4–25 describe the fields.

Figure 4–16 PS Register Formats

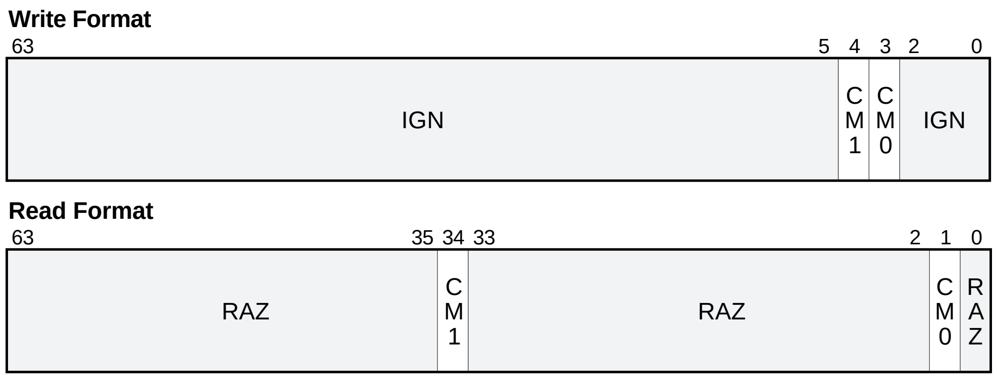


Table 4–24 PS Register Write-Format Field Description

Bits	Field	Type	Description
63:5	IGN	RW	Ignored when written.
4 3	CM1 CM0	RW	Current mode—The access mode of the currently executing process is as follows: **<4:3>** **Mode** 00 Kernel 01 Executive 10 Supervisor 11 User
2:0	IGN	RW	Ignored when written.

Table 4–25 PS Register Read-Format Field Description

Bits	Field	Type	Description
63:35	RAZ	RW	Read as zero.
34	CM1	RW	See the corresponding write-format field description (Table 4–24).

(continued on next page)

Table 4–25 (Cont.) PS Register Read-Format Field Description

Bits	Field	Type	Description
33:2	RAZ	RW	Read as zero.
1	CM0	RW	See the corresponding write-format field description (Table 4–24).
0	RAZ	RW	Read as zero.

4.1.16 PAL Base Address Register

The read and write PAL base address (PAL_BASE) register contains the base address for PALcode. Hardware clears the register at reset.

Figure 4–17 shows the PAL_BASE register format, and Table 4–26 describes its fields.

Figure 4–17 PAL_BASE Register Format

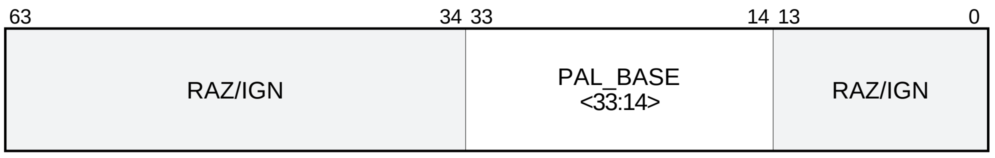


Table 4–26 PAL_BASE Register Field Description

Bits	Field	Type	Description
63:34	RAZ/IGN	RW	Read as zero. Ignored when written.
33:14	PAL_BASE <33:14>	RW	This field holds PALcode base address bits <33:14>.
13:0	RAZ/IGN	RW	Read as zero. Ignored when written.

4.2 LSU Internal Processor Registers

The LSU internal processor registers are described in Sections 4.2.1 through 4.2.14.

4.2.1 LSU Control Register

The write-only LSU control (ABOX_CTL) register is cleared when written with a value of zero.

Figure 4–18 shows the ABOX_CTL register format, and Table 4–27 describes its fields. The 21066A implements two additional bits that are described in Section A.6.1.

Figure 4–18 ABOX_CTL Register Format

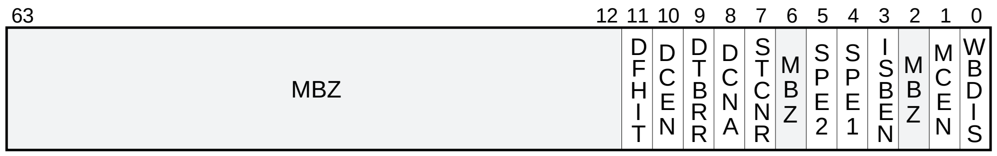


Table 4–27 ABOX_CTL Register Field Description

Bits	Field	Type	Description
63:12	MBZ	WO	Must be zero.
11	DFHIT	WO	Dcache force hit—When set, this bit forces all Dstream references to hit in the Dcache. This bit takes precedence over the DCEN bit (<10>). That is, when the DFHIT bit is set and the DCEN bit is clear, all Dstream references hit in the Dcache.
10	DCEN	WO	Dcache enable—When clear, this bit disables and flushes the Dcache. When set, this bit enables the Dcache.
9	DTBRR	WO	DTB round-robin enable—When this bit is set, the data translation buffer uses a round-robin replacement algorithm. When this bit is clear, a not-last-used algorithm is used.
8	DCNA	WO	Dcache, no allocate—Normally, this bit must be zero. When this bit is set, the Dcache line associated with a read address is not disturbed.

(continued on next page)

Table 4–27 (Cont.) ABOX_CTL Register Field Description

Bits	Field	Type	Description
7	STCNR	WO	Store conditional, no result—Normally, this bit must be zero. When this bit is set, it changes the way LDL_L, LDQ_L, STL_C, and STQ_C instructions are handled as follows: The results written in the register specified by the Ra field in STx_C and HW_ST/C instructions are UNPREDICTABLE. This allows the IDU to restart the memory reference pipeline when the STx_C is transferred from the write buffer to the memory controller, increasing the repetition rate with which STx/C instructions can be processed. LDx_L, STx_C, and HW_ST/C instructions invalidate the Dcache line associated with their generated address. The invalidated lines are not visible to load or store instructions that issue in the two CPU cycles after the LDL_L, LDQ_L, STL_C, STQ_C, or HW_ST/C is issued.
6	MBZ	WO	Must be zero.
5	SPE2	WO	Superpage enable 2—When set, this bit enables one-to-one superpage mapping of Dstream virtual addresses VA<33:13> directly to physical addresses PA<33:13>, when virtual address VA<42:41> = 2. VA<40:34> are ignored in this translation. Access is allowed only in kernel mode.
4	SPE1	WO	Superpage enable 1—When set, this bit enables one-to-one superpage mapping of Dstream virtual addresses with VA<42:30> = 1FFE to physical addresses with PA<33:30> = 0. Access is allowed only in kernel mode.
3	ISBEN	WO	Icache stream buffer enable—When set, this bit enables operation of a single-entry Icache stream buffer.
2	MBZ	WO	Must be zero.
1	MCEN	WO	Machine check enable—When this bit is set, the load and store unit generates a machine check when the hardware encounters errors that it cannot correct. When this bit is cleared, uncorrectable errors do not cause a machine check; however, the Dcache status register (DC_STAT, Section 4.4) is updated and locked when such errors occur.

(continued on next page)

Table 4–27 (Cont.) ABOX_CTL Register Field Description

Bits	Field	Type	Description
0	WBDIS	WO	Write buffer unload disable—When set, this bit prevents the write buffer from sending write data to the memory controller. This bit should be set only for diagnostics.

4.2.2 Translation Buffer Control Register

The write-only translation buffer control (TB_CTL) register contains the granularity hint (GH) field. The GH field (<6:5>) specifies the size (number of 8-KB pages that the translation buffer can map as a single page) as follows:

GH Field	Size
0	1
1	8
2	64
3	512

The DTB supports all four sizes. The ITB supports only two sizes, as follows:

- When GH = 3, the ITB supports 4-MB (512 × 8 KB) pages.
- When GH = 2, 1, or 0, the ITB supports 8-KB pages.

The GH field affects DTB and ITB read and write operations.

Figure 4–19 shows the TB_CTL register format, and Table 4–28 describes its field.

Figure 4–19 TB_CTL Register Format

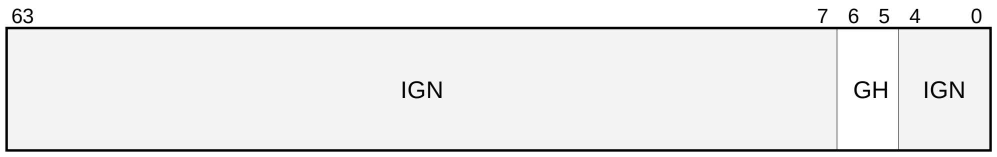


Table 4–28 TB_CTL Register Field Description

Bits	Field	Type	Description
63:7	IGN	WO	Ignored when written.
6:5	GH	WO	Granularity hint.
4:0	IGN	WO	Ignored when written.

4.2.3 Data Translation Buffer Page Table Entry Register

The read and write data translation buffer page table entry (DTB_PTE) register represents the 32-entry DTB. The entry to be written is chosen by a not-last-used (NLU) algorithm implemented in the hardware.

The DTB_PTE register is written using the format described in the *Alpha Architecture Reference Manual*, but some fields are ignored. The valid bit (V) is not represented in hardware.

The DTB tag array is updated from the TB_TAG register (Section 4.1.1) when the DTB_PTE register is written. Two instructions are required to read the DTB_PTE register:

1. The first instruction sends the PTE data to the data translation buffer page table entry temporary register (DTB_PTE_TEMP, Section 4.2.4).
2. The second instruction reads the DTB_PTE_TEMP register and returns the PTE to the register file.

To access the complete set of DTB_PTE register entries, the TB entry pointer is incremented when the DTB_PTE register is read or written.

Figure 4–20 shows the DTB_PTE register format, and Table 4–29 describes its fields.

Figure 4–20 DTB_PTE Register Format

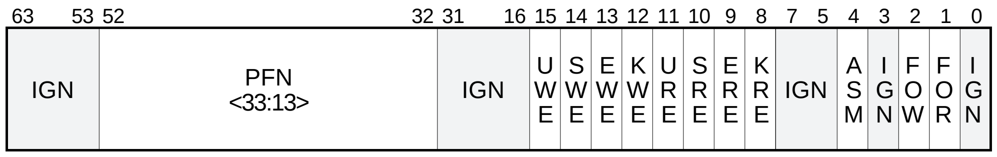


Table 4–29 DTB_PTE Register Field Description

Bits	Field	Type	Description*
63:53	IGN	RW	Ignored when written.
52:32	PFN <33:13>	RW	Page frame number—This field always points to a page boundary.
31:16	IGN	RW	Ignored when written.
15	UWE	RW	User write-enable—When set, this bit enables user mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in user mode.
14	SWE	RW	Supervisor write-enable—When set, this bit enables supervisor mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in supervisor mode.
13	EWE	RW	Executive write-enable—When set, this bit enables executive mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in executive mode.
12	KWE	RW	Kernel write-enable—When set, this bit enables kernel mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in kernel mode.
11	URE	RW	User read-enable—When set, this bit enables user mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in user mode.
10	SRE	RW	Supervisor read-enable—When set, this bit enables supervisor mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in supervisor mode.
9	ERE	RW	Executive read-enable—When set, this bit enables executive mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in executive mode.
8	KRE	RW	Kernel read-enable—When set, this bit enables kernel mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in kernel mode.
7:5	IGN	RW	Ignored when written.

*For more information about the fields, see the *Alpha Architecture Reference Manual*.

(continued on next page)

Table 4–29 (Cont.) DTB_PTE Register Field Description

Bits	Field	Type	Description*
4	ASM	RW	Address space match—When this bit is set, this PTE matches all address space numbers (ASNs). For a given virtual address, this bit must be set consistently in all processes; otherwise, the address mapping is UNPREDICTABLE.
3	IGN	RW	Ignored when written.
2	FOW	RW	Fault on write. This bit is set if the reference was a write operation and the PTE FOW bit was set.
1	FOR	RW	Fault on read. This bit is set if the reference was a read operation and the PTE FOR bit was set.
0	IGN	RW	Ignored when written.

*For more information about the fields, see the *Alpha Architecture Reference Manual*.

4.2.4 Data Translation Buffer Page Table Entry Temporary Register

The data translation buffer page table entry temporary (DTB_PTE_TEMP) register is a read-only holding register for DTB_PTE read data. The following two instructions are required to read the DTB_PTE register (Section 4.2.3) and return data to the integer register file (IRF):

1. Read the DTB_PTE register data to the DTB_PTE_TEMP register.
2. Read the DTB_PTE_TEMP register data to the IRF.

Figure 4–21 shows DTB_PTE_TEMP register format, and Table 4–30 describes its fields.

Figure 4–21 DTB_PTE_TEMP Register Format

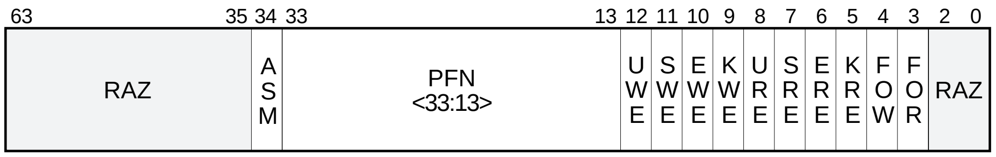


Table 4–30 DTB_PTE_TEMP Register Field Description

Bits	Field	Type	Description*
63:35	RAZ	RO	Read as zero.
34	ASM	RW	Address space match—When this bit is set, this PTE matches all address space numbers (ASNs). For a given virtual address, this bit must be set consistently in all processes; otherwise, the address mapping is UNPREDICTABLE.
33:13	PFN <33:13>	RW	Page frame number—This field always points to a page boundary.
12	UWE	RW	User write-enable—When set, this bit enables user mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in user mode.
11	SWE	RW	Supervisor write-enable—When set, this bit enables supervisor mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in supervisor mode.
10	EWE	RW	Executive write-enable—When set, this bit enables executive mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in executive mode.
9	KWE	RW	Kernel write-enable—When set, this bit enables kernel mode write operations. When this bit is clear, an ACV occurs if a store instruction is attempted in kernel mode.
8	URE	RW	User read-enable—When set, this bit enables user mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in user mode.
7	SRE	RW	Supervisor read-enable—When set, this bit enables supervisor mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in supervisor mode.
6	ERE	RW	Executive read-enable—When set, this bit enables executive mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in executive mode.
5	KRE	RW	Kernel read-enable—When set, this bit enables kernel mode read operations. When this bit is clear, an ACV occurs if a load or instruction fetch is attempted in kernel mode.

*For more information about the fields, see the *Alpha Architecture Reference Manual*.

(continued on next page)

Table 4–30 (Cont.) DTB_PTE_TEMP Register Field Description

Bits	Field	Type	Description*
4	FOW	RW	Fault on write. This bit is set if the reference was a write operation and the PTE FOW bit was set.
3	FOR	RW	Fault on read. This bit is set if the reference was a read operation and the PTE FOR bit was set.
2:0	RAZ	RO	Read as zero.

*For more information about the fields, see the *Alpha Architecture Reference Manual*.

4.2.5 Data Translation Buffer ZAP Register

A write operation to the data translation buffer ZAP (DTBZAP) register invalidates all 32 DTB entries. It also resets the not-last-used (NLU) pointer to its initial state.

4.2.6 Data Translation Buffer ASM Register

A write operation to the data translation buffer ASM (DTBASM) register invalidates all 32 DTB entries in which the ASM address space match (ASM) bit is equal to zero.

4.2.7 Data Translation Buffer Invalidate Single Register

A write operation to the data translation buffer invalidate single (DTBIS) register invalidates the DTB entry that maps the virtual address held in the integer register. The integer register is identified by the Rb field of the HW_MTPR instruction used to perform the write operation.

4.2.8 Virtual Address Register

When Dstream faults or DTB misses occur, the effective virtual address associated with the fault or miss is latched in the read-only virtual address (VA) register. The VA register and memory-management CSR (MM_CSR, Section 4.2.9) are locked against further updates until the software reads the VA register. The VA register is unlocked after reset. PALcode must explicitly unlock the VA register when its entry point is higher in priority than a DTB miss.

Figure 4–22 shows the VA register format, and Table 4–31 describes its field.

Figure 4–22 VA Register Format

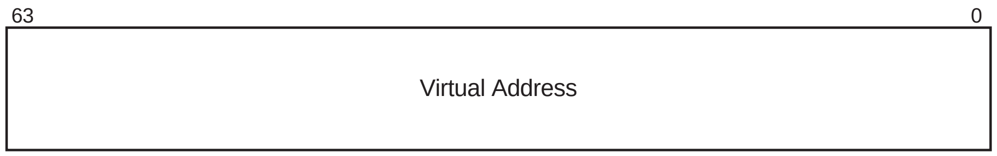


Table 4–31 VA Register Field Description

Bits	Field	Type	Description
63:0	Virtual Address	RO	This field is the effective virtual address associated with a Dstream fault or DTB miss.

4.2.9 Memory Management Control and Status Register

When Dstream faults occur, the information about the fault is latched and saved in the read-only memory management control and status register (MM_CSR). The VA register (Section 4.2.8) and the MM_CSR are locked against further updates until the software reads the VA register. PALcode must explicitly unlock this register when the entry point of the fault is higher in priority than a DTB miss. The MM_CSR bits are modified by the hardware only when the register is not locked and a memory-management error or a DTB miss occurs. The MM_CSR is unlocked after reset.

Figure 4–23 shows the MM_CSR format, and Table 4–32 describes its fields.

Figure 4–23 MM_CSR Format

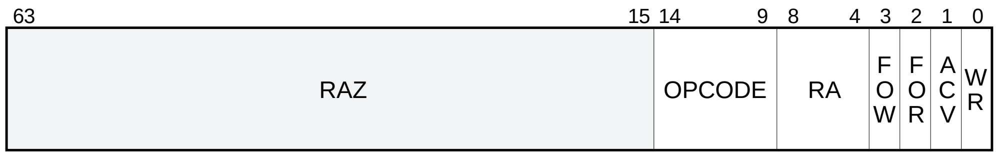


Table 4–32 MM_CSR Field Description

Bits	Field	Type	Description
63:15	RAZ	RO	Read as zero.
14:9	OPCODE	RO	This field holds the opcode field of the faulting instruction.
8:4	RA	RO	This field holds the Ra field of the faulting instruction.
3	FOW	RO	Fault on write—This bit is set if the reference was a write operation and the PTE FOW bit was set.
2	FOR	RO	Fault on read—This bit is set if the reference was a read operation and the PTE FOR bit was set.
1	ACV	RO	Access violation—This bit is set if the reference caused an access violation.
0	WR	RO	Write reference—This bit is set if a write reference caused the error.

4.2.10 Flush Instruction Cache Register

A write operation to the flush instruction cache (FLUSH_IC) register flushes the entire instruction cache.

4.2.11 Flush Instruction Cache ASM Register

A write operation to the flush instruction cache ASM (FLUSH_IC_ASM) register invalidates all Icache blocks in which the ASM bit is clear.

4.2.12 Alternate Processor Mode Register

The alternate processor mode (ALT_MODE) register is a write-only register. The AM field specifies the alternate processor mode used by HW_LD and HW_ST instructions in which the ALT bit (<14>) is set.

Figure 4–24 shows the ALT_MODE register format, and Table 4–33 describes its fields.

Figure 4–24 ALT_MODE Register Format

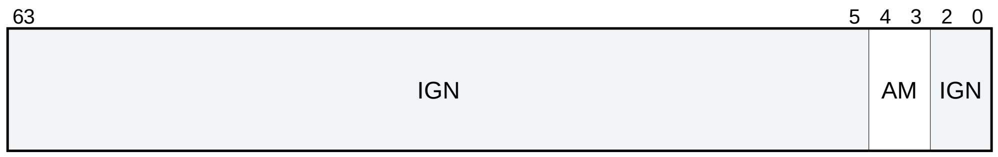


Table 4–33 ALT_MODE Register Field Description

Bits	Field	Type	Description
63:5	IGN	WO	Ignored when written.
4:3	AM	WO	Alternate processor mode—This field is interpreted as follows: **<4:3>** **Mode** 00 Kernel 01 Executive 10 Supervisor 11 User
2:0	IGN	WO	Ignored when written.

4.2.13 Cycle Counter Register

The microprocessor supports a cycle counter as described in the *Alpha Architecture Reference Manual*. When enabled, the read and write cycle counter (CC) register increments once each CPU cycle. The instruction, `HW_MTPR Rn, CC`, writes the value in Rn bits <63:32> to CC register bits <63:32>; CC register bits <31:0> are not changed. This register is read by the read process cycle counter (RPCC) instruction.

Figure 4–25 shows the CC register formats, and Tables 4–34 and 4–35 describe the fields.

Figure 4–25 CC Register Formats

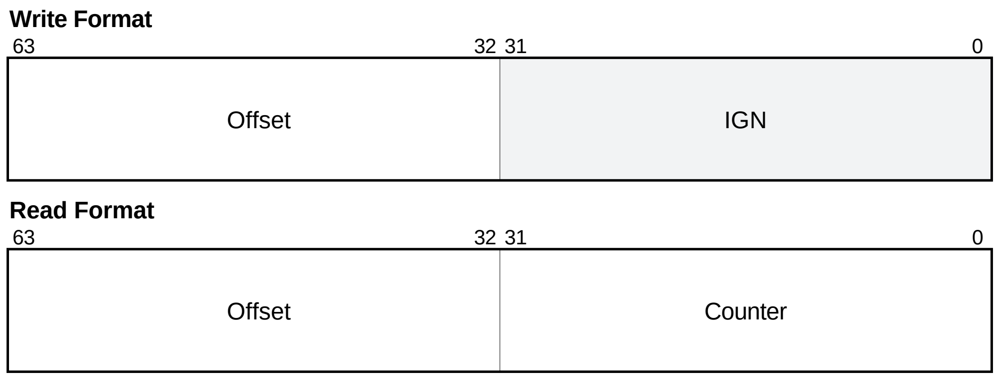


Table 4–34 CC Register Write-Format Field Description

Bits	Field	Type	Description
63:32	Offset	RW	When an HW_MTPR Rn, CC instruction writes the CC register. This field holds the value from Rn bits <63:32>.
31:0	IGN	RW	Ignored when written.

Table 4–35 CC Register Read-Format Field Description

Bits	Field	Type	Description
63:32	Offset	RW	See the write-format description (Table 4–34).
31:0	Counter	RW	This field holds the process cycle count as defined in the *Alpha Architecture Reference Manual*.

4.2.14 Cycle Counter Control Register

The cycle counter control (CC_CTL) register is a write-only register. The instruction, `HW_MTPR Rn, CC_CTL`, writes the value in Rn bits <31:0> to CC register bits <31:0>; CC register bits <63:32> are not changed and CC register bits <3:0> must be written with zero. If Rn bit 32 is set, the cycle counter is enabled; otherwise, it is disabled.

Figure 4–26 shows the CC_CTL register format, and Table 4–36 describes its fields.

Figure 4–26 CC_CTL Register Format

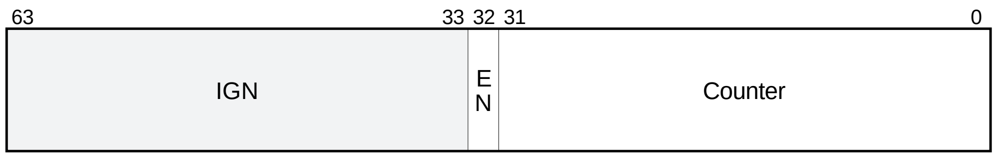


Table 4–36 CC_CTL Register Field Description

Bits	Field	Type	Description
63:33	IGN	WO	Ignored when written.
32	EN	WO	Enable—This bit is set to enable the cycle counter if Rn bit 32 is set; otherwise, the counter is disabled.
31:0	Counter	WO	This field holds the value from Rn bits <31:0> when an HW_MTPR Rn, CC_CTL instruction writes the CC_CTL register.

4.3 PALcode Temporary Registers

The 32 PALcode temporary (PAL_TEMP<31:0>) registers provide temporary storage for PALcode and are accessible by the HW_MFPR and HW_MTPR instructions.

4.4 Data Cache Status Register

The read-only data cache status (DC_STAT) register is for use only by the diagnostics and is used by the 21066. The 21066A implements the cache status (C_STAT) register (refer to Section A.6.3).

Figure 4–27 shows the DC_STAT register format, and Table 4–37 describes its fields.

Figure 4–27 DC_STAT Register Format

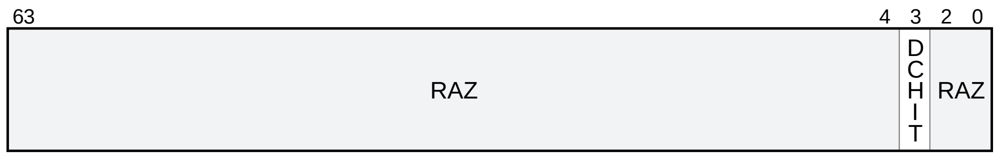


Table 4–37 DC_STAT Register Field Description

Bits	Field	Type	Description
63:4	RAZ	RO	Read as zero.
3	DCHIT	RO	Dcache hit—This bit indicates whether the last load or store instruction processed by the LSU hit (DCHIT set) or missed (DCHIT clear) the Dcache. Loads that miss the Dcache can be completed without requiring external read operations.
2:0	RAZ	RO	Read as zero.

4.5 Internal Processor Registers Reset State

Table 4–38 lists the state of all the internal processor registers (IPRs) immediately after reset. The table also specifies which registers must be initialized by power-up PALcode.

Table 4–38 Internal Process Register Reset State

IPR	Reset State	Comments
ABOX_CTL	Cleared	<11:0> Write buffer enabled, machine checks disabled, Icache stream buffer disabled, superpages 1 and 2 disabled, and Dcache forced hit mode off.
ALT_MODE	UNDEFINED	—
ASTER	UNDEFINED	PALcode must initialize.
ASTRR	UNDEFINED	PALcode must initialize.
CC	UNDEFINED	Cycle counter is disabled on reset.

NA = not applicable

(continued on next page)

Table 4–38 (Cont.) Internal Process Register Reset State

IPR	Reset State	Comments
CC_CTL	UNDEFINED	—
DC_STAT	UNDEFINED	—
DTBASM	NA	—
DTBIS	NA	—
DTB_PTE	UNDEFINED	—
DTB_PTE_TEMP	UNDEFINED	—
DTBZAP	NA	On reset, PALcode must do an HW_MTPR instruction to the DTBZAP register before writing the DTB.
EXC_ADDR	UNDEFINED	—
EXC_SUM	UNDEFINED	PALcode must do 64 read operations to clear the exception register write mask.
FLUSH_IC	UNDEFINED	—
FLUSH_IC_ASM	UNDEFINED	—
HIER	UNDEFINED	PALcode must initialize.
HIRR	NA	—
ICCSR	Cleared except ASN, PC0, PC1	Floating-point disabled, single issue mode, pipe mode enabled, JSR predictions disabled, branch predictions disabled, branch history table disabled, performance counters reset to zero, performance counter 0 = total issues÷2, performance counter 1 = Dcache misses, superpage disabled.
ITBASM	NA	—
ITBIS	NA	—
ITB_PTE	UNDEFINED	—
ITB_PTE_TEMP	UNDEFINED	—
ITBZAP	NA	On reset, PALcode must do an HW_MTPR instruction to the ITBZAP register before writing the ITB.
MM_CSR	UNDEFINED	Unlocked on reset.
PAL_BASE	Cleared	Cleared on reset.

NA = not applicable

(continued on next page)

Table 4–38 (Cont.) Internal Process Register Reset State

IPR	Reset State	Comments
PAL_TEMP<31:0>	UNDEFINED	—
PS	UNDEFINED	PALcode must set processor status.
SIER	UNDEFINED	PALcode must initialize.
SIRR	UNDEFINED	PALcode must initialize.
SL_CLR	UNDEFINED	PALcode must initialize.
SL_RCV	UNDEFINED	—
SL_XMIT	UNDEFINED	PALcode must initialize. The TMT bit appears on an external pin.
TB_TAG	UNDEFINED	—
TB_CTL	UNDEFINED	PALcode must select between small-page and large-page DTB prior to any TB fill.
VA	UNDEFINED	Unlocked on reset.

NA = not applicable

5 Memory Controller

This chapter describes the memory controller, supported memories, memory operations, error handling, graphics operations, registers, cycle timing, and interface signals.

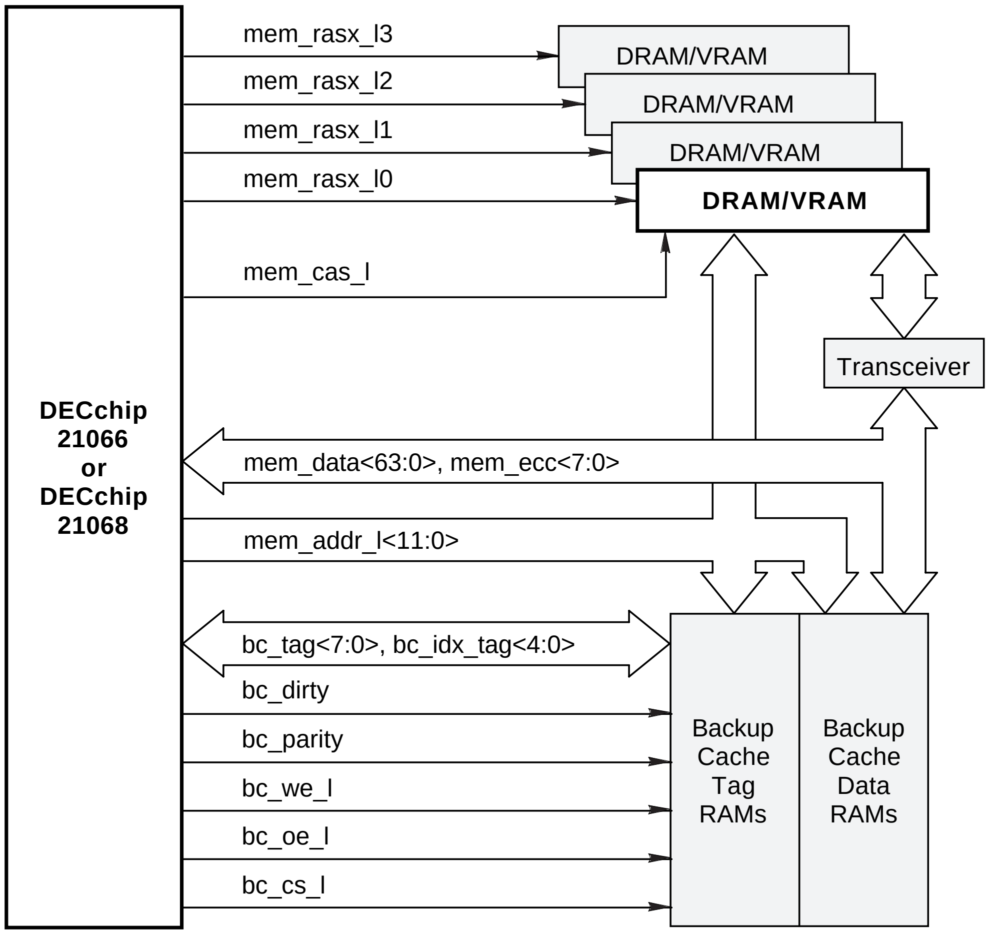
5.1 Overview

The memory controller controls the external, primary system memory and an optional, external backup cache (Bcache). The memory controller can directly control up to 512 MB of memory, organized into four banks of dynamic random-access memory (DRAM) or video random-access memory (VRAM), or both, and 64 KB to 2 MB of optional, external Bcache. Dual-bank (also called split-bank) single inline memory modules (SIMMs) are directly supported.

Each bank of memory can contain up to 16M addressable 64-bit locations. The base address of each bank can be located anywhere within the lower half of physical memory on naturally aligned boundaries. Each memory bank and the Bcache can be individually configured using the bank configuration, mask, timing registers, and the cache register. (See Section 5.6 for more information about the memory controller registers.)

Figure 5–1 shows a typical memory system. Note that all banks of memory need not be present in a system.

Figure 5–1 Typical Memory System



Notes: Signal buffering that may be required due to loading is not shown.

mem_rasx = mem_rasa or mem_rasb

▭ = optional components

The memory controller performs the following functions:

- Performs read, write, longword write, byte-write (with external logic), and write-per-bit operations to the DRAMs and VRAMs
- Controls memory refresh for the DRAMs and VRAMs

- Uses column-address strobe before row-address strobe (CAS-before-RAS) refreshing simultaneously on all banks
- Includes optimization for fast page-mode access
- Reads and writes the Bcache
- Supports optional error-correction code (ECC) detection and correction
- Supports graphics operations, including stipple and write-per-bit using a plane mask register

5.1.1 Memory Partitioning

The four banks of memory can span up to 512 MB of DRAM. Each bank is 64 bits wide (excluding ECC bits) and can be any of the following sizes: 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, or 128 MB. VRAM bank sizes of 1 MB, 2 MB, or 4 MB are supported. When VRAM is used in one of the banks, the maximum memory configuration is 384 MB of DRAM and 4 MB of VRAM.

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

5.1.2 Programmable Configuration

The memory controller registers (Section 5.6) control the programmable configuration parameters, including bank size, ECC, cache, external memory timing, and several error-checking parameters.

5.1.3 Error Correction and Detection

An 8-bit ECC can be used on all memory cycles to banks that support ECC to correct single-bit errors, detect double-bit errors, and detect 4-bit nibble errors. Nibble error detection can be useful when using $n \times 4$ memory parts. The memory data bits must be wired as shown in Table 5–1 for nibble error detection to work correctly.

Note

In some cases, more than 2 bits are in error and are not a full nibble, but appear to be either a correctable error or not an error.

Table 5–1 shows the memory data bit grouping for nibble error detection.

Table 5–1 Nibble Correction Bit Grouping

D0	D1	D2	D3
D4	D5	D6	D7
D8	D9	D10	D11
D12	D13	D14	D15
D16	D17	D18	D19
D20	D21	D22	D23
D24	D25	D26	D27
D28	D29	D30	D31
D32	D33	D34	D35
D36	D37	D38	D39
D40	D41	D42	D43
D44	D45	D46	D47
D48	D49	D50	D51
D52	D53	D54	D55
D56	D57	D58	E0
D59	D60	E1	E2
D61	D62	D63	E6
E3	E4	E5	E7

Table 5–2 shows how the 8-bit ECC is derived. The ones (1's) in the table indicate the bits that are exclusive-ORed to produce the corresponding check bit.

Table 5–2 Error Correction Code

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

*E3 and E7 are stored active low.

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

5.1.4 Graphics Features

The memory controller can perform the following simple graphics operations:

- Dumb frame buffer operation
- Transparent stipple
- Write-per-bit plane masking
- Byte write operations (with external gating)
- Full and split VRAM shift-register loads

The CPU must generate the address for all graphics operations. These operations are programmable using the graphics control register (Section 5.6) and are used only on writes to addresses in the graphics address region (a noncacheable region—see Table 2–6).

5.2 Backup Cache

The memory controller supports a 64-KB, 128-KB, 256-KB, 512-KB, 1-MB, or 2-MB Bcache. The Bcache is a direct-mapped, write-back cache with a quadword (64-bit) block size. Data ECC and tag parity protection are optional. The tag is compared onchip, and all control strobes for the cache static RAMs (SRAMs) are generated onchip.

The tag stores the upper part of an address cached at a given location. Which bits of the address are used is a function of the cache size (Section 5.6.8). The tag also stores a dirty bit, which indicates that the data stored at that cache location has been written more recently than the data stored at the corresponding memory location.

The memory controller uses the Bcache so as to minimize overall memory latency. The Bcache operates according to the following rules (see Figure 5–2):

- If the Bcache is enabled and the address is in cacheable space (bits <33:29> = 0), the memory controller checks the cache for a hit.
  - If a read operation hits, the cache data is used and the DRAM is not read.
  - If a read operation misses, the DRAM is read. A cache block is allocated only if the read is from the CPU.

    - If a write operation hits, the cache is updated.
    - If a write operation misses, a cache block is allocated only if the write operation is from the CPU.

  When a cache block is to be allocated (on either a read or write miss) and the dirty bit is set in the block's tag, the current contents of the block (called a dirty victim block) are written to DRAM before the block is allocated.

- If the Bcache is not enabled or the address is not in cacheable space, the memory controller omits the cache access and immediately accesses the DRAM.

The Bcache lookup is independent of page-mode operation; that is, the Bcache is looked into prior to page-mode and non-page-mode DRAM cycles (Section 5.3.2).

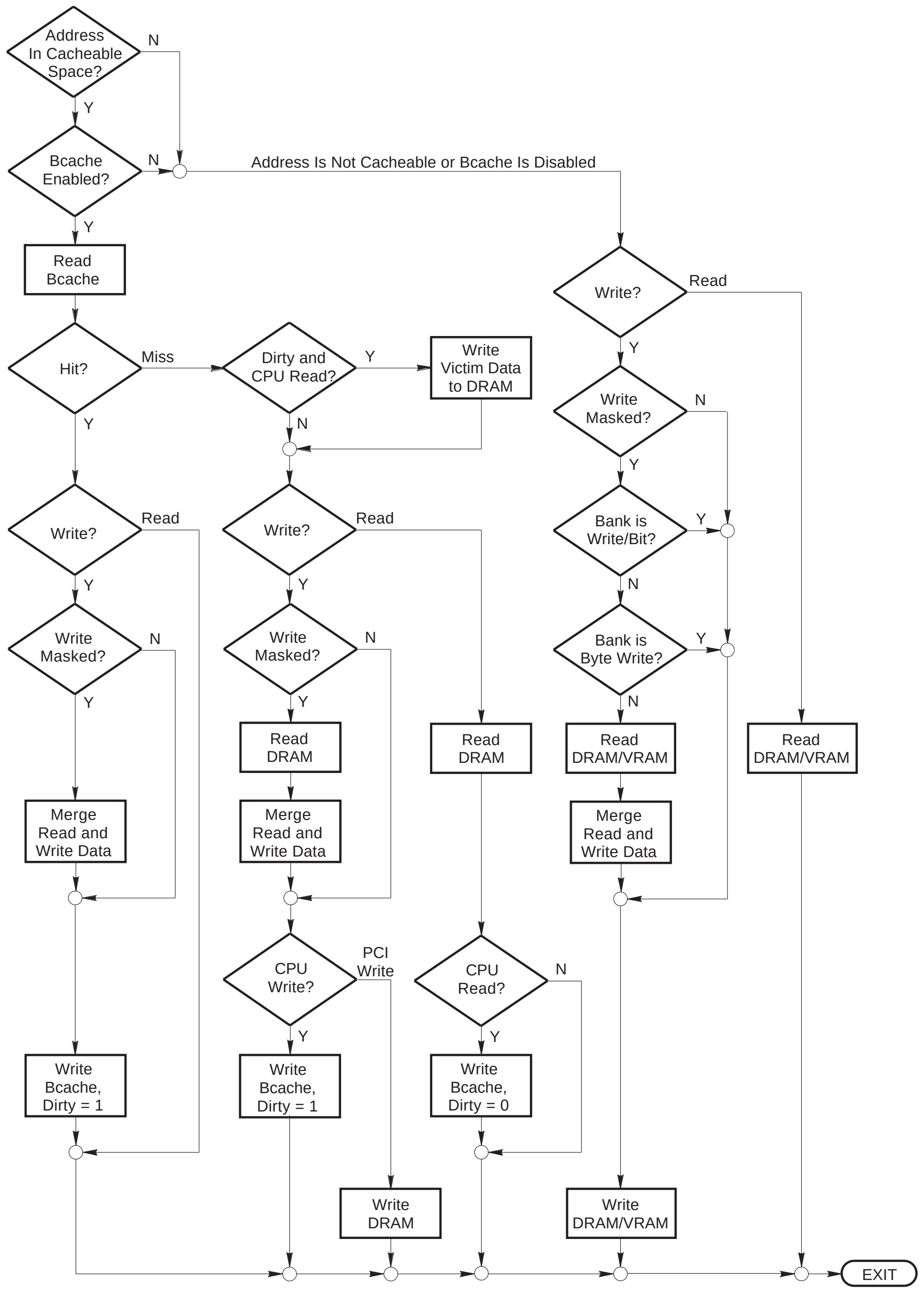
5.2.1 Backup Cache Initialization

Because the Bcache does not have valid bits, it must be initialized by software. The Bcache is initialized by sweeping the entire cache storage space with write operations. This is done by enabling the Bcache, clearing the parity and ECC enable bits in the cache register (CAR, Section 5.6.8), and then writing enough memory locations to sweep the cache's address range. To avoid nonexistent memory errors, the memory bank registers should be set up to span the largest possible cache address range. After the sweep, the Bcache will have valid data and correct tag, parity, and ECC information loaded. The parity and ECC enable bits can then be set.

5.3 Memory Controller Operation

The memory controller decodes all addresses from the CPU and PCI direct memory access (DMA) devices. If the address is in memory space (see Table 2–6), the memory controller follows the sequence shown in Figure 5–2.

Figure 5–2 Memory Controller Operation



5.3.1 Masked Write Operations

A masked write operation is a write operation that modifies less than the full quadword of data at an address. Masked write operations are the result of:

- Longword store instructions that did not get merged in the CPU write buffer
- PCI DMA write operations, which can specify write-per-byte
- Write operations to graphics address space

Because ECCs cover the full quadword, masked writes operation to memory that are protected by ECC must be handled by read-modify-write operations. In a read-modify-write operation, the memory controller reads the memory data, merges it internally with the write data, and then writes the result back to memory.

Banks of DRAM or VRAM that are not protected by ECC can have write-per-bit memories (normally VRAMs have this) or can be made byte-writable with a minimum of external logic. Both options are enabled by bits in the bank configuration registers (BCR3–BCR0, Section 5.6.2).

If the address is in graphics space and write-per-bit is enabled in the addressed bank's BCR , the memory controller drives the write-per-bit mask on the data lines during the row address period. The write data ECC is also driven during this operation, although it is not used by memory.

If byte write is enabled in the addressed bank's BCR, the memory controller drives the byte-write mask on the **mem_ecc<7:0>** pins during the column address period. External logic is required to gate CAS at the DRAMs and VRAMs. Some masked write operations to byte-writable banks are done as read-modify-write operations. For a masked write operation to skip the read portion, either of the following conditions must be met:

- The address is in noncacheable space.
- The address is in graphics space and the byte-write enable (BWE) bit is set in the video and graphics control register (VGR, Section 5.6.9).

Write-per-bit mode differs from byte-write mode in that write-per-bit mode can modify arbitrary bits, while byte-write mode can modify memory only on a byte-resolution basis. This means that byte-write mode is normally used with pixel depths that are integral numbers of bytes. Write-per-bit and byte-write modes can be used simultaneously.

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

5.3.2 Page Mode

Page-mode cycles are cycles that do not cycle RAS to latch a new row address into the DRAMs. Page-mode cycles can be any combination of read and write operations. The memory controller does a page-mode cycle when RAS has been asserted for a bank and a read or write request is made with the page the same as when RAS was asserted. *Same page* is a function of DRAM column size and is determined by comparing address bits as shown in Table 5–6.

5.3.3 Register Access

The memory controller contains the registers described in Section 5.6. The CPU can read and write the memory controller registers but I/O devices cannot. Table 5–3 lists the registers and their addresses.

5.4 Memory Error Conditions

The memory controller checks and logs memory error conditions. Each possible memory error is associated with a bit in the error status register (ESR, Section 5.6.6). When an error occurs, the associated ESR bit is set and the address of the location with the error is stored in the error address register (EAR, Section 5.6.7).

An interrupt is sent to the CPU when an error condition is detected. Software must clear any error condition bits and return the memory subsystem to correct operation. If another error occurs before the CPU clears the first error, a multiple error bit is set, but no error-type information is saved.

Sections 5.4.1 through 5.4.5 describe the errors and the action taken for each type of error.

5.4.1 Correctable Read Error

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

When a correctable read error is detected, the correctable error (CEE) bit is set in the ESR and an interrupt is sent to the CPU to indicate that an error was detected. Software must write the correct data to the address that had the single bit error. This can be done with a LDQ_L and STQ_C instruction sequence.

5.4.2 Uncorrectable Read Error

An uncorrectable read error occurs when the ECC logic detects an error in more than one data bit. The initiator of the read operation (either the CPU or a PCI device) is informed that an error occurred. (See Section 5.1.3 for more information about ECC.)

When an uncorrectable read error is detected, the memory uncorrectable error (UEE) bit is set in the ESR and an interrupt is sent to the CPU to indicate that an error was detected. Software must reestablish the validity of the faulting memory location, if possible.

5.4.3 Uncorrectable Write Error

An uncorrectable write error can occur during a read-modify-write cycle of a masked write operation. The error is detected during the read part of the operation. Because the write data cannot be correctly merged with the (uncorrectable) read data, the memory write operation is aborted.

Aborting the memory write operation preserves the uncorrectable read data in memory. A subsequent read of the location will return an uncorrectable read status. The programmer must decide whether a failed write operation is a fatal problem.

When this error is detected, the UEE bit is set in the ESR and an interrupt is sent to the CPU to indicate that an error was detected. Software must reestablish the validity of the faulting memory location, if possible.

5.4.4 Bcache Tag Parity Error

Every time the memory controller accesses the Bcache, the tag, dirty bit, and tag parity bit are checked for odd parity. The parity is calculated on the **bc_tag<7:0>** and **bc_idx_tag<4:0>** signals. If tag parity is enabled and the stored parity is incorrect, a tag parity error occurs.

If a tag parity error occurs on a read operation, the initiator of the read operation (either the CPU or a PCI device) is informed that an error occurred.

When a Bcache tag parity error is detected, the cache tag parity error (CTE) bit is set in the ESR and an interrupt is sent to the CPU to indicate that an error was detected. Software must reestablish Bcache coherency, if possible.

5.4.5 Nonexistent Memory Address Error

A nonexistent memory address error occurs when a DRAM read or write operation is attempted at an address that is not programmed in the bank address mask registers (BMR3–BMR0, Section 5.6.3).

The memory controller allocates the Bcache on write operations. A write operation to a nonexistent address sets the NXM bit and does not reallocate the cache. A nonexistent dirty address in the Bcache (possibly present at chip reset) can be evicted by setting the BMRs to expand the space that the Bcache can address. This will allow the victim write operation to occur without error and allow the read or write operation that initiated the victim write operation to allocate the cache.

If the nonexistent memory address error occurs on a read operation, the initiator of the read operation (either the CPU or a PCI device) is informed that an error occurred.

When a nonexistent memory address error is detected, the NXM bit is set in the ESR and an interrupt is sent to the CPU to indicate that an error was detected.

5.5 Graphics Operations

The simple graphics functions implemented in the memory controller hardware improve frame buffer performance.

The graphics functions are used on memory write operations when address bits <33:29> select an address in graphics space—the graphics functions are not enabled in any other address region (Table 2–6). The video and graphics control register (VGR, Section 5.6.9) specifies which graphics function is performed. Read operations to the graphics space ignore the VGR settings—graphics operations are done only on write operations.

The memory controller supports both full and split VRAM shift-register loads. External monitor timing logic is required to signal the memory controller when VRAM shift-register loads need to be done. The video display pointer (an internal linear-address generator, Section 5.6.9.1) keeps track of the video refresh address.

5.5.1 Simple Frame Buffer Mode

In simple frame buffer mode, the memory controller can function as an interface to a dumb frame buffer. The plane mask register (PMR, Section 5.6.10) masks the bits written in memory, using either write-per-bit, byte-write, or read-modify-write operations. A bit is written only if the corresponding PMR bit value is one.

In byte-write mode, the instruction (STL or STQ) and longword address specify the byte mask that is driven out on the **mem_ecc<7:0>** pins for each byte.

5.5.2 Transparent Stipple Mode

In transparent stipple mode, CPU write cycles can conditionally substitute the foreground data value for the frame buffer data.

The CPU write address specifies the address to be written. The write data and PMR contents combine to mask some of the bits written in memory. The contents of the foreground register (FOR, Section 5.6.11) is the data that is masked and written to memory, by using either byte-write, write-per-bit, or read-modify-write operations. A FOR bit is written to memory only if the corresponding write data bit and PMR bit values are both one.

In byte-write mode, the instruction (STL or STQ), longword address, and least-significant bit (LSB) of the corresponding byte-write data specify the byte mask that is driven out on the **mem_ecc<7:0>** lines for each byte.

5.6 Memory Controller Registers

Sections 5.6.1 through 5.6.11 describe the memory controller registers. All the registers are 64 bits wide and are aligned on quadword addresses. (In most of the registers, bits <63:32> are not used.)

Before memory operations can be initiated, the registers for at least one bank must be initialized with the correct memory parameters.

5.6.1 Programmable Timing Parameters

Some of the bank timing, global timing, and cache register fields are programmed to the desired number of delay cycles minus one, two, or three. For example, the following equation means that the interval T_{yz} (measured in integral clock cycles) is obtained by setting BTR<n:m> to $T_{yz} - 1$:

$$T_{yz} = BTR < n : m > +1$$

(The relation between the field value and the measured interval is given at the end of each timing field description in Sections 5.6.4, 5.6.5, and 5.6.8.)

Table 5–3 lists the registers and their addresses.

Note

Table 5–3 shows the preferred address to be used when accessing memory controller registers. While only bits <7:3> within the address range are decoded, making each register accessible at multiple addresses, addresses other than those listed are reserved by Digital.

Table 5–3 Memory Controller Registers

Register	Mnemonic	Address*
Bank configuration 0	BCR0	1 2000 0000
Bank configuration 1	BCR1	1 2000 0008
Bank configuration 2	BCR2	1 2000 0010
Bank configuration 3	BCR3	1 2000 0018
Bank address mask 0	BMR0	1 2000 0020
Bank address mask 1	BMR1	1 2000 0028
Bank address mask 2	BMR2	1 2000 0030
Bank address mask 3	BMR3	1 2000 0038
Bank timing 0	BTR0	1 2000 0040
Bank timing 1	BTR1	1 2000 0048
Bank timing 2	BTR2	1 2000 0050
Bank timing 3	BTR3	1 2000 0058
Global timing	GTR	1 2000 0060
Error status	ESR	1 2000 0068
Error address	EAR	1 2000 0070
Cache	CAR	1 2000 0078
Video and graphics control	VGR	1 2000 0080
Plane mask	PLM	1 2000 0088
Foreground	FOR	1 2000 0090

*Hexadecimal

Note

The abbreviations in the Type column of the register field description tables indicate field access behavior. The abbreviations are defined in the Conventions section of the Preface.

5.6.2 Bank Configuration Registers

Each memory bank has one bank configuration register (BCR). These registers (BCR3–BCR0) specify the following banks:

- Base address
- Error mode
- Write mode
- Row address manipulation

The BCRs are write-only registers. At reset, the base address valid (BAV) bit is cleared and all other bits are UNDEFINED.

Figure 5–3 shows the BCR format, and Table 5–4 describes its fields.

Figure 5–3 BCR3–BCR0 Format

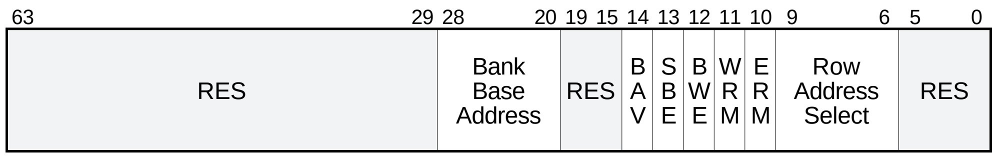


Table 5–4 BCR3–BCR0 Field Description

Bits	Field	Type	Description
63:29	RES	RAX/IGN	Reserved.
28:20	Bank Base Address	WO	This field specifies the base (starting) physical address of the associated bank of memory. Each bank of memory must begin on a naturally aligned boundary. (Naturally aligned means that for a bank with 2^n addresses, the *n* least significant bits must be zero.) The bank base address is compared to physical address bits <28:20> and masked by the bank address mask register (BMR3–BMR0, Section 5.6.3).
19:15	RES	RAX/IGN	Reserved.

(continued on next page)

Table 5–4 (Cont.) BCR3–BCR0 Field Description

Bits	Field	Type	Description
14	BAV	WO	Base address valid—This bit indicates whether the base address is valid. To enable addressing of the associated memory bank, this bit should be set when the base address is written. This bit is cleared at reset. Before setting this bit after reset, valid information must be written in the corresponding bank address mask register (BMR3–BMR0, Section 5.6.3).
13	SBE	WO	Split bank enable—When set, this bit enables dual (split) bank operation for the bank. This controls RAS selection as shown in Table 5–5, using the address bit shown in Table 5–10.
12	BWE	WO	Byte write-enable—When set, this bit enables external byte masked write operations for graphics operations. The byte mask is determined by the graphics mode (Section 5.6.9), foreground register (FOR, Section 5.6.11), and plane mask register (PMR, Section 5.6.10). The byte mask is driven on the **mem_ecc<7:0>** lines and external logic is required to gate the RAM write signals for each byte. **Note:** To use byte masking, the ERM bit (<10>) must be cleared to disable ECC for the bank.
11	WRM	WO	Write mode—When set, this bit enables write-per-bit capability for the bank. **Note:** To use write-per-bit, the ERM bit (<10>) must be cleared to disable ECC for the bank.
10	ERM	WO	Error mode—When set, this bit enables ECC checking and generation for each access to this bank. **Note:** Memories that use byte-write (<12>) or write-per-bit (<11>) mode must not be programmed to use ECC.
9:6	Row Address Select	WO	These bits specify how the memory address is multiplexed for this bank of memory. Table 5–6 lists the assignment of these bits for the number of row and column addresses, and Table 5–7 shows how the address bits are multiplexed. Column addresses always consist of address bits <14:3>.

(continued on next page)

Table 5–4 (Cont.) BCR3–BCR0 Field Description

Bits	Field	Type	Description
5:0	RES	RAX/IGN	Reserved.

Table 5–5 shows how RAS is selected according to the split-bank enable (SBE) bit in the BCR and the address bit listed in Table 5–10.

Table 5–5 RAS Selection

Dual-Bank Enable	Address Bit*	mem_rasa_l<3:0>	mem_rasb_l<3:0>
0	1 or 0	Y	N
1	0	Y	N
1	1	N	Y

*From Table 5–10.

Table 5–6 shows how the BCR row address select field is encoded and interpreted.

Table 5–6 Row Address Select Encodings

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

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

Table 5–7 shows how the row and column address bits are multiplexed on the **mem_addr<11:0>** pins.

Table 5–7 Row Address Bit Assignments

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

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

5.6.3 Bank Address Mask Registers

Each memory bank has one bank address mask register (BMR). These registers (BMR3–BMR0) specify the base address bits that should *not* be checked for an address match.

Note

Because the physical address compare does not include bits <32:29>, memory can be addressed at multiple addresses. Software must ensure that this does not cause the system to malfunction.

The bank base address mask is also used with the split-bank enable (SBE) bit in the bank configuration register (BCR, Section 5.6.2) to specify which address bit is used to assert **mem_rasa_l*n*** or **mem_rasb_l*n*** for each bank, as shown in Table 5–10.

These registers are UNDEFINED at reset. Valid information must be written to these registers before setting the valid bit in the corresponding BCR.

Figure 5–4 shows the BMR format, and Table 5–8 describes its fields.

Figure 5–4 BMR3–BMR0 Format

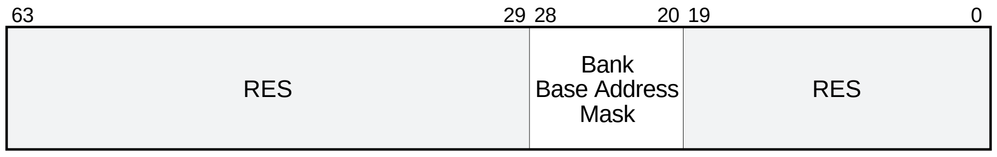


Table 5–8 BMR3–BMR0 Field Description

Bits	Field	Type	Description
63:29	RES	IGN	Reserved.
28:20	Bank Base Address Mask	WO	This field specifies the mask to be used when matching the memory address to the configured memory address range. Base address bits that correspond to BMR bits with a value of one are masked (that is, not checked). The mask corresponds to physical address bits <28:20> (Table 5–9).
19:0	RES	IGN	Reserved.

Table 5–9 shows how the bank address mask field in the BMR is set according to memory bank size.

Table 5–9 Bank Size Mask Setting

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

Table 5–10 shows which address bit selects RAS according to the value of the bank address mask field in the BMR.

Table 5–10 Dual Bank Address Select

Bank Base Address Mask BMR <28:20>	Address Bit That Selects* mem_rasa_l*n* or mem_rasb_l*n*
001111111	26
000111111	25
000011111	24
000001111	23
000000111	22
000000011	21

*Address bit value 0 selects **mem_rasa_l*n*** and value 1 selects **mem_rasb_l*n*** (*n* = 3, 2, 1, or 0). See Table 5–5.

5.6.4 Bank Timing Registers

Each memory bank has one bank timing register (BTR). These registers (BTR3–BTR0) specify the timing parameters, in cycles, independently for each bank. Most of the BTR fields are programmed to the desired number of delay cycles minus one. The relation between the field value and the measured interval is given at the end of each field description. See Section 5.6.1 for more information.

Table 5–22 and the timing diagrams in Section 5.7 identify the timing parameters.

The BTRs are write-only registers. All bits are UNDEFINED at reset.

Note

If bank 3 is not used, the read-to-write tristate field (<23:21>) in BTR3 must be initialized to zero.

Figure 5–5 shows the BTR format, and Table 5–11 describes its fields.

Figure 5–5 BTR3–BTR0 Format

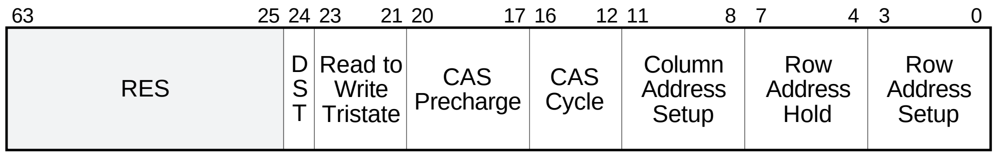


Table 5–11 BTR3–BTR0 Field Description

Bits	Field	Type	Description
63:25	RES	RAX/IGN	Reserved.
24	DST	WO	Data setup—This bit specifies the minimum number of data setup cycles prior to asserting the **mem_cas_l** signal on write operations as follows: 0 = one cycle, 1 = two cycles. The DRAM timing parameter is t_{DS}.
23:21	Read-to-Write Tristate	WO	This field specifies the number of cycles required to tristate the memory data bus when switching from a read to a write operation. The DRAM timing parameter is t_{OFF}. $t_{OFF}/cycletime = BTR < 23:21 > +1$
20:17	CAS Precharge	WO	This field specifies the number of cycles that the **mem_cas_l** signal is deasserted between page-mode cycles. It also represents the setup time before the **mem_cas_l** signal asserts the column address, write data (during memory write operations), and write-enable. The timing diagram parameter is T_{45}, and the DRAM parameter is t_{CP}. $T_{45} = BTR < 20:17 > +1$
16:12	CAS Cycle	WO	This field specifies the number of cycles that the **mem_cas_l** signal is asserted. The timing diagram parameter is T_{34}, and the DRAM parameter is t_{CAC}. $T_{34} = BTR < 16:12 > +1$

(continued on next page)

Table 5–11 (Cont.) BTR3–BTR0 Field Description

Bits	Field	Type	Description
11:8	Column Address Setup	WO	This field specifies the number of cycles between the time that column address, write data (during memory write operations), and write-enable are driven and the time that the **mem_cas_l** signal is asserted. The timing diagram parameter is T_{23}, and the DRAM parameter is t_{ASC}. $T_{23} = BTR < 11:8 >$
7:4	Row Address Hold	WO	This field specifies the number of cycles between the time that the **mem_ras_l** signal is asserted and the time that the **mem_addr<11:0>** pins switch from row address to column address. The timing diagram parameter is T_{12}, and the DRAM parameter is t_{RAH}. **Note:** The term **mem_ras_l** is equivalent to the **mem_rasa_l<3:0>** or **mem_rasb_l<3:0>** signal for the selected bank (Section 5.8.5). $T_{12} = BTR < 7:4 > +1$
3:0	Row Address Setup	WO	This field specifies the number of cycles between the time that the row address is driven on the **mem_addr<11:0>** pins and the time that the **mem_ras_l** signal is asserted. The timing diagram parameter is T_{01}, and the DRAM parameter is t_{RAS}. $T_{01} = BTR < 3:0 > +1$

5.6.5 Global Timing Register

The global timing register (GTR) specifies timing parameters for all four memory banks. Most of the GTR fields are programmed to the desired number of delay cycles minus one or two. The relation between the field value and the measured interval is given at the end of each field description. (See Section 5.6.1 for more information.)

Table 5–22 and the timing diagrams in Section 5.7 identify the timing parameters.

Figure 5–6 shows the GTR format, and Table 5–12 describes its fields.

Figure 5–6 GTR Format

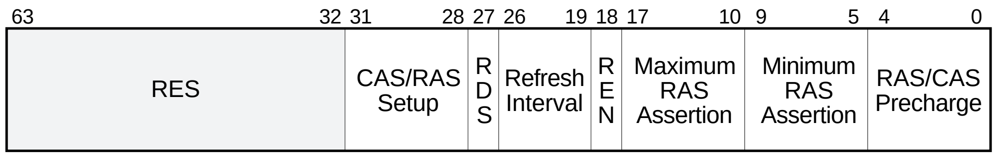


Table 5–12 GTR Field Description

Bits	Field	Type	Description
63:32	RES	RAX/IGN	Reserved.
31:28	CAS/RAS Setup	RW	This field specifies the number of cycles between the assertion of the **mem_cas_l** signal and the assertion of the **mem_ras_l** signal on refresh cycles. The number of cycles is this value plus two. $T_{CAS-RAS-SETUP} = GTR < 31 : 28 > +2$
27	RDS	RW	Refresh divide select—This bit specifies the prescale divide for the refresh interval as follows: 0 = divide by 64, 1 = divide by 512.*
26:19	Refresh Interval	RW	This field specifies the number of cycles (prescaled by 64 or 512, as specified by the RDS bit) between requests to perform a refresh cycle. The request is generated internally, and the refresh occurs at the next available free cycle; that is, after the completion of any memory cycles in progress. The refresh counter reloads this value when it requests the refresh cycle. Nothing is added to this value; therefore, a value of zero will continually refresh the memory.* $T_{REFINT} = GTR < 26 : 19 >$
18	REN	RW	Refresh enable—When set, this bit enables refresh operations for all banks. This bit is cleared at reset. When this bit is clear, the refresh counter is initialized to the value in the refresh interval field (<26:19>).*

*The refresh interval and refresh divide select fields are not initialized at reset. When they are changed, the REN bit (<18>) should be deasserted to allow them to load.

(continued on next page)

Table 5–12 (Cont.) GTR Field Description

Bits	Field	Type	Description
17:10	Maximum RAS Assertion	RW	This field specifies the maximum number of cycles (prescaled by 128) that the **mem_ras_l** signal can be asserted. The **mem_ras_l** signal stays asserted for page-mode cycles, which can be long, depending on program and I/O behavior. At the assertion of the **mem_ras_l** signal, this value is loaded into a down counter and decremented once every 128 cycles. If the count reaches zero while the **mem_ras_l** signal remains asserted, the memory controller completes the current memory cycle and then deasserts the **mem_ras_l** signal. The DRAM parameter is t_{RAS}. $t_{RAS(max)} = GTR < 17:10 > +1$
9:5	Minimum RAS Assertion	RW	This field specifies the number of cycles that the **mem_ras_l** signal is asserted during a refresh cycle. The DRAM parameter is t_{RAS}. $t_{RAS(min)} = GTR < 9:5 >$
4:0	RAS/CAS Precharge	RW	This field specifies the number of cycles to precharge RAS and CAS at the end of a memory cycle. When the precharge time has elapsed, a new memory cycle can begin. The timing diagram parameter is $T_{RASprech}$, and the DRAM parameter is t_{RP}. $T_{RASprech} = GTR < 4:0 > +1$

5.6.6 Error Status Register

The error status register (ESR) controls the error detection and correction functions of the memory controller and holds the error status when an error occurs.

The ECC bits are read-only and the write wrong ECC (WEC) bits are read and write. The error status flags (CEE, UEE, CTE, MSE, MHE) can be read and are cleared by writing a one to them; writing a zero has no effect. Reset has no effect on this register.

Figure 5–7 shows the ESR format, and Table 5–13 describes its fields. The 21066A implements an additional bit. For a description, refer to Section A.6.4.

Figure 5–7 ESR Format

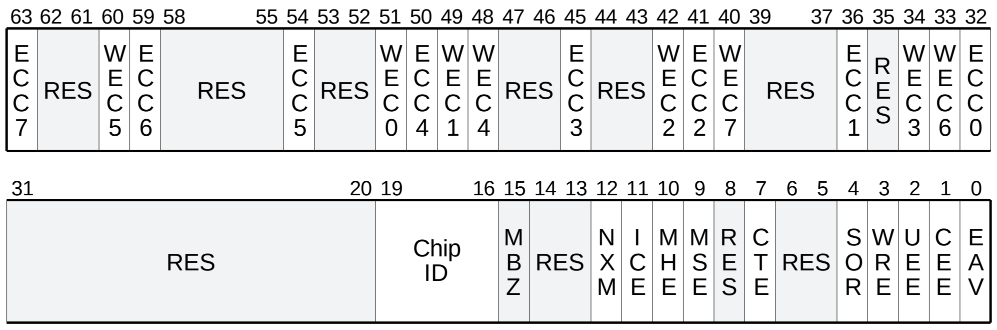


Table 5–13 ESR Field Description

Bits	Field	Type	Description
Error Correction Code Bits			
The following bits store the ECC read from the **mem_ecc<7:0>** pins. These bits are frozen when the error address valid (EAV) bit (<0>) is set.			
63	ECC7	RO	
59	ECC6	RO	
54	ECC5	RO	
50	ECC4	RO	
45	ECC3	RO	
41	ECC2	RO	
36	ECC1	RO	
32	ECC0	RO	
Write Wrong Error Correction Code Bits			
The following bits are used when testing the ECC detection and correction logic. When set to a one, the corresponding ECC check bit will be written incorrectly to memory or cache. For normal operation, the value written into these bits should be zero.			
60	WEC5	RW	
51	WEC0	RW	
49	WEC1	RW	
48	WEC4	RW	
42	WEC2	RW	
40	WEC7	RW	
34	WEC3	RW	
33	WEC6	RW	

(continued on next page)

Table 5–13 (Cont.) ESR Field Description

Bits	Field	Type	Description
Reserved Bits			
The following bits are reserved:			
62:61	RES	RAX/IGN	
58:55	RES	RAX/IGN	
53:52	RES	RAX/IGN	
47:46	RES	RAX/IGN	
44:43	RES	RAX/IGN	
39:37	RES	RAX/IGN	
35	RES	RAX/IGN	
31:20	RES	RAX/IGN	
14:13	RES	RAX/IGN	
8	RES	RAX/IGN	
6:5	RES	RAX/IGN	
Bits <19:16,15,12:9,7,4:0>			
19:16	Chip ID	RO	This field identifies the chip. Its value is 0000 for the 21066 and 0010 for the 21066A.
15	MBZ	RW	Must be zero.
12	NXM	R/W1C	Nonexistent memory—This bit is set when a DRAM read or write operation is attempted at an address that is not programmed in the bank address mask registers (BMR3–BMR0, Section 5.6.3).
11	ICE	RW	Ignore correctable errors—When set, this bit prevents the logging of correctable (soft) errors. On a soft error with ICE set: • The data is corrected before being used. • The CEE and EAV bits (<1:0>) are not set. • The error address register (EAR, Section 5.6.7) is not frozen.
10	MHE	R/W1C	Multiple hard errors—When set, this bit indicates that an additional hard (uncorrectable) error occurred after the EAV bit (<0>) was set. The secondary error status is not logged.

(continued on next page)

Table 5–13 (Cont.) ESR Field Description

Bits	Field	Type	Description
9	MSE	R/W1C	Multiple soft errors—When set, this bit indicates that an additional soft (correctable) error occurred after the EAV bit (<0>) was set. The secondary error status is not logged.
7	CTE	R/W1C	Cache tag error—When set, this bit indicates that a tag parity error was detected during a Bcache cycle.
4	SOR	RO	Error source—This bit indicates whether an error occurred while accessing Bcache or memory: 0 = cache, 1 = memory. It is updated on each cycle and frozen when the EAV bit (<0>) is set.
3	WRE	RO	Write error—This bit indicates whether an error occurred during a read or write access: 0 = read, 1 = write. It is updated on each cycle and frozen when the EAV bit (<0>) is set.
2	UEE	R/W1C	Uncorrectable error—This bit is set when an uncorrectable ECC error occurs during a read or read-modify-write operation.
1	CEE	R/W1C	Correctable error—This bit is set when a correctable ECC error occurs during a read or read-modify-write operation.
0	EAV	RO	Error address valid—When set, this bit indicates that the address in the error address register (EAR, Section 5.6.7) is a valid error address. This bit is the logical OR of bits <12,7,2:1>. While this bit is asserted, the memory controller asserts an interrupt to the CPU, and bits <12,7,4:1> are frozen, as is the EAR.

5.6.7 Error Address Register

The error address register (EAR) holds the address of a cycle that produced an error. The address is loaded in this register on a memory cycle. If an error occurs, this register is frozen and the error address valid (EAV) bit is set in the error status register (ESR, Section 5.6.6).

The EAR is UNDEFINED at reset.

Figure 5–8 shows the EAR format, and Table 5–14 describes its fields.

Figure 5–8 EAR Format

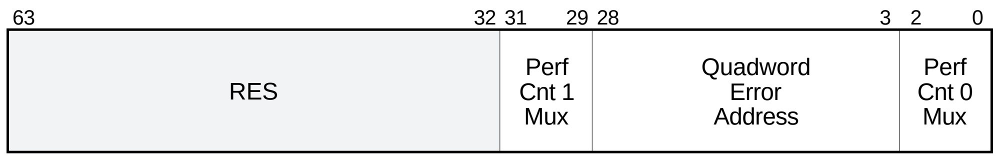


Table 5–14 EAR Field Description

Bits	Field	Type	Description
63:32	RES	RAX/IGN	Reserved.
31:29	Perf Cnt 1 Mux	WO	Performance counter 1 multiplexer—This field selects one of eight events to be counted for performance evaluation. The events are listed in Table 5–15.
28:3	Quadword Error Address	RO	This field identifies the quadword address (bits <28:3>) where an error occurred during a memory operation. The first address to have an error is logged in this register and held until the EAV bit of the error status register (ESR, Section 5.6.6) is cleared.
2:0	Perf Cnt 0 Mux	WO	Performance counter 0 multiplexer—This field selects one of eight events to be counted for performance evaluation. The events are listed in Table 5–15.

Table 5–15 lists the external interface events that can be monitored by the performance counters. See Section 4.1.14 for more information about the performance counters.

Table 5–15 Performance Counter Events

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

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

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

5.6.8 Cache Register

The cache register (CAR) is used to configure the optional, external Bcache. Some of the CAR fields are programmed to the desired number of delay cycles minus one or three. The relation between the field value and the measured interval is given at the end of each field description. (See Section 5.6.1 for more information.)

At reset, the Bcache enable (BCE) bit is set, the power-saving (PWR) bit is cleared, and all other bits are UNDEFINED.

Figure 5–9 shows the CAR format, and Table 5–16 describes its fields.

Figure 5–9 CAR Format

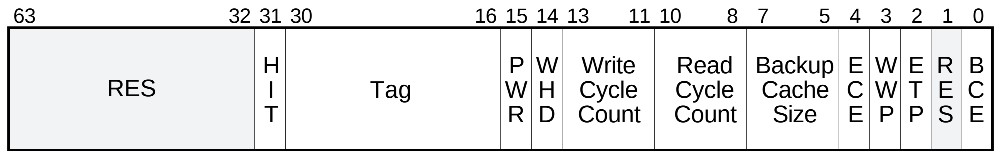


Table 5–16 CAR Field Description

Bits	Field	Type	Description
63:32	RES	RAX/IGN	Reserved.
31	HIT	RO	Bcache hit—This bit indicates the hit (1) or miss (0) status of the most recent Dstream Bcache tag comparison. This bit is for diagnostics.
30:16	Tag	RO	This field contains the tag value (parity, dirty, tag<7:0>, bc_idx_tag<4:0>) latched during the most recent Dstream Bcache lookup. This field is for diagnostics.
15	PWR	RW	Power saving—When set, this bit prevents the memory address lines from changing until a Bcache operation is detected. In addition, Bcache chip select is asserted only for Bcache operations. This adds one cycle of latency to the first read or write operation in a sequence of Bcache operations because the address lines are not speculatively updated. When this bit is cleared, the memory address lines are not held to a stable value in the absence of Bcache operations. This bit is cleared at reset.
14	WHD	RW	Write hold time—This bit specifies the number of cycles of hold time that are applied to cache write data after the **bc_we_l** signal is deasserted: 0 = one cycle, 1 = two cycles. This corresponds to timing diagram parameter T_{ch}.

(continued on next page)

Table 5–16 (Cont.) CAR Field Description

Bits	Field	Type	Description
13:11	Write Cycle Count	RW	This field specifies, in cycles, the Bcache write speed. The number of cycles is measured from the time that the chip drives data and asserts the **bc_we_l** signal to the time that the chip deasserts the **bc_we_l** signal. The number of cycles is the value of this field plus one. The write cycle count parameter is T_{wcc} and it corresponds to timing diagram parameter T_{cw}. $T_{wcc} = CAR < 13 : 11 > +1$
10:8	Read Cycle Count	RW	This field specifies, in cycles, the Bcache read speed. The number of cycles is measured from the time that the chip drives the address to the time that the chip latches cache data. The number of cycles is the value of this field plus three. The read cycle count parameter is T_{rcc} and it corresponds to timing diagram parameter T_{ca}. This field is cleared at reset. $T_{rcc} = CAR < 10 : 8 > +3$
7:5	Bcache Size	RW	This field specifies the size of the Bcache according to Table 5–17. The physical address is used as index and tag according to Figure 5–10. This field is cleared at reset.
4	ECE	RW	Enable Bcache ECC—When set, this bit enables error detection and correction on read operations that hit in the Bcache.
3	WWP	RW	Write wrong tag parity—When set, this bit causes the tag parity to be written incorrectly. This bit is for diagnostics.
2	ETP	RW	Enable tag parity check—When set, this bit enables parity checking for the Bcache.
1	RES	RAX/IGN	Reserved.
0	BCE	RW	Bcache enable—When set, this bit enables the Bcache. This bit is set at reset.

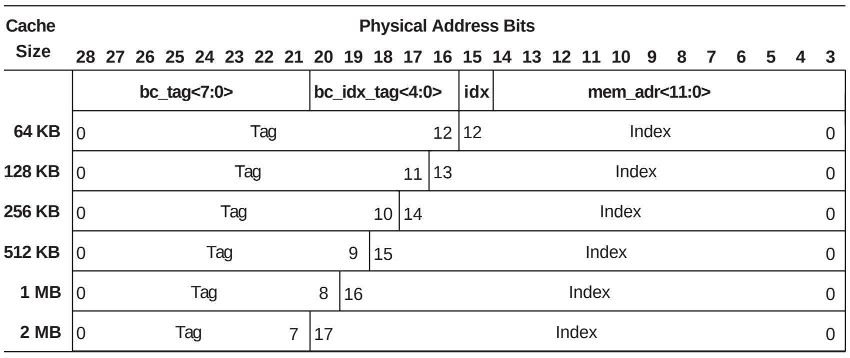
Table 5–17 shows the encodings for the Bcache size field in the CAR.

Table 5–17 Bcache Size Encodings

Code	Cache Size	Tag Address Bits	Index Address Bits
000	64 KB	28:16	15:3
001	128 KB	28:17	16:3
010	256 KB	28:18	17:3
011	512 KB	28:19	18:3
100	1 MB	28:20	19:3
101	2 MB	28:21	20:3
Unused codes are reserved.			

Figure 5–10 shows how the physical address is used as the Bcache index and tag according to cache size.

Figure 5–10 Bcache Address Partition



Notes for Figure 5–10:

- Address bits <2:0> are not used because cache references are quadword aligned.
- Address bits <14:3> are driven on the **mem_addr<11:0>** pins.
- Address bit 15 is driven on the **bc_index** pin.
- Address bits <20:16> are driven on the **bc_idx_tag<4:0>** pins and are used as index or tag according to Table 5–26.
- Address bits <28:21> are driven on the **bc_tag<7:0>** pins.

5.6.9 Video and Graphics Control Register

The video and graphics control register (VGR) is used to configure the video memory and to control graphics operations.

The VGR is a write-only register and all bits are UNDEFINED at reset.

Figure 5–11 shows the VGR format, and Table 5–18 describes its fields.

Figure 5–11 VGR Format

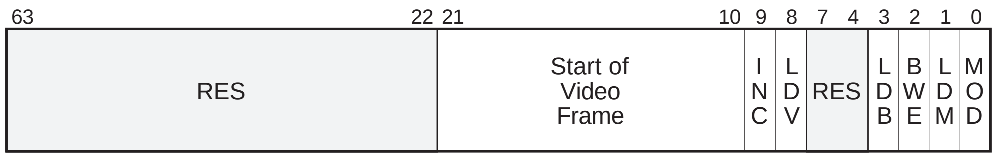


Table 5–18 VGR Field Description

Bits	Field	Type	Description
63:22	RES	IGN	Reserved.
21:10	Start of Video Frame	WO	This field specifies the frame location; that is, bits <21:10> of the base byte address of the video frame buffer in memory. Bits <9:0> of the address are implied to be 0; that is, the frame buffer is aligned to 1 KB. The video frame base address is loaded into the video display pointer (Section 5.6.9.1) when the external frame interrupt signal **vframe_l** is asserted.
9	INC	WO	Address increment—After a shift-register load operation, the video display pointer (Section 5.6.9.1) must be incremented to the address of the information for the next display. This value specifies the video display pointer bit to be incremented, according to Table 5–19. The bit location that is incremented depends on the position of the VRAM's column most-significant bit (MSB).
8	LDV	WO	Load video control—This bit determines whether the start of video frame and INC fields are loaded. Writing a one to this bit also loads the fields; writing a zero to this bit has no effect on the fields.
7:4	RES	IGN	Reserved.

(continued on next page)

Table 5–18 (Cont.) VGR Field Description

Bits	Field	Type	Description
3	LDB	WO	Load byte write-enable—This bit determines whether the BWE bit (<2>) is loaded. Writing a one to this bit also loads the BWE bit; writing a zero to this bit has no effect on the BWE bit.
2	BWE	WO	Byte write-enable—During graphics operations, this bit enables the memory controller to drive a byte mask on the **mem_ecc<7:0>** pins for write operations to banks that have byte write-enable set in their bank configuration register (BCR, Section 5.6.2).
1	LDM	WO	Load mode—This bit determines whether the MOD bit (<0>) is loaded. Writing a one to this bit also loads the MOD bit; writing a zero to this bit has no effect on the MOD bit.
0	MOD	WO	Graphics mode—This bit determines the type of memory operation on a write operation to graphics memory space: 0 = simple mode, 1 = transparent stipple mode (see Sections 5.5.1 and 5.5.2 for more information).

Table 5–19 shows how the VGR INC bit encoding selects the VRAM address bit to be incremented for shift-register loads.

Table 5–19 VRAM Address Increment Encodings

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

5.6.9.1 Video Display Pointer

The video display pointer is a 12-bit counter that supplies the VRAM address during video shift-register transfer cycles.

When the **vframe_l** signal is asserted, the video display pointer is:

1. Loaded from the VGR start of video frame field.
2. Used as the address for a full video shift-register transfer cycle.
3. Incremented twice according to the VGR INC field. (Either bit 10 or bit 11 is incremented.)

When the **vrefresh_l** signal is asserted, the video display pointer is:

1. Used as the address for a split video shift-register transfer cycle.
2. Incremented once according to the VGR INC field.

The address is multiplexed according to the VRAM bank size specified in the row address select field in the BCR (Section 5.6.2). Bits <9:0> of the video display pointer are zero.

5.6.10 Plane Mask Register

The plane mask (PLM) register determines which bits are written during a write or read-modify-write operation to the graphics address space.

Note

Software must write the same 32-bit value to <63:32> and <31:0>; otherwise, plane masked operations are UNPREDICTABLE.

The PLM register is a write-only register and all bits are UNDEFINED at reset.

Figure 5–12 shows the PLM register format, and Table 5–20 describes its fields.

Figure 5–12 PLM Register Format

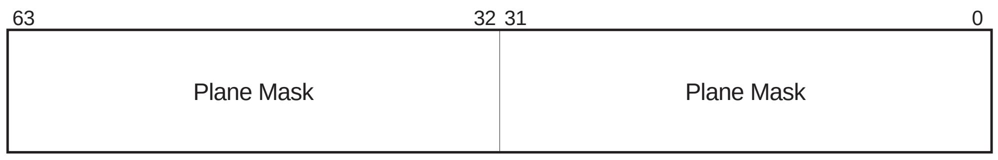


Table 5–20 PLM Register Field Description

Bits	Field	Type	Description
63:32	Plane Mask	WO	The bits in this field select which bits are going to be written during a write or read-modify-write operation.
31:0	Plane Mask	WO	The bits in this field select which bits are going to be written during a write or read-modify-write operation.

5.6.11 Foreground Register

The foreground register (FOR) defines the foreground color for transparent stipple operations.

Note

Software must replicate the foreground value across each half of the foreground register (that is, the bit 0 foreground value controls bits 0 and 32, and so on).

The FOR is a write-only register and all bits are UNDEFINED at reset.

Figure 5–13 shows the FOR format, and Table 5–21 describes its fields.

Figure 5–13 FOR Format

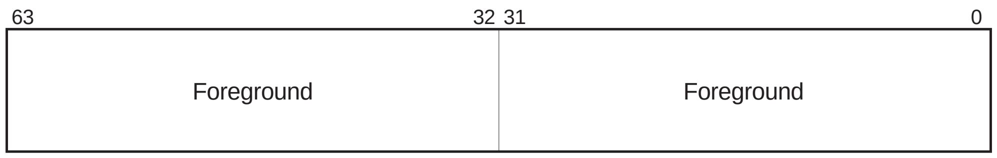


Table 5–21 FOR Field Description

Bits	Field	Type	Description
63:32	Foreground	WO	This field replicates the value in <31:0>.
31:0	Foreground	WO	This field specifies the foreground value for transparent stipple mode.

5.7 Memory Cycles

Table 5–22 describes the parameters used in the timing diagrams in Sections 5.7.2 through 5.7.11. These parameters are programmed into the BTR3–BTR0, GTR, and CAR as described in Section 5.6.1.

The timing diagrams assume all parameters are set to minimum values. Larger values can be accommodated by inserting the programmed number of cycles with the signals held constant at the levels they have reached at the end of the interval; that is, a longer time on T_{01} delays the occurrence of T_1—the signals maintain the value they changed to at T_0.

Table 5–22 describes the memory timing parameters.

Table 5–22 Memory Timing Parameters

Parameter	Events
T_{ca}	Bcache access time.
T_{cw}	Bcache write time.
T_0	Drive the row address on the **mem_addr<11:0>** pins. Drive the write mask on the **mem_data<63:0>** pins.
T_1	Assert the **mem_ras_l** signal for the selected bank. Tristate the **mem_data<63:0>** pins for read operations.
T_2	Drive the column address on the **mem_addr<11:0>** pins. If write cycle, drive the **mem_data<63:0>** pins—see Note 1.
T_3	Assert the **mem_cas_l** signal.
T_4	Deassert the **mem_cas_l** signal. If read cycle, latch the **mem_data<63:0>** pins. If page-mode cycle, drive the next column address on the **mem_addr<11:0>** pins. If page-mode write cycle, drive the **mem_data<63:0>** pins. If not page-mode cycle, deassert the **mem_ras_l** signal.
T_5	Assert the **mem_cas_l** signal.
T_6	Deassert the **mem_cas_l** signal. If read cycle, latch the **mem_data<63:0>** pins. Tristate the **mem_data<63:0>** pins. Deassert the **mem_ras_l** signal—see Note 2.
Parameter	**Description**
T_{minRAS}	The time that the **mem_ras_l** signal is asserted during a refresh cycle.
$T_{RASprech}$	RAS and CAS precharge time.

Notes for Table 5–22

1. When going from page-mode read to page-mode write operations, the memory controller will wait t_{OFF} (Section 5.6.4) before driving the **mem_data<63:0>** and **mem_ecc<7:0>** pins. If t_{OFF} is equal to or longer than T_{45} (CAS precharge), CAS precharge will be $T_{45} + 1$ cycles long and the **mem_data<63:0>** and **mem_ecc<7:0>** pins will be driven one cycle before the **mem_cas_l** signal is asserted.
2. The **mem_ras_l** signal will deassert at T_6 if there is another memory cycle pending that is not a page-mode cycle.

The memory cycle performed is a function of the type of memory operation required and the state of the memory controller at the time the operation is required. Table 5–23 describes the operations and the conditions that cause them.

Table 5–23 Memory Operations

Type of Operation	Originated By . . .
DRAM refresh	An internal refresh interval counter overflow (Section 5.6.5).
Full video shift-register transfer cycle	The assertion of the **vframe_l** pin (Section 5.8.19).
Split video shift-register transfer cycle	The assertion of the **vrefresh_l** pin (Section 5.8.20).
Read (cache or DRAM)	Any of the following: • An instruction fetch • A load instruction • A store longword instruction to a bank that does not support write-per-bit or byte write operations (read-modify-write) • A PCI device DMA read operation • A PCI device DMA write operation of less than a full quadword to a bank that does not support write-per-bit or byte write operations (read-modify-write)
Write (Cache or DRAM)	Any of the following: • A store • A PCI device DMA write operation
Victim write (DRAM)	All of the following: • A CPU read or write operation to cacheable space • A cache miss (tag does not match) • The cache block is dirty

(continued on next page)

Table 5–23 (Cont.) Memory Operations

Type of Operation	Originated By . . .
RAS precharge	Any of the following (one or more events may occur simultaneously): • The next operation will be a: – Read or write operation on a DRAM page that is different from the one for which the **mem_ras_l** signal was previously asserted. – Write-per-bit operation with a mask different from the one for which the **mem_ras_l** signal was previously asserted. – Refresh cycle. – Video shift-register load (either full or split). • The maximum RAS assertion period expired.

Table 5–24 defines the conditions that determine when read or write operations will access the Bcache and use page-mode, write-per-bit, and byte write operations.

Table 5–24 Memory Read and Write Options

Option	Used When . . .
Cacheable	The Bcache is enabled (Section 5.6.2) and the address is in cacheable space (Table 2–6).
Page-mode access	The address is on the same DRAM page as the previous cycle (Section 5.3.2) and the maximum RAS assertion time has not expired since the **mem_ras_l** signal is asserted (Section 5.6.5).
Write-per-bit	The addressed memory bank has write-per-bit enabled (Section 5.6.2) and the address is in graphics space (Table 2–6).
Byte write	The address is in graphics space (Table 2–6), the addressed memory bank has byte write enabled (Section 5.6.2), and the byte-write-enable (BWE) bit is set in the video and graphics control register (VGR, Section 5.6.9).

5.7.1 Idle Cycle

An idle cycle occurs when no other cycle type in Table 5–23 is required. If the PWR bit in the cache register is not set, the only memory controller signals that change during the idle cycle are the **mem_addr<11:0>**, **bc_index**, and **bc_idx_tag<4:0>** signals that are acting as the index. (This is done because the index is being driven out in anticipation of an internal cache miss, to make the Bcache latency as small as possible.) The **mem_ras_l** signal may be asserted or deasserted, depending on whether or not a page-mode cycle recently occurred. All other memory control signals are deasserted and the **mem_data<63:0>** signals are floating. The idle cycle is one cycle long, but multiple idle cycles can occur in succession. (See Section 5.6.8 for more information about what occurs when the PWR bit is set.)

5.7.2 RAS Precharge Timing

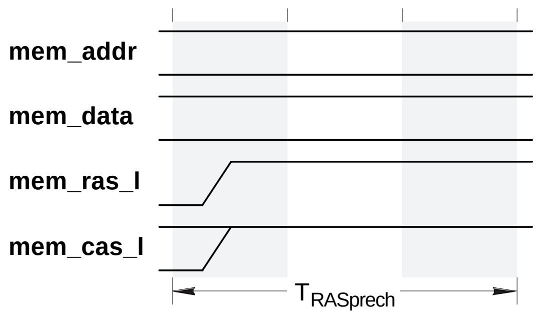
A RAS precharge cycle is performed when the **mem_ras_l** signal is asserted and one or more of the conditions in Table 5–23 is met.

The sequence is as follows:

1. The **mem_ras_l** signal is deasserted and the **mem_cas_l** signal, if it was asserted, is deasserted.
2. Wait for $T_{RASprech}$ cycles.

Figure 5–14 shows the timing for a RAS precharge cycle.

Figure 5–14 RAS Precharge Timing



5.7.3 DRAM Refresh Timing

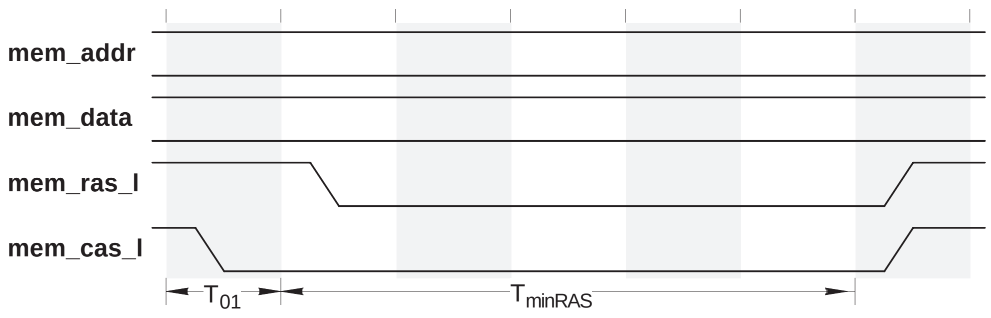
A DRAM refresh cycle is performed when the refresh interval in the global timing register (GTR, Section 5.6.5) expires. The memory controller coordinates refresh cycles with read and write operations, such that burst activities (for example, internal cache fills) are not disrupted by refresh cycles. This helps increase memory bandwidth and decrease latency.

The sequence is as follows:

1. The **mem_cas_l** signal is asserted.
2. After T_{01} cycles, all **mem_ras_l** signals are asserted.
3. After T_{minRAS} cycles, the **mem_ras_l** and **mem_cas_l** signals are deasserted.

Figure 5–15 shows the timing for a DRAM refresh cycle.

Figure 5–15 DRAM Refresh Timing



5.7.4 Full Video Shift-Register Transfer Timing

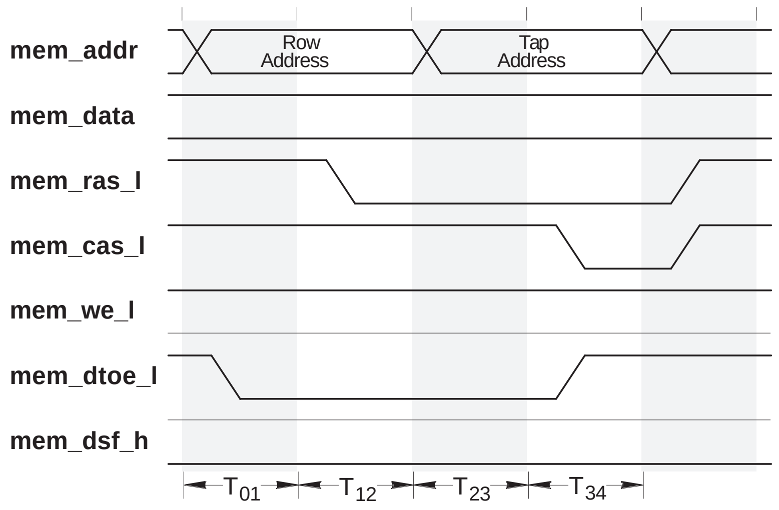
A full video shift-register transfer cycle is performed in response to the **vframe_l** signal being asserted (Section 5.8.19).

The sequence is as follows:

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

Figure 5–16 shows the timing for a full video shift-register transfer cycle.

Figure 5–16 Full Video Shift-Register Transfer Timing



5.7.5 Split Video Shift-Register Transfer Timing

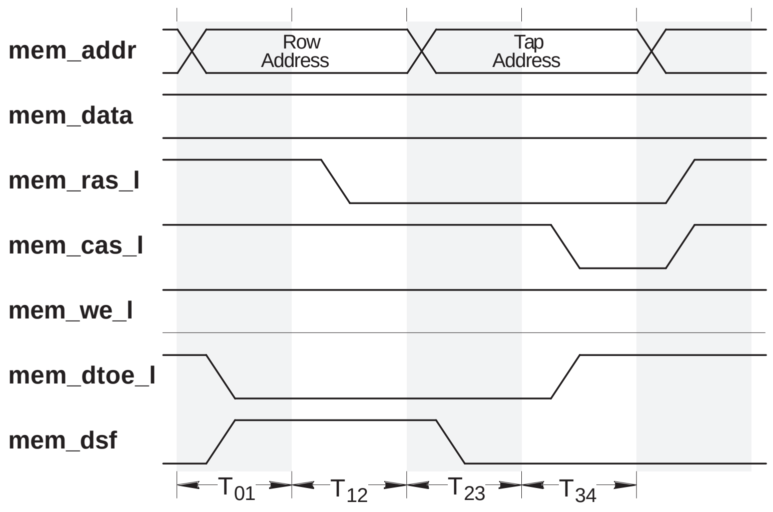
A split video shift-register transfer cycle is performed in response to the **vrefresh_l** signal being asserted (Section 5.8.20).

The sequence is as follows:

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

Figure 5–17 shows the timing for a split video shift-register transfer cycle.

Figure 5–17 Split Video Shift-Register Transfer Timing



5.7.6 Bcache Read Timing

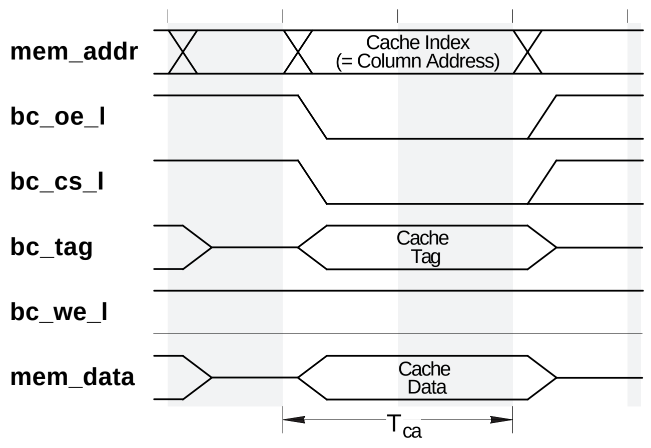
A Bcache read cycle can occur when a read operation is required from one of the operations in Table 5–23. The Bcache will be read first if the cacheable cycle conditions in Table 5–24 are met.

The sequence is as follows:

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

Figure 5–18 shows the timing for a Bcache read cycle.

Figure 5–18 Bcache Read Timing



5.7.7 Bcache Write Timing

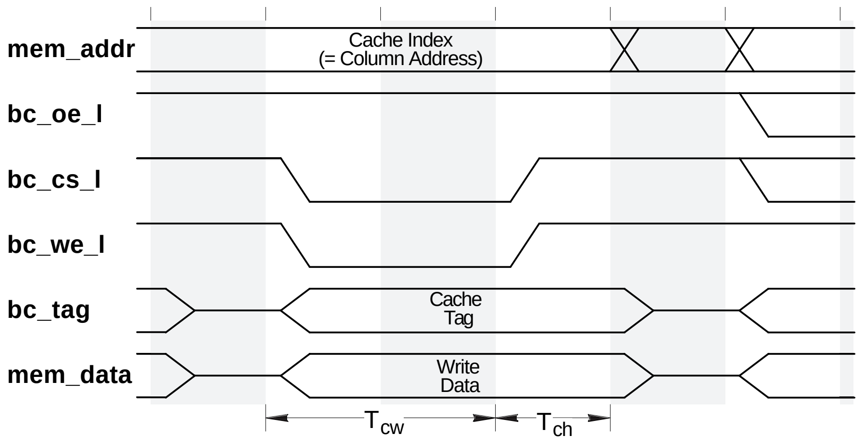
A Bcache write cycle can occur when a write operation is required from one of the operations in Table 5–23, or on a read operation that will allocate a block. On a cache write operation, the index is set up in advance from either a Bcache read or a DRAM read operation.

The sequence is as follows:

1. The **mem_data<63:0>**, **mem_ecc<7:0>**, **bc_tag_l**, **bc_dirty**, and **bc_parity** pins are driven. The **bc_oe_l** signal is deasserted. The **bc_we_l** and **bc_cs_l** signals are asserted. The **bc_dirty** signal is asserted for a write or deasserted for a read operation.
2. After T_{cw} cycles, the **bc_we_l** and **bc_cs_l** signals are deasserted.
3. After one or two cycles, depending on the value of the write hold time (WHD) bit in the cache register (CAR, Section 5.6.8), the **mem_addr<11:0>**, **bc_index**, and **bc_idx_tag<4:0>** signals are changed.

Figure 5–19 shows the timing for a Bcache write cycle.

Figure 5–19 Bcache Write Timing



5.7.8 Page-Mode Read Timing

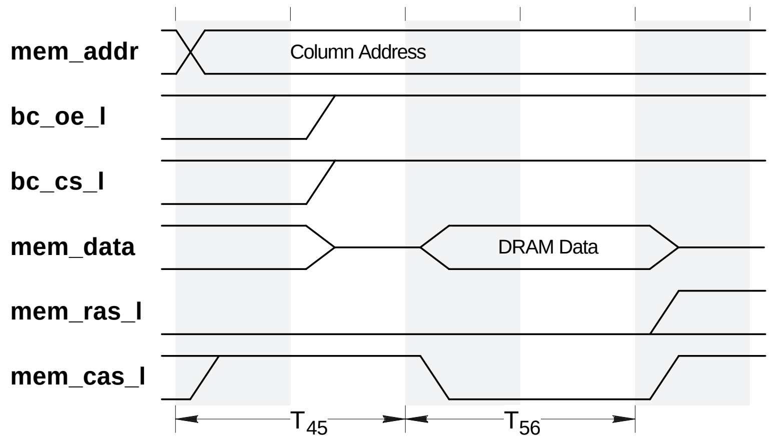
A page-mode read cycle occurs when a read operation is required from one of the operations in Table 5–23 and the page-mode conditions in Table 5–24 are met.

The sequence is as follows:

1. The column address is driven on the **mem_addr<11:0>** pins and the **mem_dtoe_l** signal is asserted. The **mem_data<63:0>** and **mem_ecc<7:0>** pins are tristated. If the Bcache was accessed immediately before this cycle, the **bc_oe_l** and **bc_cs_l** signals are deasserted. In the case when the Bcache was accessed, the cache index driven on the **mem_addr<11:0>** pins is the same as the column address, and the first cycle of Figure 5–20 corresponds to the last cycle of the Bcache read operation (Figure 5–18).
2. After T_{45} cycles, the **mem_cas_l** signal is asserted.
3. After T_{56} cycles, the **mem_data<63:0>** and **mem_ecc<7:0>** signals from the DRAMs are latched and the **mem_cas_l** signal is deasserted. If the conditions for RAS precharge in Table 5–23 are met, the **mem_ras_l** signal is deasserted.

Figure 5–20 shows the timing for a page-mode read cycle.

Figure 5–20 Page-Mode Read Timing



5.7.9 Non-Page-Mode Read Timing

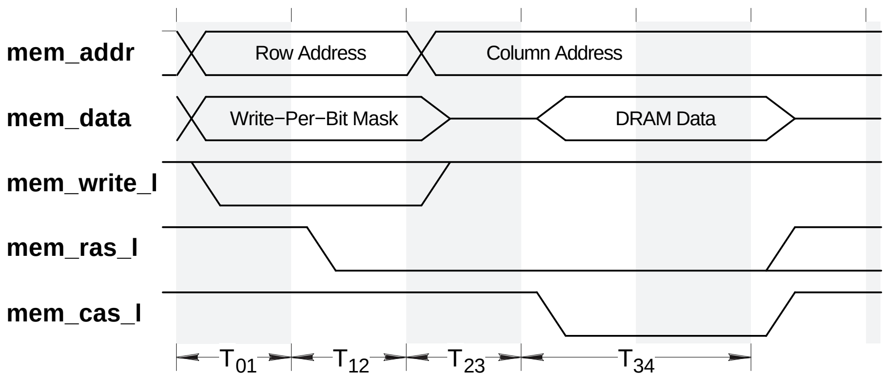
A non-page-mode read cycle can occur when a read operation is required from one of the operations in Table 5–23 and the page-mode conditions in Table 5–24 are not met.

The sequence is as follows:

1. The row address is driven on the **mem_addr<11:0>** pins. If the write-per-bit conditions in Table 5–24 are met, the write mask is driven on the **mem_data<63:0>** pins and the **mem_write_l** signal is asserted.
2. After T_{01} cycles, the **mem_ras_l** signal is asserted.
3. After T_{12} cycles, the column address is driven on the **mem_addr<11:0>** pins. The **mem_data<63:0>** and **mem_ecc<7:0>** pins are tristated.
4. After T_{23} cycles, the **mem_cas_l** signal is asserted.
5. After T_{34} cycles, the **mem_data<63:0>** and **mem_ecc<7:0>** signals from the DRAMs are latched and the **mem_cas_l** signal is deasserted. If the RAS precharge conditions in Table 5–23 are met, the **mem_ras_l** signal is deasserted.

Figure 5–21 shows the timing for a non-page-mode read cycle.

Figure 5–21 Non-Page-Mode Read Timing



5.7.10 Page-Mode Write Timing

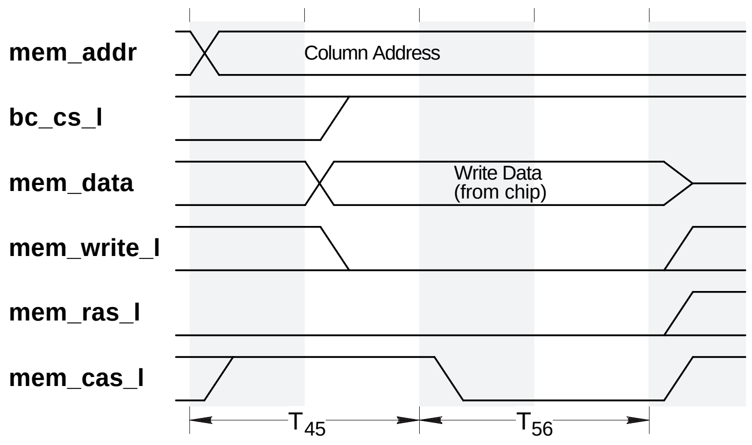
A page-mode write cycle can occur when a write operation is required from one of the operations in Table 5–23 and the page-mode conditions in Table 5–24 are met.

The sequence is as follows:

1. The column address is driven on the **mem_addr<11:0>** pins. Write data and ECC are driven on the **mem_data<63:0>** and **mem_ecc<7:0>** pins. The **mem_write_l** signal is asserted. If the Bcache was accessed immediately before this cycle, the **bc_oe_l** and **bc_cs_l** signals are deasserted. In the case when the Bcache was accessed, the cache index driven on the **mem_addr<11:0>** pins is the same as the column address, and the first cycle of Figure 5–22 corresponds to the last cycle of the Bcache read (Figure 5–18).
2. After T_{45} cycles, the **mem_cas_l** signal is asserted.
3. After T_{56} cycles, the **mem_cas_l** and **mem_write_l** signals are deasserted. If the RAS precharge conditions in Table 5–23 are met, the **mem_ras_l** signal is deasserted.

Figure 5–22 shows the timing for a page-mode write cycle.

Figure 5–22 Page-Mode Write Timing



5.7.11 Non-Page-Mode Write Timing

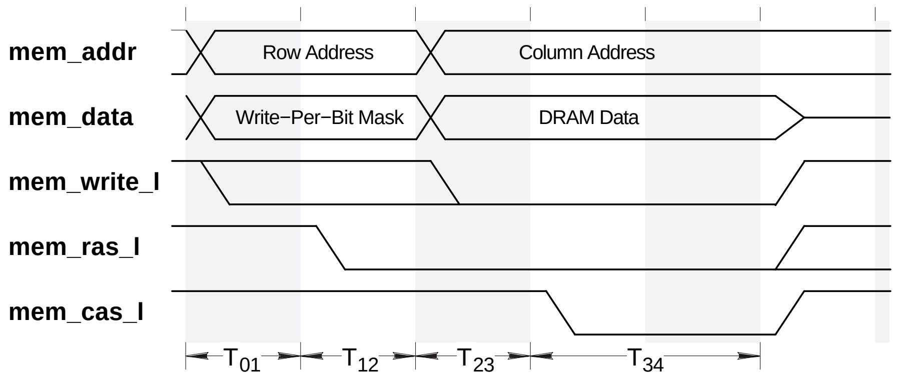
A non-page-mode write cycle can occur when a write operation is required from one of the operations in Table 5–23 and the page-mode conditions in Table 5–24 are not met.

The sequence is as follows:

1. The row address is driven on the **mem_addr<11:0>** pins. If the write-per-bit conditions in Table 5–24 are met, the write mask is driven on the **mem_data<63:0>** pins and the **mem_write_l** signal is asserted.
2. After T_{01} cycles, the **mem_ras_l** signal is asserted.
3. After T_{12} cycles, the column address is driven on the **mem_addr<11:0>** pins. The **mem_write_l** signal is asserted, and the **mem_data<63:0>** and **mem_ecc<7:0>** signals are driven.
4. After T_{23} cycles, the **mem_cas_l** signal is asserted.
5. After T_{34} cycles, the **mem_cas_l** is deasserted. If the RAS precharge conditions in Table 5–23 are met, the **mem_ras_l** signal is deasserted.

Figure 5–23 shows the timing for a non-page-mode write cycle.

Figure 5–23 Non-Page-Mode Write Timing



5.8 Memory Controller Signals

Sections 5.8.1 through 5.8.20 describe the function of each memory controller signal.

5.8.1 mem_data<63:0>

The **mem_data<63:0>** signals transmit data to and from memory or Bcache and the memory controller.

During a read from the Bcache, the Bcache drives the **mem_data<63:0>** pins when the **bc_oe_l** signal is asserted.

During a read from memory, memory (or optional transceivers) drives the **mem_data<63:0>** pins when the **mem_cas_l** and **mem_rd_oe** signals are asserted and the **mem_write_l** signal is not asserted.

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

During a write-per-bit operation, the **mem_data<63:0>** pins are driven with the write-per-bit mask by the memory controller and are valid when the **mem_ras_l**, **mem_wr_oe_l**, and **mem_write_l** signals are asserted.

The **mem_data<63:0>** lines are weakly pulled down when a memory operation is not in progress.

5.8.2 mem_ecc<7:0>

The **mem_ecc<7:0>** signals transmit the error correction codes (ECC) to and from memory or the Bcache and the memory controller. The memory controller generates ECC for writes and checks it for reads. The **mem_ecc<7:0>** signals have the same external timing as the **mem_data<63:0>** signal. Memory storage for ECC is optional, and ECC checking can be disabled using the bank configuration registers (BCR3–BCR0, Section 5.6.2). Table 5–2 shows the correspondence between the **mem_ecc<7:0>** and **mem_data<63:0>** signals.

The **mem_ecc<7:0>** signals also transmit a byte write mask for banks that have byte write enabled in their BCR. External logic is required to gate the DRAM write signals. When used in this way, the **mem_ecc0** signal corresponds to the **mem_data<7:0>** signals, the **mem_ecc1** signal corresponds to the **mem_data<15:8>** signals, and so on. The mask is active high; that is, a value of one allows writing and a value of zero inhibits writing.

The **mem_ecc<7:0>** pins are weakly pulled down when a memory operation is not in progress.

5.8.3 mem_addr<11:0>

The **mem_addr<11:0>** signals transmit the row and column address to memory, and the 12 least significant bits (LSBs) of the index to the Bcache (Figure 5–10). For memory read and write operations (that is, not refresh), the **mem_addr<11:0>** signals contain a valid row address when the **mem_ras_l** signal is asserted and a valid column address when the **mem_cas_l** signal is asserted.

5.8.4 mem_write_l

The **mem_write_l** signal provides the read and write control for memory and enables the write-per-bit mask to be loaded. For a write operation to the memory devices, the **mem_write_l** signal is asserted before the **mem_cas_l** signal is asserted. For a read operation to the memory devices, the **mem_write_l** signal is deasserted before the **mem_cas_l** signal is asserted.

The write-per-bit function is used to write selected bits to memory. To activate the write-per-bit function in the DRAMs and VRAMs, the **mem_write_l** signal is asserted before the **mem_ras_l** signal is asserted. The value in the **mem_data<63:0>** bits is loaded into the memory's internal write mask latch. Data bits that are low when the **mem_ras_l** signal is asserted inhibit subsequent write operations to that bit. Data bits that are high when the **mem_ras_l** signal is asserted are modified during the write operation. If the **mem_write_l** signal is deasserted when the **mem_ras_l** signal is asserted, the write-per-bit capability is disabled and the write operation is performed on all data bits.

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

Note

Where specificity is not required, the eight **mem_rasa_l<3:0>** and **mem_rasb_l<3:0>** signals are often abbreviated to the generic term **mem_ras_l**.

Each pair of **mem_rasa_l<3:0>** and **mem_rasb_l<3:0>** signals are associated with a bank of memory. During a normal read or write operation, the assertion of the **mem_ras_l** signal indicates that the **mem_addr<11:0>** bits contain a valid row address. Which RAS is asserted depends on which bank is addressed, whether or not dual bank is enabled in the BCR (Section 5.6.2), and the row address bit shown in Table 5–5.

During a memory refresh cycle, if the refresh enable bit is set in the global timing register (GTR, Section 5.6.5), all **mem_rasa_l<3:0>** and **mem_rasb_l<3:0>** outputs are asserted together. Refresh cycles are the CAS-before-RAS type.

During VRAM shift-register loads, the **mem_rasa_l*n*** **mem_rasb_l*n*** signals for bank *n* are both asserted.

5.8.6 mem_cas_l

When asserted during memory read and write operations, the **mem_cas_l** signal indicates that the **mem_addr<11:0>**, **mem_data<63:0>** (for write operations), and **mem_write_l** signals are valid.

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

5.8.7 mem_dtoe_l

The multifunction **mem_dtoe_l** pin controls either the output enable function for DRAMs or the shift-register load function for VRAMs.

For DRAM operations, the **mem_dtoe_l** signal is deasserted when the **mem_ras_l** signal is asserted. To enable the memory's data output drivers during a read cycle, the **mem_dtoe_l** signal is asserted before the **mem_cas_l** signal is asserted.

To indicate a shift-register load sequence to VRAMs, the **mem_dtoe_l** signal is asserted before the **mem_ras_l** signal is asserted. Either a full or split register-load sequence will be performed, depending on the value of the **mem_dsf** signal.

5.8.8 mem_dsf

The **mem_dsf** special function select signal determines whether a full or split shift-register load is to be performed, as shown in Table 5–25. The **mem_dsf** signal is valid before the **mem_ras_l** signal is asserted when the **mem_dtoe_l** signal is asserted.

Table 5–25 VRAM Shift-Register Load Selection

mem_dtoe_l	mem_dsf	Function at RAS Assertion
1	1 or 0	Normal read or write
0	0	Full shift-register load
0	1	Split shift-register load

A full shift-register load cycle loads a full row of memory data into the VRAM's shift register. A split shift-register load cycle loads only one-half of a row into one-half of the VRAM's shift register. The column address MSB determines which half of the row is to be loaded (some VRAMs control this internally). The remaining column address bits are zeros.

5.8.9 mem_rd_oe

The **mem_rd_oe** signal enables the optional, external memory transceiver to drive data from the memory parts on the **mem_data<63:0>** and **mem_ecc<7:0>** lines. It is asserted during read cycles when the **mem_cas_l** signal is asserted. If a transceiver is not used, the **mem_rd_oe** pin should be ignored.

5.8.10 mem_wr_oe_l

The **mem_wr_oe_l** signal enables the optional, external memory transceiver to drive data from the **mem_data<63:0>** and **mem_ecc<7:0>** pins to the memory parts. It is asserted during write cycles when the **mem_data<63:0>** pins are driven. If a transceiver is not used, the **mem_wr_oe_l** pin should be ignored.

5.8.11 bc_tag<7:0>

The **bc_tag<7:0>** signals represent the upper part of the physical address stored in a cache block (Figure 5–10). During a Bcache lookup, the cache drives these bits (and possibly the **bc_idx_tag<4:0>** bits—Section 5.8.15) with the tag corresponding to the current index being driven on the **mem_addr<11:0>** pins, the **bc_index** pin, and the non-tag bits of the **bc_idx_tag<4:0>** pins. During a Bcache fill or write operation, the memory controller drives the upper bits of the address on the **bc_tag<7:0>** pins, to be stored in the Bcache tag array.

The **bc_tag<7:0>** pins are weakly pulled down when a memory operation is not in progress.

5.8.12 bc_dirty

The **bc_dirty** signal indicates the status of the data stored in a Bcache block. A value of zero indicates that the cache and memory contain the same data; a value of one indicates that the cache contains more recently written data than memory (in other words, memory data is stale). The **bc_dirty** signal has the same timing characteristics as the **bc_tag<7:0>** signals.

The **bc_dirty** pin is weakly pulled down when a memory operation is not in progress.

5.8.13 bc_parity

The **bc_parity** signal transmits the Bcache tag parity to the tag array during cache fill and write operations and from the tag array during cache lookups. The **bc_parity** signal has the same timing characteristics as the **bc_tag<7:0>** signals.

The **bc_parity** pin is weakly pulled down when a memory operation is not in progress.

5.8.14 bc_index

The **bc_index** signal transmits Bcache index bit 12 to the cache (Figure 5–10).

5.8.15 bc_idx_tag<4:0>

Each **bc_idx_tag<4:0>** signal can act as either an index bit or a tag bit, depending on the size of the Bcache. The function of each bit is determined by the size of the Bcache, as shown in Table 5–26 and Figure 5–10.

Table 5–26 Use of bc_idx_tag

	bc_idx_tag<4:0>				
Bcache Size	**4**	**3**	**2**	**1**	**0**
64 KB	Tag	Tag	Tag	Tag	Tag
128 KB	Tag	Tag	Tag	Tag	Index
256 KB	Tag	Tag	Tag	Index	Index
512 KB	Tag	Tag	Index	Index	Index
1 MB	Tag	Index	Index	Index	Index
2 MB	Index	Index	Index	Index	Index

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

The **bc_idx_tag<4:0>** pins are weakly pulled down when a memory operation is not in progress.

5.8.16 bc_cs_l

The **bc_cs_l** signal is the chip select for the Bcache SRAMs. The SRAMs are enabled when this signal is asserted; otherwise, they are disabled. The **bc_cs_l** signal is deasserted when the cache is not being accessed and the cache power-saving mode is on; that is, the PWR bit is set in the cache register (CAR, Section 5.6.8).

5.8.17 bc_we_l

The **bc_we_l** signal is asserted to write the data from the **mem_data<63:0>** and **mem_ecc<7:0>** signals into the Bcache data SRAM array, and to write the tag from the **bc_tag<7:0>**, **bc_idx_tag<4:0>**, **bc_dirty**, and **bc_parity** signals into the tag SRAMs.

5.8.18 bc_oe_l

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

5.8.19 vframe_l

When the **vframe_l** signal is asserted, the memory controller does the following:

1. Reloads the video display pointer (Section 5.6.9.1) with the value stored in the start of the video frame field of the video and graphics control register (VGR, Section 5.6.9).
2. Performs a full shift-register load cycle to the VRAM bank selected by the start of video frame field of the VGR.
3. Increments the video display pointer twice, as specified by the address increment (INC) field of the VGR.

The **vframe_l** signal is edge-sensitive. The assertion (falling) edge is captured by an edge-detection circuit, the output of which is synchronized and allowed to settle before being used. The edge-detection circuit is rearmed when the shift-register load cycle is done. A subsequent assertion of the **vframe_l** or **vrefresh_l** signal before the shift-register load cycle is completed will not be captured.

5.8.20 vrefresh_l

When the **vrefresh_l** signal is asserted, the memory controller does the following:

1. Performs a split shift-register load cycle to the VRAM bank selected by the start of the video frame field of the video and graphics control register (VGR, Section 5.6.9).
2. Increments the video display pointer (Section 5.6.9.1) as specified by the INC field of the VGR.

The **vrefresh_l** signal is edge-sensitive. The assertion (falling) edge is captured by an edge-detection circuit, the output of which is synchronized and allowed to settle before being used. The edge-detection circuit is rearmed when the shift-register load cycle is done. A subsequent assertion of the **vframe_l** or **vrefresh_l** signal before the shift-register load cycle is completed will not be captured.

6 I/O Controller

This chapter describes the I/O controller (IOC); I/O operations; and IOC error handling, registers, and interface signals.

This chapter does not duplicate information that can be found in the *PCI Local Bus Specification, Revision 2.0*.

6.1 Overview

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

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

6.1.1 Scatter-Gather Mapping Support

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

6.1.2 Sparse and Dense Memory Support

The IOC supports both sparse and dense memory space.

Sparse memory space is a region in which the lower bits of the CPU address indicate transfer size, and the address is shifted down by 5 bits during the translation from a CPU address to a PCI address. The host address extension register (IOC_HAE, Section 6.4.1) extends the shifted address to 32 bits. The size of transfers in this space can be byte, word, tribyte, longword, and quadword. The CPU references sparse address space when CPU address bits <33:32> = 10. See Section 6.2.6.1 for more information about how the low-order CPU address bits are encoded.

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

6.1.3 PCI Parity Support

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

Note

Because the chip does not provide an **serr_l** signal pin, external logic must report **serr_l** assertions on one of the chip's **irq<2:0>** interrupt signal pins.

Usually, all address parity errors and data parity errors during PCI special cycles are reported with the PCI **serr_l** signal. Such errors detected by the IOC (which PCI devices would normally report on **serr_l**) are reported to the CPU by an internal interrupt. External logic can report the assertion of the **serr_l** signal by PCI peripherals using the chip's interrupt signal pins

irq<2:0>. The **serr_l** signal is an asynchronous PCI signal that is usually asserted one cycle after valid parity is driven on the **par** signal (that is, two cycles after valid address or data). PCI peripherals can also use the **serr_l** signal to report other severe error conditions.

Data parity errors on all PCI transactions (except during special cycles) are reported with the PCI **perr_l** signal. Only data parity errors are reported on the **perr_l** line. The chip provides a signal pin to connect the **perr_l** signal directly to the IOC. The **perr_l** signal is synchronous with the PCI clock. Data parity errors are reported on the **perr_l** signal one cycle after valid parity is driven on the **par** signal (that is, two cycles after valid data).

The IOC parity disable register (IOC_PAR_DIS, Section 6.4.8) allows parity checking and reporting to be disabled. When this register is written, the IOC does not log or report parity errors.

6.1.4 PCI Exclusive Access Support

The IOC supports the PCI exclusive access protocol for peripheral-initiated transfers with system memory (also called direct memory access or DMA). The chip cannot generate exclusive access cycles for CPU-initiated PCI transfers.

6.1.5 Clocking

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

The IOC provides the **pci_clk_in** pin for the PCI clock input. It controls PCI signal latching in the IOC and generates the control strobes for the IOC outputs. The **pci_clk_in** signal is asynchronous to the internal CPU clock. The PCI and CPU clock domains are synchronized in the IOC.

6.2 CPU-Initiated PCI Cycles

The IOC operates as a PCI master when the CPU executes a load or store instruction that addresses a PCI peripheral. The IOC provides address windows to the CPU that allow access to the PCI memory, I/O, and configuration address spaces. The interrupt vector and special cycle register (IOC_IACK_SC, Section 6.4.13) generates either a PCI interrupt acknowledge cycle when read or a PCI special cycle when written.

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

Table 6–1 CPU-to-PCI Address Translation

Read Write	CPU a<33:32	31:29>	PCI ad<31:27	26:24	23:2	1:0>	PCI c_be_l <3:0>	Transaction
Write	11	XXX	**a<31:27>**	**a<26:24>**	**a<23:2>**	00	0111	DMW
Read	11	XXX	**a<31:27>**	**a<26:24>**	**a<23:2>**	00	0110	DMR
Write	10	≠ 0	HAE<31:27>	**a<31:29>**	**a<28:7>**	00	0111	SMW
Write	10	000	00000	**a<31:29>**	**a<28:7>**	00	0111	SMW
Read	10	≠ 0	HAE<31:27>	**a<31:29>**	**a<28:7>**	00	0110	SMR
Read	10	000	00000	**a<31:29>**	**a<28:7>**	00	0110	SMR
Write	01	111	00000	000	**a<28:7>**	CFG<1:0>	1011	CFW
Read	01	111	00000	000	**a<28:7>**	CFG<1:0>	1010	CFR
Write	01	110	00000	000	**a<28:7>**	**a<6:5>**	0011	IOW
Read	01	110	00000	000	**a<28:7>**	**a<6:5>**	0010	IOR
Write	01	101	HAE<31:27>	**a<31:29>**	**a<28:7>**	00	0001	SPC
Read	01	101	HAE<31:27>	**a<31:29>**	**a<28:7>**	00	0000	IAK

Key to abbreviations:

CFG—Configuration cycle type register
CFR—Configuration read
CFW—Configuration write
DMR—Dense memory read
DMW—Dense memory write
HAE—Host address extension register
IAK—Interrupt acknowledge
IOR—I/O read
IOW—I/O write
SMR—Sparse memory read
SMW—Sparse memory write
SPC—Special cycle
X—1 or 0
≠ 0—Non-zero

Table 6–2 lists the supported PCI address spaces and command codes.

Table 6–2 CPU Address Windows to PCI Address Spaces

		PCI Command Code c_be_l<3:0>	
CPU Address <33:0>	**PCI Address Space**	**Read Operations**	**Write Operations**
3 FFFF FFFF..3 0000 0000	Dense memory	0110	0111
2 FFFF FFFF..2 0000 0000	Sparse memory	0110	0111
1 FFFF FFFF..1 E000 0000	Configuration	1010	1011
1 DFFF FFFF..1 C000 0000	I/O	0010	0011
1 BFFF FFFF..1 A000 0000	Special cycle	NA	0001
1 BFFF FFFF..1 A000 0000	Interrupt acknowledge	0000	NA

NA = not applicable

6.2.1 CPU Request Queue

The CPU request queue is used for all CPU-initiated PCI transfers and for CPU accesses to registers that are internal to the IOC. The two-entry CPU request queue can hold an address and up to one quadword of write data for CPU-initiated PCI transactions as follows:

- Up to two CPU-initiated PCI write transfers
- Only one CPU-initiated PCI read transfer or one CPU access to an IOC register (read or write)
- One CPU-initiated PCI write transfer in the first entry and one CPU-initiated PCI read transfer or one CPU access to an IOC register in the second entry

6.2.2 Requesting PCI Mastership

When a CPU-initiated PCI read or write cycle is queued in the IOC and the IOC grant signal **gnt_l** is not asserted, the IOC asserts its bus request signal **req_l**. External arbitration logic is required to arbitrate requests from the various PCI devices in the system.

When the IOC **gnt_l** signal is asserted and the PCI bus is idle, the IOC assumes bus ownership. (The PCI bus is idle when both the **frame_l** and **irdy_l** signals are sampled deasserted.) In the next cycle, the IOC initiates a PCI transfer (Section 6.2.5).

The IOC automatically deasserts its **req_l** signal in the same cycle that it initiates the PCI bus cycle and asserts its **frame_l** signal. If another request is queued in the IOC and the IOC **gnt_l** signal is not asserted (that is, the PCI bus is not parked with the CPU), the IOC will not reassert its **req_l** signal until one cycle after the completion of the current IOC-initiated PCI transaction.

6.2.3 Default Mastership (Bus Parking)

The *PCI Local Bus Specification, Revision 2.0* defines bus parking as granting the PCI bus to a device when no requests are pending.

Digital recommends that the arbiter grant bus ownership to the CPU by asserting the IOC **gnt_l** signal when no devices are requesting bus mastership. This reduces the latency for CPU-initiated PCI transfers when the bus is idle.

If the IOC is granted the bus when it is not asserting its **req_l** signal, it enables the **ad<31:0>**, **c_be_l<3:0>**, and **par** output drivers one cycle after its **gnt_l** signal is asserted to prevent the drivers from floating. If the IOC has not initiated a PCI transfer, it tristates these output drivers one cycle after its **gnt_l** signal is deasserted.

If the IOC **gnt_l** signal is asserted, the PCI is idle (both the **frame_l** and **irdy_l** signals are sampled deasserted), and a request is queued in the IOC, the IOC initiates a PCI transaction on the next PCI cycle without asserting its **req_l** signal.

6.2.4 Memory Barrier Instruction Requirements

The memory barrier (MB) instruction must be used to guarantee strict write-ordering and prevent write-merging operations in the internal CPU write buffer when not addressing dense space. (Merging in the write buffer is permitted for dense memory space.) When executed, the MB instruction forces any data in the internal CPU write buffers to be flushed.

Because a shifted address is used to access PCI sparse memory spaces, the behavior of the internal write buffer is important. Write operations to addresses on the PCI that differ by at least a byte address will be written to different hexword entries in the internal CPU write buffer. Write operations to the same PCI address with different length encodings will be written to different quadwords within the same hexword write buffer entry. However, the IOC accepts only a longword or a quadword of data from any write buffer entry. Therefore, to ensure correct operation, write operations to the same PCI sparse address using different length encodings (byte, word, tribyte, and unmasked) require the use of memory barrier instructions between each write operation. In addition, write operations with the same length encoding to the same PCI byte address will get merged in the internal CPU write buffer if they are not separated by memory barrier instructions.

6.2.5 Address Phase

All PCI transfers begin with the address phase. If the cycle immediately after the IOC detects both a bus idle condition and the assertion of the **gnt_l** signal, the IOC initiates a PCI transaction as follows:

- Drives the PCI target address on the **ad<31:0>** pins
- Drives a valid cycle code on the **c_be_l<3:0>** pins
- Asserts the **frame_l** signal
- Drives even parity for the address on the **par** pin in the next cycle

If any PCI device detects an address parity error, it reports it on the PCI **serr_l** signal two cycles after the **frame_l** signal is asserted. (See Section 6.2.12 for more information.)

Note

Because the chip does not provide an **serr_l** signal pin, external logic must report **serr_l** assertions on one of the chip's **irq<2:0>** interrupt signal pins.

Configuration cycles use address stepping, which drives the address and command for two consecutive cycles. The **frame_l** signal is asserted during the second cycle.

6.2.6 Sparse Memory Space

The sparse memory space is the region in which maskable, CPU-initiated PCI transactions occur. This region uses the low bits of the CPU address to indicate transfer size and uses the IOC_HAE register to extend the shifted address to cover the 32-bit PCI address space.

6.2.6.1 Generating PCI Addresses for Sparse Space

CPU physical address bits <31:7> are used for PCI address bits <26:2>. To support various transfer size (byte enable) combinations in sparse space, the low-order bits of the CPU physical addresses that reference PCI address spaces are encoded as follows:

- Physical address bits <6:3> generate the PCI byte enable **c_be_l<3:0>** signals. Byte enables generated by combinations of physical address bits <6:3>, not shown in Table 6–3, are unspecified.

- Physical address bits <4:3> also specify transfer size.
- Physical address bits <2:0> must be zero for transfers in sparse space.

Table 6–3 shows how the low-order physical address bits are used to indicate transfer size.

Table 6–3 Byte Enable Encoding for CPU-Initiated Transfers in Sparse Space

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

[*]Byte enables are active low (asserted when shown as zero).

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

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

Although the PCI supports arbitrary byte enables, the IOC supports only the byte-enable combinations listed in Table 6–3.

Masked Transfers: The following types of masked transfers are permitted in sparse space:

- Longword read transfers—An LDL instruction can generate a masked PCI read transfer with a burst length of one.
- Longword write transferss—An STL instruction must be used. The STL instruction generates a PCI write transfer with a burst length of one.

Unmasked Transfers: The following types of unmasked transfers are permitted in sparse space:

- Longword read transfers—An LDL instruction can generate an unmasked PCI read transfer with a burst length of one.
- Quadword read transfers—An LDQ instruction can generate an unmasked PCI read transfer with a burst length of two.
- Longword write transfers—An STL instruction can generate an unmasked PCI write transfer with a burst length of one.
- Quadword write transfers—An STQ instruction can generate an unmasked PCI write transfer with a burst length of two.

6.2.6.2 Address Extension for Sparse Space

Because the CPU address is encoded to generate PCI byte enables, only 27 physical address bits (CPU address bits **a<31:5>**) can be used to generate PCI address bits **ad<26:0>**, giving an effective PCI address space of 128 MB. This 128-MB address space is subdivided into a 16-MB region and a 112-MB region. Generation of PCI address bits **ad<31:27>** depends on which of these regions are referenced, as shown in Table 6–4.

Table 6–4 Host Address Extension

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

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

The 112-MB region is referenced when CPU address bits **a<31:29>** are non-zero. During CPU-initiated PCI transactions to this 112-MB region, PCI address bits **ad<31:27>** are generated using the host address extension (HAE) field in the IOC_HAE register (Section 6.4.1). The HAE field allows the 112-MB region to be relocated to the upper 112 MB of any aligned 128-MB PCI address segment. A value of zero in the HAE field defines a single 128-MB region, beginning at address 0 in PCI address space.

6.2.6.3 Burst Order in Sparse Space

For sparse memory space, PCI address bits <1:0> in memory cycles specify the burst order requested by the bus master. The microprocessor always uses a linearly incrementing burst order. Only a CPU-initiated PCI unmasked-quadword read or write cycle can generate a transfer with a burst length greater than one (Section 6.2.6.6). For dense space, bursts can be greater than one.

6.2.6.4 Unmasked Longword and Quadword Read Operations to Sparse Space

CPU-initiated PCI longword read operations are generated by executing LDL or LDQ instructions that reference a physical address in the PCI sparse memory space. Encoded CPU address bits <6:5> must equal 00 for a longword and 11 for a quadword; CPU address bits <4:3> must equal 11, as specified in Table 6–3.

Address Phase: In the address phase of a CPU-initiated PCI longword or quadword read cycle, the IOC drives a memory read command on the **c_be_l<3:0>** pins.

Address Phase + 1: In the cycle following the address phase of a longword read cycle, the IOC deasserts the **frame_l** signal, asserts the PCI initiator ready signal **irdy_l**, and begins to sample the target ready signal **trdy_l** and address signals **ad<31:0>**. When the IOC samples the **trdy_l** signal asserted, it latches the read data from the **ad<31:0>** pins.

In the cycle following the address phase of a quadword read cycle, the IOC deasserts the **frame_l** signal after it samples the **trdy_l** signal asserted.

Address Phase + 2: In the second cycle following the address phase of a longword read cycle, the IOC deasserts its **irdy_l** signal and samples the parity data that the target is driving on the **par** line. The IOC returns the longword of read data from the PCI to the CPU in the low longword of the returned quadword.

In the second cycle following the address phase of a quadword read cycle, the IOC latches both the upper longword on the **ad<31:0>** pins and the parity associated with the first longword. Parity associated with the second longword will appear in the third cycle. The IOC packs the two longwords of read data into the single quadword that is returned to the CPU.

If the parity calculated by the IOC does not match the sampled parity, the IOC uses an internal interrupt to report the error to the CPU (Section 6.2.13.1).

Deassertion of the **irdy_l** signal indicates cycle completion to the PCI target device.

6.2.6.5 Unmasked Longword Write Operations to Sparse Space

CPU-initiated PCI longword write operations are generated by executing STL instructions that reference a physical address in the PCI sparse memory space. Encoded CPU address bits <6:5> must equal 00 and bits <4:3> must equal 11, as specified in Table 6–3.

Address Phase: In the address phase of a CPU-initiated PCI longword write cycle, the IOC drives a memory write command on the **c_be_l<3:0>** pins.

Address Phase + 1: In the cycle following the address phase, the IOC does the following:

- Deasserts the **frame_l** signal
- Asserts the PCI initiator ready signal **irdy_l**
- Drives valid write data on the **ad<31:0>** pins
- Begins to sample the target ready signal **trdy_l**

Address Phase + 2: In the second cycle following the address phase, the IOC drives even parity for the **ad<31:0>** and **c_be_l<3:0>** signals on the **par** pin.

Target Ready + 1: In the cycle after the IOC samples the **trdy_l** signal asserted, it deasserts its **irdy_l** signal, completing the longword write transaction.

If the target of the write transaction detects a parity error, it asserts the **perr_l** signal two cycles after it asserts the **trdy_l** signal. If the IOC samples the **perr_l** signal asserted in this cycle (two cycles after the data transfer), it uses an internal interrupt to report a parity error to the CPU (Section 6.2.13.2).

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

6.2.6.6 Unmasked Quadword Write Operations to Sparse Space

CPU-initiated PCI quadword write operations are generated by executing STQ instructions that reference a physical address in the PCI sparse memory space. Encoded CPU address bits <6:5> must equal 11 and bits <4:3> must equal 11, as specified in Table 6–3.

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

Address Phase: In the address phase of a CPU-initiated PCI quadword write cycle, the IOC drives a memory write command on the **c_be_l<3:0>** pins.

To complete the quadword write cycle, the IOC initiates a 2-longword burst write cycle by using linearly incrementing burst order on the PCI.

Address Phase + 1: In the cycle following the address phase, the IOC does the following:

- Continues to assert the **frame_l** signal
- Asserts the PCI initiator ready signal **irdy_l**
- Drives the lower longword of write data on the **ad<31:0>** pins
- Begins to sample the target ready signal **trdy_l**

Address Phase + 2: In the second cycle following the address phase, the IOC drives even parity for the **ad<31:0>** and **c_be_l<3:0>** signals on the **par** pin.

First Target Ready + 1: In the cycle after the IOC samples the **trdy_l** signal asserted, the IOC does the following:

- Drives the upper longword of write data on the **ad<31:0>** pins
- Deasserts its **frame_l** signal
- Continues to assert its **irdy_l** signal

First Target Ready + 2: In the second cycle after the IOC samples the **trdy_l** signal asserted, the IOC drives even parity for the **ad<31:0>** and **c_be_l<3:0>** signals on the par pin.

Second Target Ready + 1: In the cycle after the IOC again samples the **trdy_l** signal asserted, it deasserts its **irdy_l** signal, completing the quadword write transaction.

If the target of the write transaction detects a parity error on either longword, it asserts the **perr_l** signal two cycles after it asserts the **trdy_l** signal corresponding to the data transfer that caused the parity error. If the IOC samples the **perr_l** signal asserted two cycles after either data transfer, it uses an internal interrupt to report a parity error to the CPU (Section 6.2.13.2).

6.2.6.7 Masked Longword Read Operations to Sparse Space

CPU-initiated PCI longword read operations are generated by executing LDL instructions that reference a physical address in the PCI sparse memory space. Encoded CPU address bits <4:3> must equal 00, 01, or 10. The encoded address determines the byte mask used during the transfer, as specified in Table 6–3.

Masked longword read operations are similar to unmasked longword read operations (Section 6.2.6.4), except that when the IOC is asserting the **irdy_l** signal, it is also asserting only the byte enables specified by the encoded address.

6.2.6.8 Masked Longword Write Operations to Sparse Space

CPU-initiated PCI longword write operations are generated by executing STL instructions that reference a physical address in the PCI sparse memory space. Encoded CPU address bits <4:3> must equal 00, 01, or 10. The encoded address determines the byte mask used during the transfer, as specified in Table 6–3.

Masked longword write operations are similar to unmasked longword write operations (Section 6.2.6.4), except that when the IOC is asserting the **irdy_l** signal, it is also asserting only the byte enables specified by the encoded address.

6.2.7 Dense Memory Space

The dense memory space is the region in which only unmasked longword or quadword transfers can be specified. This memory space uses the onchip write buffer more efficiently by condensing multiple quadword transfers into a single burst.

6.2.7.1 Generating PCI Addresses for Dense Space

Dense memory space uses the unshifted CPU address bits **a<31:0>** to generate the PCI address. Because this region has sufficient address space, the IOC_HAE register is not needed or used to generate PCI addresses. Note that any PCI address can be generated from dense space. Sparse space address generation must follow the address extension described in Section 6.2.6.2.

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

Because the address is not shifted, masked transfers are not permitted in dense space.

6.2.7.2 Unmasked Read Operations to Dense Space

The PCI returns an aligned quadword in response to a longword read operation, regardless of the address alignment. The correct longword is returned to the register file in the low 32 bits.

The PCI returns an aligned quadword in response to a quadword read operation. The address is assumed to be quadword aligned.

6.2.7.3 Unmasked Burst Write Operations to Dense Space

Unmasked PCI write operations can transfer up to 1 hexword by executing a combination of STL and STQ instructions with addresses that fall within an aligned hexword. All longwords of the hexword need not be written; that is, the software can write any longword portion of the hexword—the IOC simply sequences through an unwritten longword by deasserting the byte enables for that longword.

The IOC sequences through successive longwords by using a linearly incrementing burst order on the PCI.

The IOC always transfers the low longword of a quadword first, followed by the high longword.

Address Phase + 1: In the cycle following the address phase, the IOC does the following:

- Continues to assert the **frame_l** signal
- Asserts the PCI initiator ready signal **irdy_l**
- Drives the first longword of write data on the **ad<31:0>** pins
- Begins to sample the target ready signal **trdy_l**

First Target Ready + 1: In the cycle after the IOC samples the **trdy_l** signal asserted, the IOC does the following:

- Drives the next longword of write data on the **ad<31:0>** pins
- Deasserts its **frame_l** signal
- Continues to assert its **irdy_l** signal

This process continues until the last longword of the transaction.

Second Target Ready + 1: In the cycle after the IOC again samples the **trdy_l** signal asserted, it deasserts its **irdy_l** signal, completing the burst.

If the target of the write transaction detects a parity error on a longword, it asserts the **perr_l** signal two cycles after it asserts the **trdy_l** signal corresponding to the data transfer that caused the parity error. If the IOC samples the **perr_l** signal asserted two cycles after a data transfer, it uses an internal interrupt to report a parity error to the CPU (Section 6.2.13.2).

6.2.8 I/O Space Cycles

The IOC generates addresses in this region for transactions to the I/O region specified by the PCI architecture. This region supports the same access types and masked-address generation as sparse space (Section 6.2.6), with the following exceptions:

- I/O space is accessed with CPU address bits **a<33:32>** = 01 and **a<31:29>** = 110.
- PCI address bits **ad<31:27>** = 00000 and the IOC_HAE register is not used.
- PCI address bits **ad<1:0>** come directly from physical address bits **a<6:5>** during CPU-initiated PCI transfers to I/O space.
- An I/O read or write command (Table 6–2) is issued on the **c_be_l<3:0>** pins coincident with a read or write operation to this region.

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

6.2.9 Configuration Cycles

The PCI architecture specifies a configuration address space for defining configuration registers that are used to initialize and configure devices. The IOC supports transfers to the configuration address space to access the configuration registers. This region supports the same access types and masked address generation as sparse space (Section 6.2.6), with the following exceptions:

- Configuration space is accessed with CPU address bits **a<33:32>** = 01 and **a<31:29>** = 111.
- PCI address bits **ad<31:27>** = 00000 and the IOC_HAE register is not used.
- PCI address bits **ad<1:0>** come directly from configuration cycle type register (IOC_CONF, Section 6.4.2) bits <1:0> during CPU-initiated PCI transfers to configuration space.

- A configuration space read or write command (Table 6–2) is issued on the **c_be_l<3:0>** pins coincident with a read or write operation to this region.

During configuration cycles, addresses are restricted both by the encoding requirements of PCI configuration cycles (as described in the *PCI Local Bus Specification, Revision 2.0*) and by the system-dependent implementation of PCI initialization device select (IDSEL) encoding. Configuration cycles use one cycle of address stepping; that is, the address and command are driven for two consecutive cycles.

6.2.10 Special Cycles and Interrupt Acknowledge Cycles

The PCI special cycle command implements a simple broadcast mechanism that allows the broadcast of processor status information and sideband signaling. The command is not addressed to a specific device, and the receivers determine whether the message applies to them. Section 6.3.11 describes two supported sideband signals. Also see the *PCI Local Bus Specification, Revision 2.0* for more information.

6.2.10.1 Special Cycle

The CPU can initiate a special cycle transfer on the PCI by executing an STL instruction to the interrupt vector and special cycle register (IOC_IACK_SC, Section 6.4.13). When this register is written, the IOC initiates a PCI cycle with a special cycle bus command. During a special cycle, the IOC generates an unspecified address on the PCI that is ignored by the PCI devices. The write data generated by the STL instruction passes unmodified through the IOC to the PCI during the data phase of the special cycle. The special cycle message is encoded in the write data as described in the *PCI Local Bus Specification, Revision 2.0*.

The special cycle address region is accessed only by STL instructions. This region supports the same write access types and masked address generation as sparse space (Section 6.2.6), with the following exceptions:

- Write access to the IOC_IACK_SC register is enabled when CPU address bits **a<33:32>** = 01 and **a<31:29>** = 101.
- The IOC_HAE register is used to generate PCI address bits **ad<31:27>** under all circumstances.
- A special cycle command (Table 6–2) is issued on the **c_be_l<3:0>** pins, and a PCI interrupt acknowledge cycle follows as a result.

6.2.10.2 Interrupt Acknowledge Cycle

During a PCI interrupt acknowledge cycle, the IOC can acknowledge an interrupt and receive an interrupt vector. This allows external interrupt-arbitration hardware to supply interrupt vectors for PCI devices.

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

- Read access to the IOC_IACK_SC register is enabled when CPU address bits **a<33:32>** = 01 and **a<31:29>** = 101.
- The IOC_HAE register is used to generate PCI address bits **ad<31:27>** under all circumstances.
- An interrupt acknowledge cycle command (Table 6–2) is issued on the **c_be_l<3:0>** pins, and a PCI interrupt acknowledge cycle follows as a result.

Note

Digital recommends that software use only address 1A0000000 to reference the IOC_IACK_SC register. However, in the current chip implementation, CPU address bits <28:0> are not decoded to access the IOC_IACK_SC register, and any address in the range 1A0000000..1BFFFFFE0 will alias to this single register. For implementation convenience, the chip will drive the supplied address onto the PCI bus.

When the IOC_IACK_SC register is read, the IOC initiates a PCI cycle with an interrupt acknowledge bus command. External interrupt-arbitration logic must supply an interrupt vector in response to the interrupt acknowledge bus cycle. The interrupt vector data is returned to the CPU as the IOC_IACK_SC register read data. During an interrupt acknowledge cycle, the IOC generates an unspecified address on the PCI that is ignored by the PCI devices.

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

6.2.11 Back-to-Back Transfers

The IOC always inserts a bus-idle (bus-turnaround) cycle following the completion of a CPU-initiated PCI transfer. If the PCI grant signal **gnt_l** is sampled asserted during the idle cycle, the IOC initiates another transfer in the next PCI cycle without asserting the bus request signal **req_l**. This sequence of two uninterrupted transfers by the IOC is a back-to-back transfer. Because it inserts an idle cycle between back-to-back transfer cycles, the IOC is not capable of fast back-to-back transfers.

6.2.12 PCI Address Parity

For CPU-initiated PCI transactions, the IOC calculates even parity on the address driven on the **ad<31:0>** pins and the cycle command code driven on the **c_be_l<3:0>** pins. It drives the calculated parity on the PCI **par** pin in the next PCI clock cycle. Because the IOC is the source of the address, command code, and parity, the IOC does not detect or report address parity errors during CPU-initiated PCI transactions. PCI peripherals that detect PCI address parity errors report them with the PCI **serr_l** signal.

Note

Because the chip does not provide an **serr_l** signal pin, external logic must report **serr_l** assertions on one of the chip's **irq<2:0>** interrupt signal pins.

6.2.13 PCI Data Parity

For CPU-initiated PCI write transfers, the IOC generates even parity on the write data. The PCI **ad<31:0>** and **c_be_l<3:0>** signals are included in the parity computation.

For CPU-initiated PCI read transfers, the IOC checks for even parity on the read data. Parity is generated and checked for all bytes of the PCI bus (regardless of the byte-enables).

6.2.13.1 PCI Read Data Parity Errors

During a CPU-initiated PCI read data transfer, the IOC samples and calculates even parity on the **ad<31:0>** and **c_be_l<3:0>** signals. The calculated parity is compared to the parity received on the **par** pin in the PCI clock cycle following a valid data transfer (the **irdy_l** and **trdy_l** signals are asserted). A read data parity error is detected when the comparison fails. In such cases, the IOC reports the error to the CPU with an internal interrupt, and logs the error condition as described below. Additionally, the read data returned to the CPU is flagged with an error, which causes an instruction exception.

When bad data parity is detected, the IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 2 (bad data parity) in the IOC_STAT0 register.
- Logs the PCI command code of the CPU-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates that the error occurred during a PCI read cycle.
- Logs the starting address of the CPU-initiated PCI transfer into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set, the IOC sets the LOST bit in the IOC_STAT0 register. The LOST bit remains set until it is cleared by writing a one to it. The CODE and CMD error information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.2.13.2 PCI Write Data Parity Errors

During a CPU-initiated PCI write data transfer, the IOC drives and calculates even parity on the **ad<31:0>** and **c_be_l<3:0>** signals. The IOC drives the calculated parity on the **par** pin in the PCI clock cycle after it drives the write data. PCI devices that support parity checking calculate even parity on received **ad<31:0>** and **c_be_l<3:0>** signals when they are the target of PCI memory, I/O, or configuration write cycles. The target samples the **par** signal one cycle after a valid data transfer (the **irdy_l** and **trdy_l** signals are asserted). A write data parity error is detected when the calculated parity and sampled parity comparison fails. In such cases, the target asserts **perr_l** in the next cycle (two cycles after a valid data transfer). The IOC reports the error to the CPU with an internal interrupt, and logs the error condition as described in Section 6.2.13.1; however, in this case, the CMD field in the IOC_STAT0 register indicates that the error was generated during a PCI write cycle.

6.2.14 PCI Master Timeout

The *PCI Local Bus Specification, Revision 2.0* specifies a mechanism to limit the duration of a bus master's burst sequence. The mechanism requires PCI masters to implement a latency timer that counts the number of cycles since the assertion of the **frame_l** signal. If the master's latency timer expires and the master's **gnt_l** signal has been deasserted, the master must surrender the bus. This mechanism prevents masters from holding bus ownership for extended periods of time, but limits throughput as well as latency.

The IOC will time out in the sixteenth cycle after the **frame_l** signal is asserted.

6.2.15 PCI Target-Disconnects

The *PCI Local Bus Specification, Revision 2.0* requires all PCI bus masters to support target-disconnect cycle terminations. Target-disconnects are signaled with the **stop_l** signal. The IOC will end the current transfer if it detects a target-disconnect termination signal.

If the target-disconnect is not a target-abort and the disconnected transfer was not completed, the IOC attempts to complete the transfer by rearbitrating for PCI mastership and initiating a transfer starting at the address of the next untransferred data. An error is not reported unless the PCI retry limit is exceeded or a target-abort disconnect is detected (Sections 6.2.16 and 6.2.17).

6.2.16 PCI Retry Limit

If the PCI target of a CPU-initiated transfer retries the transfer $2^{24} - 1$ times, a retry-timeout error is generated and the IOC deletes the CPU-initiated transfer from its CPU request queue. If the transaction is a read operation, UNDEFINED data is returned to the CPU with error status, which causes an instruction exception. The IOC also reports the error to the CPU with an internal interrupt, and logs the error condition.

The IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 0 (retry limit) in the IOC_STAT0 register.
- Logs the PCI command code of the CPU-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates the type of PCI cycle in which the error occurred.
- Logs the starting address of the CPU-initiated PCI transfer into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set, the IOC sets the LOST bit in the IOC_STAT0 register. The LOST bit remains set until it is cleared by writing a one to it. The CODE and CMD error information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.2.17 PCI Target-Abort

A PCI target can signal the target-abort termination when a fatal error occurs during a transaction or when the target can never respond to a transaction. When the IOC detects a target-abort termination, it ends the transaction by deasserting the **frame_l** signal and asserting the **irdy_l** signal, and then deasserting the **irdy_l** signal one cycle later. The IOC will not retry this access on the PCI. If the CPU-initiated access was a read operation, the IOC returns arbitrary data with error status to the CPU, which causes an instruction exception. The IOC also reports the error to the CPU with an internal interrupt, and logs the error condition.

The IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 3 (target abort) in the IOC_STAT0 register.
- Logs the PCI command code of the CPU-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The CMD field indicates the type of PCI cycle in which the error occurred.
- Logs the starting address of the CPU-initiated PCI transfer into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set, the IOC sets the LOST bit in the IOC_STAT0 register. The LOST bit remains set until it is cleared by writing a one to it. The CODE and CMD error information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

See the *PCI Local Bus Specification, Revision 2.0* for more information about the target-abort termination.

6.2.18 PCI Master Abort

The IOC terminates a CPU-initiated PCI transaction if no PCI device responds by asserting **devsel_l** within five PCI clock cycles after the IOC asserts the **frame_l** signal. The IOC will not retry this transfer on the PCI. If the CPU-initiated access was a read operation, the IOC returns arbitrary data with error status to the CPU, which causes an instruction exception. The IOC also reports the error to the CPU with an internal interrupt, and logs the error condition.

The IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 1 (no device) in the IOC_STAT0 register, except for CPU-initiated PCI special cycles.
- Logs the PCI command code of the CPU-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates the type of PCI cycle in which the error occurred.
- Logs the starting address of the CPU-initiated PCI transfer into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set, the IOC sets the LOST bit in the IOC_STAT0 register. The LOST bit remains set until it is cleared by writing a one to it. The CODE and CMD error information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3 Peripheral-Initiated PCI Cycles

The IOC is a PCI target when it responds to cycles initiated by a PCI peripheral device or external DMA controller operating as a bus master. The IOC treats PCI-initiated masked read operations as unmasked read operations. Arbitrary byte-enable combinations for PCI-initiated write transfers are implemented by read-modify-write operations in the memory controller. PCI-initiated transfers can target pages of memory that are mapped in cacheable or noncacheable address spaces. The chip ensures that memory coherency is maintained for all PCI-initiated transfers to cacheable memory.

PCI devices cannot access registers that are internal to the chip.

6.3.1 PCI Cycle Command Code Support and Aliasing

Table 6–5 lists all the PCI-initiated cycle commands. The IOC supports only PCI-initiated memory transfers. PCI memory read and memory write cycle commands are supported directly. Other types of PCI memory commands are supported by aliasing them to either the memory read or memory write command. The IOC does not support or respond to other PCI-initiated cycle commands.

Table 6–5 Commands for PCI-Initiated Transactions

c_be_l<3:0>	PCI Command Type*	Support
0000	Interrupt acknowledge	Not supported
0001	Special cycle	Not supported
0010	I/O read	Not supported
0011	I/O write	Not supported
0100	Reserved	Not supported
0101	Reserved	Not supported
0110	Memory read	Directly supported
0111	Memory write	Directly supported
1000	Reserved	Not supported
1001	Reserved	Not supported
1010	Configuration read	Not supported
1011	Configuration write	Not supported
1100	Memory read multiple	Aliased to memory read command
1101	Dual address cycle	Not supported
1110	Memory read line	Aliased to memory read command
1111	Memory write and invalidate	Aliased to memory write command

*Reserved command types are reserved in the *PCI Local Bus Specification, Revision 2.0*.

6.3.2 PCI Address Translation

Because the PCI uses a 32-bit address and the CPU uses a 34-bit address, PCI addresses to which the IOC responds must be translated to an equivalent address in the CPU address space. The IOC implements two programmable address windows, called PCI target windows, that control PCI peripheral access to system memory. A set of three IOC registers is associated with each PCI target window: the window base; window mask; and translated base registers (IOC_W_BASE<1:0>, Section 6.4.9; IOC_W_MASK<1:0>, Section 6.4.10; and IOC_T_BASE<1:0>, Section 6.4.11).

The window mask register IOC_W_MASK provides a mask corresponding to bits <31:20> of an incoming PCI address. The size of each window is programmed to be from 1 MB to 4 GB by masking bits of the incoming PCI address with the IOC_W_MASK register.

Table 6–6 shows the supported values of IOC_W_MASK register bits <31:20>; other values are unsupported.

Table 6–6 PCI Window Mask

IOC_W_MASK <31:20>	Window Size
0000 0000 0000	1 MB
0000 0000 0001	2 MB
0000 0000 0011	4 MB
0000 0000 0111	8 MB
0000 0000 1111	16 MB
0000 0001 1111	32 MB
0000 0011 1111	64 MB
0000 0111 1111	128 MB
0000 1111 1111	256 MB
0001 1111 1111	512 MB
0011 1111 1111	1 GB
0111 1111 1111	2 GB
1111 1111 1111	4 GB

IOC_W_BASE register bits <31:20> specify the target window starting address in the PCI memory address space. The IOC_W_BASE and IOC_W_MASK registers determine when a PCI transaction references an address in a target window, as follows:

1. Each of the incoming PCI address bits <31:20> is exclusive-ORed with the corresponding bit of each window base register (IOC_W_BASE1, IOC_W_BASE0, bits <31:20>).
2. The result of each exclusive-OR is then ANDed with the complement of the corresponding bits of the corresponding window mask register (IOC_W_MASK1, IOC_W_MASK0, bits <31:20>).
3. The result of the AND operation for each window is then ORed to produce a hit (match) or miss indication. If the result of the final OR is a zero for a given window, the incoming address has hit in that PCI target window; otherwise, it has missed in that window.

Note

The window base address must be on a naturally aligned address boundary equal to the window size (Table 6–6). Additionally, to ensure that only one PCI target window can match an incoming PCI address, the PCI target windows must be programmed so that they do not respond to overlapping PCI address ranges.

Each target window is independently enabled when the window enable bit (WEN) is set in the corresponding IOC_W_BASE register. The IOC responds only to PCI-initiated transfers that hit in an enabled window. If the WEN bit is cleared, the IOC ignores PCI-initiated transfers that hit in that window.

When a hit occurs in an enabled PCI target window, the IOC responds to the PCI cycle by asserting the **devsel_l** signal. The IOC translates the 32-bit PCI address to a 34-bit CPU address. The scatter-gather (SG) bit in the corresponding IOC_W_BASE register determines how the address is translated. If the SG bit is clear, direct-mapped address translation is used (Section 6.3.2.1). If the SG bit is set, scatter-gather mapped address translation is used (Section 6.3.2.2).

6.3.2.1 Direct-Mapped Address Translation

Direct-mapped address translation is used when an incoming PCI address hits in a PCI target window and the SG bit is not set in the corresponding IOC_W_BASE register. The translated address is generated by concatenating bits from the corresponding IOC_T_BASE register with bits from the incoming PCI address. Bits <33:32> of the translated address are always zero. The IOC_W_MASK register determines which IOC_T_BASE register bits and PCI address bits are concatenated to form the translated address, as shown in Table 6–7. The translated address is the starting address in system memory for the PCI transaction.

Caution

Unused bits of the translated base register must be clear (zero) for correct operation.

Table 6–7 PCI Address Translation—Scatter-Gather Mapping Disabled

IOC_W_MASK	Translated Address <31:0>		Unused
<31:20>	IOC_T_BASE	PCI Address	IOC_T_BASE
0000 0000 0000	<31:20>	<19:0>	<19:9>
0000 0000 0001	<31:21>	<20:0>	<20:9>
0000 0000 0011	<31:22>	<21:0>	<21:9>
0000 0000 0111	<31:23>	<22:0>	<22:9>
0000 0000 1111	<31:24>	<23:0>	<23:9>
0000 0001 1111	<31:25>	<24:0>	<24:9>
0000 0011 1111	<31:26>	<25:0>	<25:9>
0000 0111 1111	<31:27>	<26:0>	<26:9>
0000 1111 1111	<31:28>	<27:0>	<27:9>
0001 1111 1111	<31:29>	<28:0>	<28:9>
0011 1111 1111	<31:30>	<29:0>	<29:9>
0111 1111 1111	<31>	<30:0>	<30:9>
1111 1111 1111	—	<31:0>	<31:9>

6.3.2.2 Scatter-Gather Mapped Address Translation

Scatter-gather mapped address translation is used when an incoming PCI address hits in a PCI target window and the SG bit is set in the corresponding IOC_W_BASE register. The translated address is generated by table lookup. The table is called a scatter-gather map and is stored in system memory. The IOC combines the incoming PCI address with the corresponding IOC_T_BASE register to generate the index address into the scatter-gather map. The IOC_T_BASE register specifies the starting address of the scatter-gather map, and the incoming PCI address bits are used as an offset from that address.

The size of the PCI target window, as defined by the IOC_W_MASK register, determines the length of the scatter-gather map, as shown in Table 6–8. Each scatter-gather map entry maps an 8-KB page of the PCI address space into an 8-KB page of the CPU address space.

The IOC_W_MASK register determines which IOC_T_BASE register bits and PCI address bits generate the address of the scatter-gather map entry, as shown in Table 6–8. Translated address bits <33:32> are always zero, and because each scatter-gather map entry is a quadword (64 bits), the three LSBs (<2:0>) of a scatter-gather map address are also always zero. The resultant address is used to read an entry from the scatter-gather map in system memory.

Caution

Unused IOC_T_BASE register bits must be clear (zero) for correct operation.

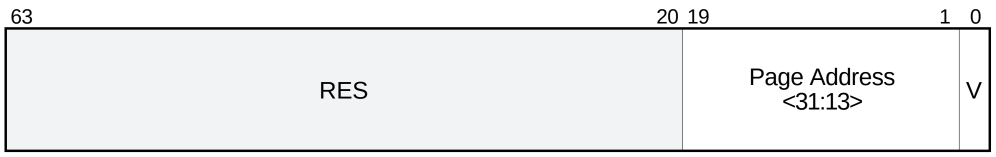
Table 6–8 Scatter-Gather Map Address

IOC_W_MASK	Scatter-Gather	Translated Address <31:0>			Unused
<31:20>	Map Size	IOC_T_BASE	PCI Address	<2:0>	IOC_T_BASE
0000 0000 0000	1 KB	<31:10>	<19:13>	000	None
0000 0000 0001	2 KB	<31:11>	<20:13>	000	<10>
0000 0000 0011	4 KB	<31:12>	<21:13>	000	<11:10>
0000 0000 0111	8 KB	<31:13>	<22:13>	000	<12:10>
0000 0000 1111	26 KB	<31:14>	<23:13>	000	<13:10>
0000 0001 1111	32 KB	<31:15>	<24:13>	000	<14:10>
0000 0011 1111	64 KB	<31:16>	<25:13>	000	<15:10>
0000 0111 1111	128 KB	<31:17>	<26:13>	000	<16:10>
0000 1111 1111	256 KB	<31:18>	<27:13>	000	<17:10>
0001 1111 1111	512 KB	<31:19>	<28:13>	000	<18:10>
0011 1111 1111	1 MB	<31:20>	<29:13>	000	<19:10>
0111 1111 1111	2 MB	<31:21>	<30:13>	000	<20:10>
1111 1111 1111	4 MB	<31:22>	<31:13>	000	<21:10>

The scatter-gather map entry read from system memory specifies a page address in the 34-bit CPU address space. Because the page size is 8 KB and translated address bits <33:32> are always zero, the page table entry (PTE) specifies only address bits <31:13>.

Figure 6–1 shows the scatter-gather map PTE format. The scatter-gather map PTE includes a valid bit (V). If the V bit is clear, the scatter-gather map PTE is not valid. When the IOC encounters an invalid entry during a translation, it logs the error condition and reports the error to the CPU with an internal interrupt. The IOC also terminates the PCI transfer (Section 6.3.5).

Figure 6–1 Scatter-Gather Map Page Table Entry



The page address specified by the scatter-gather map entry is used to translate the incoming 32-bit PCI address to an equivalent 34-bit CPU address. To generate the final translated address, bits <31:13> (the page address) are read from the scatter-gather map PTE and concatenated with bits <12:0> of the incoming PCI address. Address bits <33:32> are always zero. The IOC uses this 34-bit translated address as the physical address to access a system memory location and complete the peripheral-initiated PCI cycle. If the memory controller does not decode the translated address, the IOC logs an error condition and reports the error to the CPU with an internal interrupt. The IOC also terminates the PCI transfer, by using the target-abort protocol (Section 6.3.8.2).

6.3.3 Scatter-Gather Map Translation Buffer

The IOC implements an eight-entry translation lookaside buffer (TLB) for scatter-gather map entries. The TLB is a fully associative cache. Each entry in the TLB stores a PCI page address (cache tag) and the corresponding CPU page address (cache data) from the eight most recently used scatter-gather map entries. Each time an incoming PCI address selects a PCI target window with scatter-gather mapping enabled, bits <31:13> of the PCI address are compared with the PCI page addresses (cache tag) in the TLB. If a match is found, the CPU page address is in the data of that TLB entry. This makes it unnecessary to look in the scatter-gather map and reduces latency to increase performance.

If a match is not found in the TLB, the scatter-gather map is referenced (Section 6.3.2.2). The data read from the scatter-gather map and the incoming PCI page address are written over an existing TLB entry. TLB entries are replaced (overwritten) according to a round-robin algorithm.

All entries in the TLB can be flushed by writing to the translation buffer invalidate all register (IOC_TBIA, Section 6.4.5). Writing to the IOC_TBIA register invalidates all TLB entries at the start of the next PCI transaction. The TLB is also flushed if a page table read error occurs.

If the TLB enable (TEN) bit in the translation buffer enable register (IOC_TB_ENA, Section 6.4.6) is clear, all entries are invalidated at the start of every subsequent PCI bus cycle (during address decoding by the IOC). Clearing the TEN bit does not disable scatter-gather mapped translations. If a PCI-initiated transaction requires scatter-gather map translation while the TEN bit is clear, the IOC reads the required TLB entry from system memory and temporarily stores the entry in the TLB for the current transaction. This entry is invalidated at the start of any subsequent PCI transaction if the TEN bit is not set.

6.3.4 Page Table Read Errors

A page table read error is reported if any of the following occurs:

- A scatter-gather map entry references a nonexistent address.
- The read data has an uncorrectable error.
- A cache tag parity error is detected when reading cached data.

The IOC reports the error to the CPU with an internal interrupt and logs the error information. If the error occurs on a PCI-initiated read transfer, the IOC terminates the PCI transaction using the target-abort protocol. If the error occurs on a PCI-initiated write transfer, the IOC completes the bus cycle but does not transfer the write data to memory.

When a page table read error occurs, the IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3)
- Sets the error CODE field to 5 (page table read error) in the IOC_STAT0 register
- Logs starting address bits <31:13> of the incoming PCI transfer into the P_NBR field of the IOC_STAT0 register

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set prior to this transaction, the IOC_STAT0 register LOST bit is set. The LOST bit remains set until it is cleared by writing a one to it. The P_NBR and CODE information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3.5 Invalid Page Errors

If the page table entry read from the scatter-gather map in system memory is invalid, the IOC reports an error to the CPU with an internal interrupt and logs the error information. If the error occurs on a PCI-initiated read transfer, the IOC terminates the PCI transaction by using the target-abort protocol. If the error occurs on a PCI-initiated write transfer, the IOC completes the bus cycle but does not transfer the write data to memory.

When an invalid page error occurs, the IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3)
- Sets the error CODE field to 6 (invalid page error) in the IOC_STAT0 register
- Logs starting address bits <31:13> of the incoming PCI transfer into the P_NBR field of the IOC_STAT0 register

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set prior to this transaction, the IOC_STAT0 register LOST bit is set. The LOST bit remains set until it is cleared by writing a one to it. The P_NBR and CODE information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3.6 Address Phase

When the IOC detects the assertion of the **frame_l** signal by an external PCI master, it samples the address on the **ad<31:0>** pins and the PCI command code on the **c_be_l<3:0>** pins. During the next PCI clock cycle, the IOC decodes the address to determine whether the referenced address is in a PCI target window and samples the address phase parity on the **par** pin. If the address hits in a PCI target window, the IOC asserts the **devsel_l** signal in the next PCI clock cycle (two cycles after the assertion of the **frame_l** signal) to indicate to the PCI master that the IOC will respond to the PCI transaction.

6.3.6.1 PCI Address Queue

The two-entry PCI address queue can hold address and command information for up to two PCI-initiated transactions. When the IOC samples an address, it inserts the address and command code in its PCI address queue. When a transaction completes, the associated address queue entry is cleared.

Because the IOC buffers write data for PCI-initiated write transactions, a write transaction could complete on the PCI bus while the IOC PCI write-data queue (Section 6.3.7.1) holds write data waiting to be transferred to memory. In such cases, the address and command information for a subsequent PCI-initiated transaction is stored in the second entry of the IOC PCI address queue. The second transaction will be stalled (the **trdy_l** signal will not be asserted) until the buffered write transaction is completed.

6.3.6.2 Address Burst Order

The IOC supports only a linearly incrementing burst order. The initiator of a PCI transaction specifies a linearly incrementing burst order by setting incoming PCI address bits <1:0> to 00. The IOC allows a burst to continue until one of the following occurs:

- A page boundary is crossed (Section 6.3.6.3).
- The target latency timeout has been exceeded (Section 6.3.9).
- An error occurs.

If incoming PCI address bits <1:0> are not 00, the initiator of the PCI transaction did not specify a linearly incrementing order. In such cases, the IOC completes only one data transfer. If the initiator requests a transfer with a burst length greater than one, the IOC terminates the transfer using the PCI target-disconnect protocol (Section 6.2.15). This forces the initiator to break the burst transfer into multiple, single transfer cycles. Each single transfer cycle specifies a new address, consistent with the burst order specified by the initiator.

6.3.6.3 Page Boundary Crossing

If a PCI write burst crosses an 8-KB page boundary, the IOC will stop the burst by using the target-disconnect protocol and by asserting the **stop_l** signal. This forces PCI devices to break transfers at 8-KB page boundaries. The IOC does not assert the PCI **stop_l** signal until the cycle after it asserts the **trdy_l** signal that completes the transfer of the last longword address within the current 8-KB page. An error is not generated when an 8-KB page boundary is reached.

6.3.6.4 Address Parity Errors

If the IOC detects an address parity error during a PCI-initiated transaction, it reports the error to the CPU with an internal interrupt and logs the error condition. The IOC completes the PCI transaction as follows: If the error occurred on a PCI-initiated read transfer, the IOC returns the requested read data from memory. If the error occurred on a PCI-initiated write transfer, the IOC does not write the data to system memory (the data is lost).

When an address parity error occurs, the IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 4 (address parity error) in the IOC_STAT0 register.
- Logs the PCI command code of the incoming PCI-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates the type of PCI cycle in which the error occurred.
- Logs the starting address of the PCI transfer into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set prior to this transaction, the IOC_STAT0 register LOST bit is set. The LOST bit remains set until it is cleared by writing a one to it. The CMD and CODE information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3.7 PCI-Initiated Write Data Transfers

On a PCI-initiated write transfer, the IOC transfers the write data it receives from the PCI bus to system memory through the memory controller. Because the chip uses a quadword memory organization, read-modify-write cycles must be used to write less than a full quadword to memory.

In a given PCI-initiated bus cycle, if the first longword of write data that the IOC receives is for an odd longword address, the memory controller performs a read-modify-write operation to write the longword to the destination address.

If the IOC receives a longword of write data that is not the last longword of the bus cycle and is for an even longword address, the IOC waits until it receives the next longword before issuing a write transfer to the destination address. In this case, the IOC packs both longwords into a single quadword and writes the data to the destination address with a single quadword write transfer. If either longword is masked, the memory controller performs a read-modify-write operation to the addressed location.

If the last longword received is write data to an even longword address, the memory controller uses a read-modify-write operation to complete the longword transfer.

6.3.7.1 PCI Write-Data Queue

The four-entry PCI write-data queue holds incoming data during PCI-initiated memory write cycles. Each queue entry holds one quadword of write data. The PCI write-data queue is not used during PCI-initiated memory read cycles.

If a PCI-initiated write-burst transfer begins on an odd longword address boundary, only one longword of write data is stored in the upper longword of the first queue entry. After the first data transfer of an odd longword-aligned transfer or if the burst begins on a quadword-aligned boundary, the IOC packs two longwords of write data into each PCI write-data queue entry. If the burst ends on an even longword address, only one longword of data is stored in the low longword of the last queue entry.

After each PCI write-data queue entry is successfully written to system memory, the entry is reused to store subsequent write-burst data.

6.3.7.2 PCI-Initiated Memory Write Data Errors

The IOC does not report Bcache tag parity errors, nonexistent memory errors, or uncorrectable read errors that occur during a memory operation resulting from a PCI-initiated memory write transfer. The memory controller reports such errors to the CPU.

During PCI-initiated memory write data transfers, the IOC calculates even parity for the **ad<31:0>** and **c_be_l<3:0>** signals sampled by the IOC. The calculated parity is compared with the parity received on the **par** pin in the next PCI clock cycle. If the comparison fails, a write data parity error was detected. The IOC asserts the **perr_l** signal during the next PCI clock cycle to inform the initiator that a write data parity error was detected. The write data with the parity error is written to memory. The IOC reports the error to the CPU with an internal interrupt and logs the error condition.

The IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 7 (data error) in the IOC_STAT0 register.
- Logs the PCI command code of the incoming PCI-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates that the error occurred during a PCI write cycle.
- Logs the PCI address of the errant longword into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set prior to this transaction, the IOC_STAT0 register LOST bit is set. The LOST bit remains set until it is cleared by writing a one to it. The CMD and CODE information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3.8 PCI-Initiated Read Data Transfers

The IOC prefetches read data to increase the performance of PCI-initiated burst read transfers. After the address of a PCI-initiated read transfer is loaded into the PCI address queue (Section 6.3.6.1), the IOC begins to prefetch quadwords from sequential memory locations.

If the PCI master attempts to continue a burst across an 8-KB page boundary, the IOC terminates the transaction using the target-disconnect protocol when the last longword in the 8-KB page is transferred. When the IOC prefetch address counter reaches the last longword of an 8-KB boundary, it wraps around to the starting address of the same 8-KB page, and the IOC continues to prefetch data. However, the prefetched data is flushed from the IOC read-data queue when the IOC terminates the PCI read transaction by using the target-disconnect protocol.

The returned memory read data and associated error status is stored in the IOC's PCI read-data queue (Section 6.3.8.1). The IOC returns the requested longwords of read data to the PCI initiator from the read-data queue.

Because the IOC prefetches read data, the PCI initiator can complete its transfer while the IOC is waiting for prefetch data to be returned from memory. If the IOC responds to another PCI transaction while prefetch read transfers from the previous transaction are pending, the IOC buffers the new address in the PCI address queue. The IOC does not issue any new memory requests until all the prefetch data requested during the previous transaction is received and flushed from the read-data queue.

6.3.8.1 PCI Read-Data Queue

The four-entry PCI read-data queue stores read data for PCI-initiated read transfers. Each entry contains a naturally aligned quadword. The queue allows the IOC to prefetch read data (Section 6.3.8). The queue is used for only one bus cycle at a time and is flushed when the current PCI-initiated read transfer is completed.

6.3.8.2 PCI-Initiated Memory Read Data Errors

Bcache tag parity errors that occur during PCI-initiated read transfers are reported to the CPU by the memory controller using an internal interrupt; however, the IOC might not be notified and, in that case, complete the transfer as normal. If the IOC is notified, it handles the error in the same way as nonexistent memory errors and uncorrectable read errors.

Nonexistent memory errors and uncorrectable read errors that occur during PCI-initiated read transfers are also reported to the CPU by the memory controller by using an internal interrupt.

If the read data is driven on the PCI bus (that is, the transaction is not an unused prefetch), the associated error status bit is checked. If the error bit is set, the IOC will use the target-abort protocol (Section 6.2.17) to terminate the PCI read transfer. The IOC will log the error.

The IOC does the following:

- Sets the ERR bit (if it is not set) in the IOC status 0 register (IOC_STAT0, Section 6.4.3).
- Sets the error CODE field to 7 (data error) in the IOC_STAT0 register.
- Logs the PCI command code of the incoming PCI-initiated transfer in which the error was detected into the CMD field of the IOC_STAT0 register. The command code indicates that the error occurred during a PCI read cycle.
- Logs the PCI address of the errant longword into the ADDR field of the IOC status 1 register (IOC_STAT1, Section 6.4.4).

The ERR bit and the logged error information are frozen until the ERR bit is cleared by writing a one to it.

If the ERR bit was set prior to this transaction, the IOC_STAT0 register LOST bit is set. The LOST bit remains set until it is cleared by writing a one to it. The CMD and CODE information is logged only for the error that caused the ERR bit to be set—such information for the second error is lost.

6.3.9 PCI Target Latency Timeout

If a memory operation introduces more than eight PCI cycles of latency since the IOC last asserted the **trdy_l** signal for a PCI-initiated memory transfer, the IOC terminates the transfer by using the target-disconnect protocol and assert the **stop_l** signal in the next PCI cycle. Data could be transferred in the same bus cycle in which the IOC signals a disconnect. This will occur if the master is asserting the **irdy_l** signal and the IOC state changes such that it asserts the **trdy_l** signal in the same cycle that it asserts the **stop_l** signal.

6.3.10 PCI-Initiated Exclusive Access

The IOC supports the PCI exclusive-access protocol for PCI-initiated transfers that access system memory through the IOC and memory controller. The IOC considers an access to be an exclusive access and marks itself locked when the following sequence of events occur and the stated conditions are met during a PCI transfer.

1. The **lock_l** signal is sampled deasserted in the same cycle that the **frame_l** signal is asserted.
2. The PCI-initiated cycle addresses a location in an enabled PCI target window.
3. The IOC samples the **lock_l** signal asserted in a cycle that is subsequent to the assertion of the **frame_l** signal but prior to the cycle in which the IOC asserts the **trdy_l** signal.

While the IOC is locked, the internal CPU lock flag is held reset, preventing the completion of store conditional instructions. The IOC maintains the locked status until it samples the **frame_l** and **lock_l** signals deasserted.

While the IOC is locked, it continues to sample the **lock_l** signal in the same cycle that the **frame_l** signal is asserted during subsequent PCI-initiated transfers as follows:

- If the **lock_l** signal is sampled asserted, the transfer is a nonexclusive access. If this nonexclusive cycle addresses a location in an enabled PCI target window, the IOC uses the PCI target-disconnect protocol to signal a retry, and no data is transferred.
- If the **lock_l** signal is sampled deasserted, the transfer is a continuation of the exclusive access. If this exclusive access cycle addresses a location in an enabled PCI target window, the IOC completes the access normally (the IOC does not retry the access).

6.3.11 Guaranteed Access Arbitration

In some applications, to minimize access latency, a PCI device may need to win arbitration for memory before initiating a PCI cycle. The IOC supports two PCI sideband signals for memory arbitration, **memreq_l** and **memack_l**. Both signals are synchronous to the PCI clock signal **pci_clk_in**.

A device wins arbitration for memory according to the following sequence:

1. The device must assert the **memreq_l** signal to the IOC.
2. When the IOC samples the **memreq_l** signal asserted, it arbitrates (internal to the chip) for access to memory.

3. When the IOC wins arbitration for memory, it asserts the **memack_l** signal. While **memack_l** is asserted, the CPU is blocked from accessing memory.
4. When the requesting device samples the **memack_l** signal asserted (and if it has won arbitration for the PCI bus), it can initiate a PCI cycle without any intervening CPU access or additional latency.

The **memack_l** signal remains asserted for one or more cycles after the device deasserts the **memreq_l** signal. The number of cycles depends on the ratio of the internal clock frequency to the PCI clock frequency.

To avoid buffering a large number of write operations, the IOC allows only one longword per DMA-write transaction while **memack_l** is asserted. After the first longword, the IOC terminates the transaction by using the target disconnect protocol.

When the internal clock is 166 MHz and the PCI clock is 33 MHz, and if **memack_l** is asserted before **frame_l** is asserted, the delay from the assertion of **frame_l** to the assertion of **trdy_l** is less than 2 µs.

6.4 I/O Controller Registers

The IOC implements a number of registers to control its operation and facilitate its testing. The CPU can access the IOC registers directly, but they are not accessible from the PCI bus.

All references to the IOC registers must allow for quadword access; longword access is not supported.

The IOC registers are described in ascending-address order in Sections 6.4.1 through 6.4.14. Unless stated otherwise, the assertion of **reset_in_l** does not modify any IOC register bits and all IOC register bits are UNDEFINED after a power-up reset.

Note

In the register field description tables, the abbreviations in the Type column indicate field access behavior. The abbreviations are defined in the Conventions section of the Preface.

6.4.1 Host Address Extension Register

In the sparse memory space, the lower bits of the CPU address indicate transfer size, and the address is shifted down by 5 bits during the translation from CPU address to PCI address. The host address extension (IOC_HAE) register extends the shifted address to 32 bits.

The IOC_HAE register is accessed at address 180000000_{16}. All bits of this register are UNDEFINED after reset.

Figure 6–2 shows the IOC_HAE register format, and Table 6–9 describes its fields.

Figure 6–2 IOC_HAE Register Format

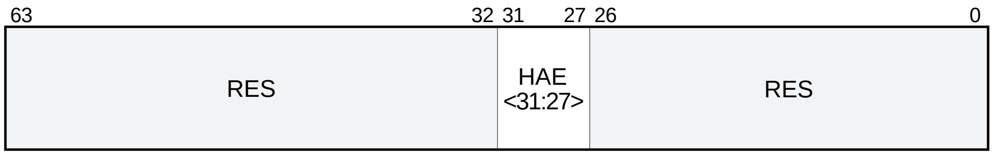


Table 6–9 IOC_HAE Register Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:27	HAE<31:27>	WO	Host address extension—This field is used for PCI address bits **ad<31:27>** during CPU-initiated PCI transactions when CPU physical address bits <31:29> are non-zero (see Table 6–1).
26:0	RES	X	Reserved.

6.4.2 Configuration Cycle Type Register

During CPU-initiated PCI configuration space transfers, the configuration cycle type (IOC_CONF) register supplies PCI address bits **ad<1:0>**.

The IOC_CONF register is accessed at address 180000020_{16}. All bits of this register are UNDEFINED after reset.

Figure 6–3 shows the IOC_CONF register format, and Table 6–10 describes its fields.

Figure 6–3 IOC_CONF Register Format

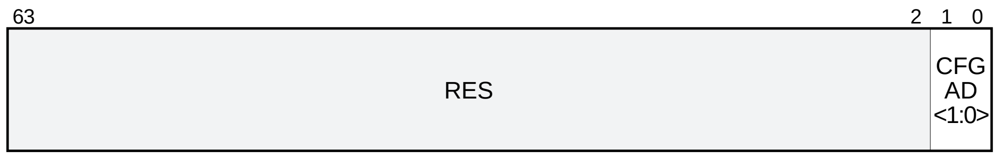


Table 6–10 IOC_CONF Register Field Description

Bits	Field	Type	Description
63:2	RES	X	Reserved.
1:0	CFG AD<1:0>	WO	Configuration address— This field is used as PCI address bits **ad<1:0>** during CPU-initiated PCI configuration cycles.

6.4.3 Status 0 Register

The IOC logs error information in the status 0 (IOC_STAT0) register as follows:

- If the ERR bit (<4>) is not set when an IOC error is detected, all of the register bits are updated and the ERR bit is set.
- After the ERR bit is set, all of the register bits except the TREF, THIT, and LOST bits (<7:5>) are frozen until software clears the ERR bit.
- If the ERR bit is set, and another IOC error is detected, the LOST bit is set. Because the ERR bit is set, information associated with the second error cannot be logged and is lost. The LOST bit remains set until it is cleared by software.

The IOC_STAT0 register is accessed at address 180000040_{16}. All bits of this register are UNDEFINED after reset.

Figure 6–4 shows the IOC_STAT0 register format, and Table 6–11 describes its fields.

Figure 6–4 IOC_STAT0 Register Format

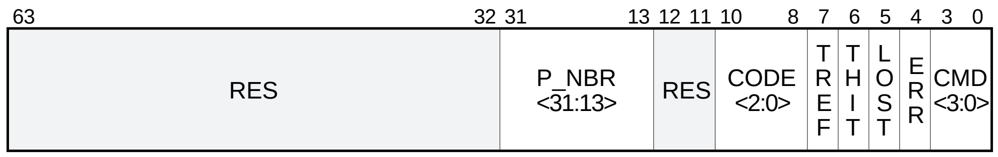


Table 6–11 IOC_STAT0 Register Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:13	P_NBR <31:13>	RO	Page number—This field stores starting address bits **ad<31:13>** of a PCI-initiated transfer during which a page table read error or invalid page table error was detected.
12:11	RES	X	Reserved.
10:8	CODE <2:0>	RO	Error code—This field contains the error code for the error condition that caused the ERR bit (<4>) to be set. This field is encoded as follows:

Code	Error Condition
CPU-Initiated Transfers	
000	Retry limit
001	No device
010	Bad data parity
011	Target abort
PCI-Initiated Transfers	
100	Bad address parity
101	Page table read error
110	Invalid page
111	Data error

Bits	Field	Type	Description
7	TREF	RO	Test reference—This bit is for diagnostic testing. This bit is set if the last PCI transaction referenced an address in a target window. This bit is clear if the last PCI transaction referenced an address outside of the address ranges defined by the target windows.

(continued on next page)

Table 6–11 (Cont.) IOC_STAT0 Register Field Description

Bits	Field	Type	Description
6	THIT	RO	Test hit—This bit is for diagnostic testing. This bit is set if the last PCI transaction that referenced an address in a target window also hit in the translation lookaside buffer (TLB, Section 6.3.3). This bit is clear if the last PCI transaction that referenced an address in a target window missed in the TLB.
5	LOST	R/W1C	This bit is set when an error occurs and either of the following conditions applies: • The ERR bit (<4>) was set before the error occurred. • The CODE field (<2:0>) value is 5 or 6 and the ERR bit was set at the beginning of the transaction that caused the error. When set, this bit indicates that error-logging information associated with the new error was lost. This bit remains set until it is written with a one.
4	ERR	R/W1C	Error—This bit is set when the IOC logs an error. This bit remains set until it is written with a one.
3:0	CMD <3:0>	RO	Command—When the IOC detects an error, this field stores the PCI command code of either of the following: • A CPU-initiated PCI transaction during which the error occurred. • A PCI-initiated transaction during which the error, other than a page table read error or invalid page table error, occurred (see bits <31:13>).

6.4.4 Status 1 Register

The status 1 (IOC_STAT1) register stores the starting address of PCI transactions during which the IOC detected an error other than a page table read error or invalid page table error.

The IOC_STAT1 register is accessed at address 180000060_{16}. All bits of this register are UNDEFINED after reset.

Figure 6–5 shows the IOC_STAT1 register format, and Table 6–12 describes its fields.

Figure 6–5 IOC_STAT1 Register Format

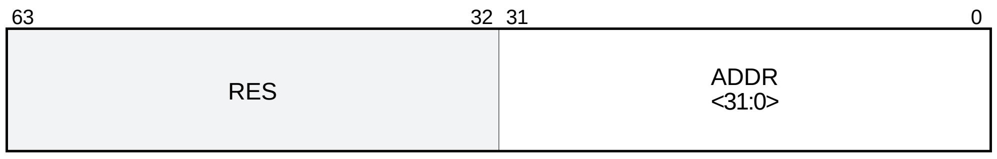


Table 6–12 IOC_STAT1 Register Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:0	ADDR <31:0>	RO	Address—When the IOC detects an error, this field stores either of the following addresses (**ad<31:0>**): • If the error occurred during a CPU-initiated dense-space burst, this field contains the address of the first longword transferred during the current bus cycle. • If the error is a PCI-initiated data error (IOC_STAT0 bits <10:8> = 111), this field contains the PCI address of the errant longword.

6.4.5 Translation Buffer Invalidate All Register

A write operation to the translation buffer invalidate all (IOC_TBIA) register invalidates all entries in the scatter-gather map TLB at the start of the next PCI-initiated transaction (Section 6.3.3).

The IOC_TBIA register is accessed at address 180000080_{16}. All bits of this register are reserved and are ignored on write operations and UNDEFINED on read operations.

6.4.6 Translation Buffer Enable Register

The translation buffer enable (IOC_TB_ENA) register determines whether the scatter-gather map TLB is enabled (Section 6.3.3).

The IOC_TB_ENA register is accessed at address 1800000A0. All bits of this register are UNDEFINED after reset.

Figure 6–6 shows the IOC_TB_ENA register format, and Table 6–13 describes its fields.

Figure 6–6 IOC_TB_ENA Register Format

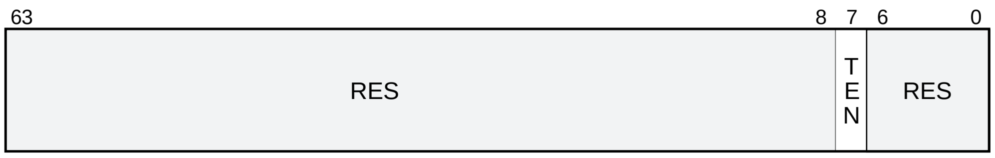


Table 6–13 IOC_TB_ENA Register Field Description

Bits	Field	Type	Description
63:8	RES	X	Reserved.
7	TEN	WO	TLB enable—When written with a zero, this bit disables the scatter-gather map TLB. The TLB remains disabled until this bit is written with a one. Disabling the TLB does not disable scatter-gather map translations.
6:0	RES	X	Reserved.

6.4.7 PCI Soft Reset Register

The PCI soft reset (IOC_SFT_RST) register allows software to assert and deassert the PCI reset signal **rst_l**. The required time that the **rst_l** signal is asserted is system dependent and must be timed by software.

The IOC_SFT_RST register is accessed at address 1800000C0. The RST bit (<6>) will be set after reset.

Figure 6–7 shows the IOC_SFT_RST register format, and Table 6–14 describes its fields.

Figure 6–7 IOC_SFT_RST Register Format

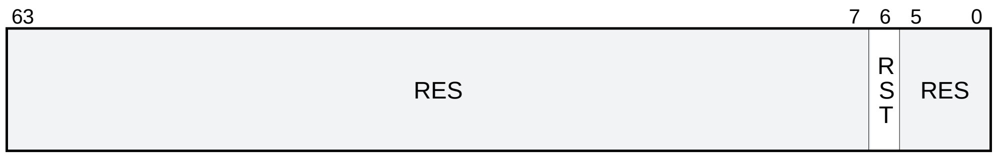


Table 6–14 IOC_SFT_RST Register Field Description

Bits	Field	Type	Description
63:7	RES	X	Reserved.
6	RST	WO	Reset—When this bit is written with a one, the IOC asserts the PCI reset signal **rst_l**. The **rst_l** signal remains asserted until this bit is written with a zero. This bit is also set and the **rst_l** signal is asserted when the chip reset signal **reset_in_l** is asserted.
5:0	RES	X	Reserved.

6.4.8 PCI Parity Disable Register

The PCI parity disable (IOC_PAR_DIS) register determines whether IOC parity checking and associated error reporting and logging are enabled.

The IOC_PAR_DIS register is accessed at address 1800000E0. The PAR bit (<5>) is cleared after reset.

Figure 6–8 shows the IOC_PAR_DIS register format, and Table 6–15 describes its fields.

Figure 6–8 IOC_PAR_DIS Register Format

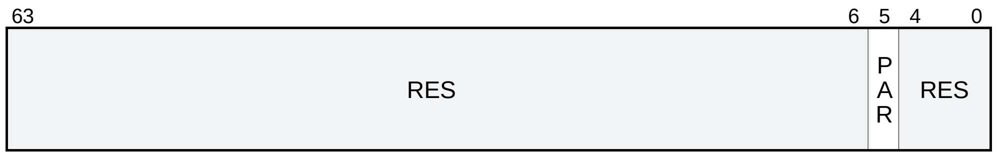


Table 6–15 IOC_PAR_DIS Register Field Description

Bits	Field	Type	Description
63:6	RES	X	Reserved.
5	PAR	WO	Parity—When set, this bit disables IOC parity checking (comparing calculated parity to the **par** signal) as well as the reporting and logging of parity errors detected by the parity check. In CPU-initiated PCI write transactions, the PCI target device reports parity errors on the **perr_l** signal, and the IOC logs a bad data parity error (CODE = 2) in the IOC_STAT0 register (Section 6.4.3). This bit does *not* disable such reporting. This bit is initialized to zero.
4:0	RES	X	Reserved.

6.4.9 Window Base Registers

There is one window base (IOC_W_BASE) register for each PCI target window. These registers (IOC_W_BASE1, IOC_W_BASE0) specify the starting addresses of the PCI target windows in PCI memory address space. The window base addresses must be on a naturally aligned address boundary equal to the window's size. Additionally, to ensure that only one PCI target window can match an incoming PCI address, the PCI target windows must be programmed so that they do not respond to overlapping PCI address ranges. (See Section 6.3.2 for more information about using these registers.)

Software should ensure that the PCI is idle before updating these registers.

The IOC_W_BASE0 register is accessed at address 180000100_{16} and the IOC_W_BASE1 register is accessed at address 180000120_{16}. All bits of these registers are UNDEFINED after reset.

Figure 6–9 shows the IOC_W_BASE register format, and Table 6–16 describes its fields.

Figure 6–9 IOC_W_BASE1–0 Registers Format

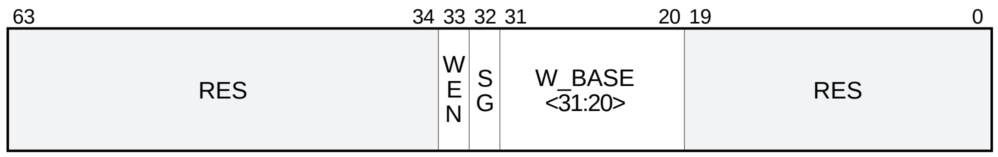


Table 6–16 IOC_W_BASE1–0 Registers Field Description

Bits	Field	Type	Description
63:34	RES	X	Reserved.
33	WEN	WO	Window enable—When this bit is set, the PCI target window is enabled and responds to PCI-initiated transfers that hit in the address range of the target window. When this bit is clear, the PCI target window is disabled and does not respond to PCI-initiated transfers.
32	SG	WO	Scatter-gather—When this bit is set, the PCI target window uses scatter-gather mapping to translate a PCI address to a CPU address. When this bit is clear, the PCI target window uses direct mapping to translate a PCI address to a CPU address.
31:20	W_BASE <31:20>	WO	Window base—This field specifies the starting address of the PCI target window in PCI memory address space.
19:0	RES	X	Reserved.

6.4.10 Window Mask Registers

There is one window mask (IOC_W_MASK) register for each PCI target window. These registers (IOC_W_MASK1, IOC_W_MASK0) provide a mask that corresponds to incoming PCI address bits <31:20>. The mask determines the size of the PCI target window. (See Section 6.3.2 for more information about using these registers.)

Software should ensure that the PCI is idle before updating these registers.

The IOC_W_MASK0 register is accessed at address 180000140_{16} and the IOC_W_MASK1 register is accessed at address 180000160_{16}. All bits of these registers are UNDEFINED after reset.

Figure 6–10 shows the IOC_W_MASK register format, and Table 6–17 describes its fields.

Figure 6–10 IOC_W_MASK1–0 Registers Format

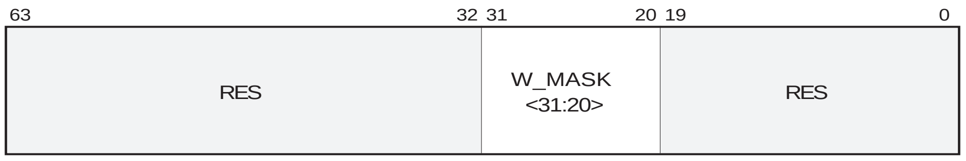


Table 6–17 IOC_W_MASK1–0 Registers Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:20	W_MASK <31:20>	WO	Window mask—This field specifies the size of the PCI target window. It can also mask the address bits that are not used when translating the PCI address to the CPU address to determine whether an address matches the PCI target window address range.
19:0	RES	X	Reserved.

6.4.11 Translated Base Registers

There is one translated base (IOC_T_BASE) register for each PCI target window.

When direct-mapped address translation is used, the translated address is generated by concatenating bits from an IOC_T_BASE register with bits from the incoming PCI address. The IOC_W_MASK register determines which IOC_T_BASE register bits and PCI address bits are concatenated to form the translated address. The translated address is the starting address in system memory for the PCI transaction.

When scatter-gather mapped address translation is used, the translated address is generated by table lookup. The table (scatter-gather map) is stored in system memory. The IOC_T_BASE register specifies the starting address of the scatter-gather map, and the incoming PCI address bits are used as an offset from that address. (See Sections 6.3.2.1 and 6.3.2.2 for more information about using these registers.)

Software should ensure that the PCI is idle before updating these registers.

The IOC_T_BASE0 register is accessed at address 180000180_{16} and the IOC_T_BASE1 register is accessed at address 1800001A0. All bits of these registers are UNDEFINED after reset.

Figure 6–11 shows the IOC_T_BASE register format, and Table 6–18 describes its fields.

Figure 6–11 IOC_T_BASE1–0 Registers Format

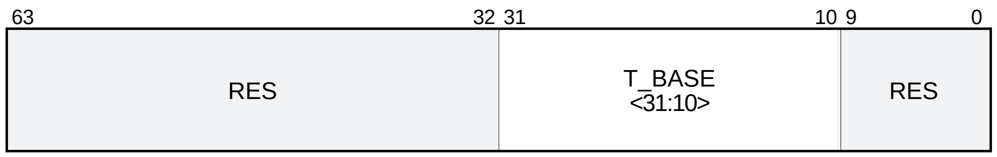


Table 6–18 IOC_T_BASE1–0 Registers Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:10	T_BASE <31:10>	WO	Translation base—When scatter-gather mapping is enabled, this field specifies the base address in system memory for the scatter-gather map. When scatter-gather mapping is disabled, this field and the incoming PCI address are concatenated to form the translated address.
9:0	RES	X	Reserved.

6.4.12 Translation Buffer Tag Registers

There is one translation buffer tag (IOC_TB_TAG) register for each TLB entry. The IOC automatically updates these registers (IOC_TB_TAG7–0) when all of the following conditions are met:

- The TEN bit is set in the IOC_TB_ENA register (Section 6.4.6).
- A PCI transaction hits in a PCI target window.
- The WEN and SG bits are both set in the IOC_W_BASE register (Section 6.4.9) associated with the PCI target window.

The IOC_TB_TAG registers can also be written directly for diagnostic purposes.

See Section 6.3.3 for more information about the TLB.

The IOC_TB_TAG registers can be accessed at the following addresses:

Register	Address
IOC_TB_TAG0	188000000
IOC_TB_TAG1	188000020
IOC_TB_TAG2	188000040
IOC_TB_TAG3	188000060
IOC_TB_TAG4	188000080
IOC_TB_TAG5	1880000A0
IOC_TB_TAG6	1880000C0
IOC_TB_TAG7	1880000E0

All bits of these registers are UNDEFINED after reset.

Figure 6–12 shows the IOC_TB_TAG register format, and Table 6–19 describes its fields.

Figure 6–12 IOC_TB_TAG7–0 Registers Format

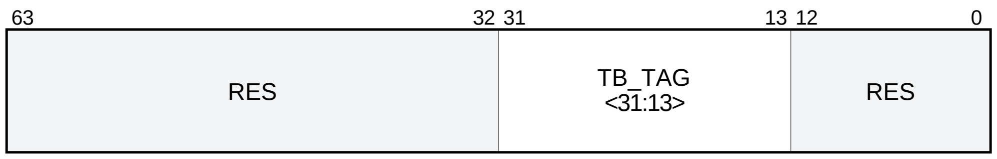


Table 6–19 IOC_TB_TAG7–0 Registers Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:13	TB_TAG <31:13>	WO	Translation buffer tag—This field provides write-only access to one of eight TLB tags.
12:0	RES	X	Reserved.

6.4.13 Interrupt Vector and Special Cycle Register

When the interrupt vector and special cycle (IOC_IACK_SC) register is read, the IOC generates an interrupt acknowledge cycle on the PCI bus and returns the vector data provided by external logic as the read data to the CPU.

When the IOC_IACK_SC register is written, the IOC generates a PCI transfer with a special cycle command code. The data written to the register is transferred unmodified to the PCI bus and contains the special message data.

Note

Digital recommends that software use only address 1A0000000 to reference the IOC_IACK_SC register. However, in the current chip implementation, CPU address bits <28:0> are not decoded to access the IOC_IACK_SC register, and any address in the range 1A0000000..1BFFFFFE0 will alias to this single register. For implementation convenience, the chip will drive the supplied address onto the PCI bus.

All bits of the IOC_IACK_SC register are UNDEFINED after reset.

Figure 6–13 shows the IOC_IACK_SC register format, and Table 6–20 describes its fields.

Figure 6–13 IOC_IACK_SC Register Format

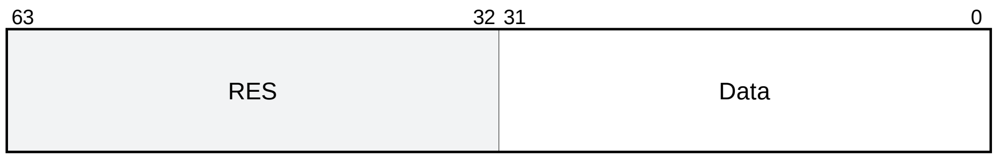


Table 6–20 IOC_IACK_SC Register Field Description

Bits	Field	Type	Description
63:32	RES	X	Reserved.
31:0	Data	RW	When the IOC_IACK_SC register is read, this field contains an interrupt vector supplied by external logic and is returned to the CPU. When the IOC_IACK_SC register is written, this field contains the data for a PCI special cycle.

6.4.14 Initialization Requirements

After the system power is turned on or after a reset condition (**reset_in_l** is asserted), serial ROM (SROM) code must initialize the IOC registers.

The SROM code does the following:

1. Deasserts the **rst_l** signal by writing zero (0) to the IOC_SFT_RST register (Section 6.4.7).

2. Clears the ERR and LOST bits by writing ones (1's) to IOC_STAT0 register bits <5:4> (Section 6.4.3).
3. Prevents the IOC from responding to itself by writing zero to the IOC_W_BASE1 and IOC_W_BASE0 registers (Section 6.4.9).
4. Initializes the HAE<31:27> field in the IOC_HAE register (Section 6.4.1).

This level of initialization is sufficient to allow CPU-initiated PCI transfers to operate correctly. Boot code must do additional initialization to configure the IOC such that it can respond to PCI-initiated transfers (DMA).

6.5 I/O Controller Signal Pins

Sections 6.5.1 through 6.5.16 describe the IOC signal pins.

6.5.1 req_l

The **req_l** signal is asserted when the CPU needs to initiate a PCI transfer. External arbitration logic is required.

The **req_l** signal is an output-only signal and can be tristated.

6.5.2 gnt_l

The **gnt_l** signal is asserted by external arbitration logic when the CPU is granted ownership of the PCI bus. Digital recommends that the arbiter default grant (park) the PCI bus to the CPU when no other device is requesting ownership.

During configuration cycles (read or write operations), the IOC performs address stepping (drives address and command for two PCI cycles). If the **gnt_l** signal is deasserted after the IOC has driven the address and command for one cycle but before the **frame_l** signal is asserted, the IOC will relinquish the bus and wait for the **gnt_l** signal to be asserted and the PCI bus idle condition.

Note

The microprocessor might *never* be able to complete a configuration cycle if the minimum assertion time for the **gnt_l** signal is one PCI cycle. The system designer must design around this potential problem by giving the microprocessor adequate time to complete its transactions. A simple solution is to specify two PCI cycles as the minimum assertion time for the **gnt_l** signal.

The **gnt_l** signal is an input-only signal.

6.5.3 frame_l

The **frame_l** signal is asserted at the beginning of a PCI transaction. It is also used to control the number of data transfers during the transaction (burst length). The **frame_l** signal is deasserted during the final data phase of a transaction.

The **frame_l** signal is a bidirectional signal and can be tristated.

6.5.4 ad<31:0>

PCI address and data are multiplexed on the **ad<31:0>** pins. During the first clock of a PCI transaction, the byte address is driven on the **ad<31:0>** pins. During subsequent clock cycles, data is driven on the **ad<31:0>** pins.

The **ad<31:0>** signals are bidirectional signals and can be tristated.

6.5.5 c_be_l<3:0>

PCI bus command codes and byte enables are multiplexed on the **c_be_l<3:0>** pins. During the address cycle of a PCI transaction, the PCI bus command code is driven on the **c_be_l<3:0>** pins. During data cycles, inverted byte enables are driven on the **c_be_l<3:0>** pins.

The **c_be_l<3:0>** signals are bidirectional signals and can be tristated.

6.5.6 irdy_l

The initiator of a PCI transaction asserts the **irdy_l** signal to indicate its ability to complete the current data phase of a PCI transaction. During a read cycle, the initiator asserts the **irdy_l** signal to indicate that it is ready to accept read data. During a write cycle, the initiator asserts the **irdy_l** signal to indicate that it is driving valid write data on the **ad<31:0>** pins. The current data phase is completed when both the **trdy_l** and **irdy_l** signals are sampled asserted.

The **irdy_l** signal is a bidirectional signal and can be tristated.

6.5.7 trdy_l

The target of a PCI transaction asserts the **trdy_l** signal to indicate its ability to complete the current data phase of a PCI transaction. During a read cycle, the device asserts the **trdy_l** signal to indicate that valid data is being driven onto the **ad<31:0>** pins. During a write cycle, the device asserts the **trdy_l** signal to indicate that it is ready to accept write data. The current data phase is completed when both the **trdy_l** and **irdy_l** signals are sampled asserted.

The **trdy_l** signal is a bidirectional signal and can be tristated.

6.5.8 devsel_l

The device that is addressed by the current PCI transaction asserts the **devsel_l** signal.

The **devsel_l** signal is a bidirectional signal and can be tristated.

6.5.9 lock_l

The **lock_l** signal is used to implement exclusive (atomic) access operations on the PCI bus.

The **lock_l** signal is an input-only signal.

In test mode, this pin is also used to test the retry timeout counter at the chip tester (Section 8.4.1).

6.5.10 stop_l

The target of a PCI transaction drives the **stop_l** signal to request that the initiator stop the current transaction.

The **stop_l** signal is a bidirectional signal and can be tristated.

6.5.11 par

The **par** signal is the even-parity signal for the **ad<31:0>** and **c_be_l<3:0>** signals.

The **par** signal is a bidirectional signal and can be tristated.

6.5.12 perr_l

The **perr_l** signal is asserted when a data parity error has been detected.

The **perr_l** signal is a bidirectional signal and can be tristated.

6.5.13 rst_l

The **rst_l** signal is the PCI reset signal. It is generated by the CPU. The RST bit (<6>) in the IOC_SFT_RST register (Section 6.4.7) allows the **rst_l** signal to be asserted under software control. The **rst_l** signal is automatically asserted when the **reset_in_l** signal is asserted.

The **rst_l** signal is an output-only signal.

6.5.14 pci_clk_in

The **pci_clk_in** signal provides timing for all transactions on the PCI bus. All of the IOC PCI signals, except the **rst_l** signal, are synchronous with the **pci_clk_in** signal. Inputs are sampled on the rising edge of the **pci_clk_in** signal; outputs change state as a result of the rising edge of the **pci_clk_in** signal.

The **pci_clk_in** signal is an input-only signal.

6.5.15 memreq_l

The **memreq_l** signal is a PCI sideband, input-only signal and is synchronous with the **pci_clk_in** signal. The IOC arbitrates (internal to the chip) for access to memory when it samples the **memreq_l** signal asserted. The IOC asserts the **memack_l** signal when it wins arbitration for memory access.

6.5.16 memack_l

The **memack_l** signal is a PCI sideband, output-only signal and is synchronous with the **pci_clk_in** signal. The IOC asserts the **memack_l** signal when it wins arbitration (internal to the chip) for access to memory as requested by the **memreq_l** signal. The **memack_l** signal remains asserted until the **memreq_l** signal is deasserted.

The **memack_l** signal is tristated during reset.

7 Clocks

This chapter describes the clock interface, the phase-locked loop (PLL), clock frequency selection, and noise reduction requirements.

7.1 Overview

The chip implements a PLL to synthesize a high-frequency internal clock from a low-frequency external clock. The synthesized clock is only used internally. It has no timing relationship to the PCI bus. Furthermore, a timing relationship should not be assumed between the synthesized clock and the external reference clock, the internal clock, or test clock output.

7.2 Phase-Locked Loop

Figure 7–1 is a simplified diagram of the internal clock generator. The multiplexer and PLL blocks comprise the voltage-controlled oscillator (VCO) and related components that implement the PLL. The divide by N (÷N) block completes the PLL.

N is the frequency ratio between the VCO output and the external clock reference driving the **pll_clk_in** and **pll_clk_in_l** pins. The ÷N block provides a feedback path from the VCO output to the PLL phase-detector input. The feedback path is programmable when the system power is turned on to select different initial synthesis ratios for the PLL.

The PLL output is further divided by Q (÷Q) to complete the internal frequency generation process. The internal clock frequency is a multiple of the value derived by the following equation:

$(N \div 2Q) \times \textit{external clock frequency}$

The final divide by 2 (÷2) provides a clean 50% duty-cycle square wave to be driven on **lca_clk** (the internal clock node). The **test_clk_out** signal is simply driven offchip from some selected point in the global buffering between the final multiplexer output and **lca_clk**. The skew between **lca_clk** and the **test_clk_out** signal is controlled only to the extent that the rising edges are nominally matched in simulation.

Figure 7–1 PLL Clock Generator

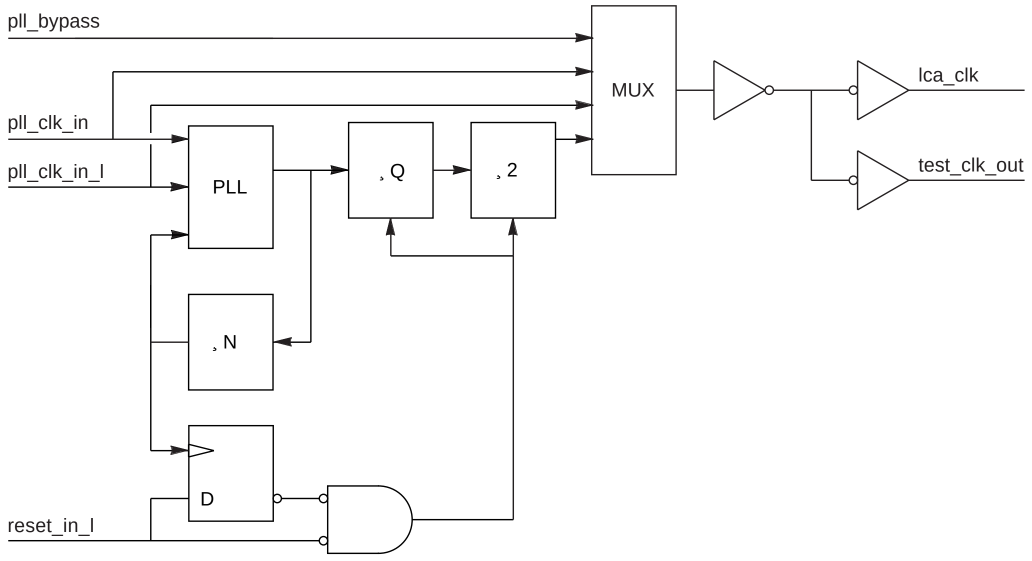


Table 7–1 lists the external values supplied on the **irq<2:0>** pins while the **reset_in_l** signal is asserted. The values supplied on the **irq<2:0>** pins are latched when the **reset_in_l** signal is deasserted. During and after chip reset, these values select the N and Q values listed in Table 7–1. The N divider starts without reset; its starting state is UNPREDICTABLE.

Table 7–1 Internal Clock Programming

irq			Internal Clock Frequency ÷ External Clock Input Frequency	
<2:0>	N	Q	pll_bypass = 0	pll_bypass = 1
000	12	3	2	1
001	12	2	3	1
010	16	2	4	1
011	10	1	5	1
100	12	1	6	1
101	14	1	7	1
110	16	1	8	1
111	18	1	9	1

Note

The maximum internal clock frequency is determined by system (package and power) requirements. Operating frequencies (**lca_clk** speeds) from two to nine times the selected oscillator or input clock frequency are supported. Making the **pll_clk_in** signal the **pci_clk_in** signal eliminates the need for separate oscillators.

7.3 Clock Signal Pins

Sections 7.3.1 through 7.3.8 describe the signal pins associated with the clock function.

7.3.1 irq<2:0>

The **irq<2:0>** pins are not usually part of the clock function. However, during reset these pins are tristated to receive (through external resistors connected to **Vss** or **Vdd**) the values that set the frequency ratio of the internal clock to the external reference clock when the power is turned on (Section 7.2).

7.3.2 pll_bypass

When the **pll_bypass** signal is asserted, the frequency ratio of the internal clock to the external reference is 1:1 and the external clock input signal **pll_clk_in** drives the internal logic directly. In other words, the clock received at the input pins is buffered to become the internal clock.

The **pll_bypass** signal should change state only when the **reset_in_l** signal is asserted.

7.3.3 pll_clk_in, pll_clk_in_l

For normal operation, a low-frequency (less than 50 MHz), single-ended clock is supplied to the **pll_clk_in** pin and an appropriate reference and bias voltage is applied to the **pll_clk_in_l** pin.

To minimize jitter induced by module and package noise, the **pll_clk_in** and **pll_clk_in_l** pins receive a high-frequency (greater than 50 MHz), differential reference clock (logically complementary, nominal square waves).

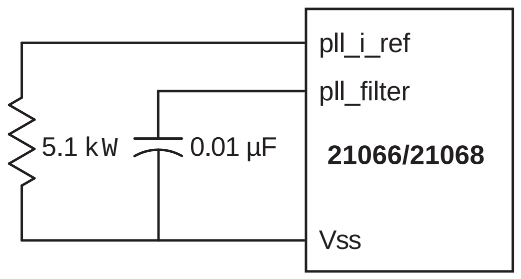
Caution

The **pll_clk_in** signal must be active before power is applied to the chip with the **pll_bypass** signal asserted.

7.3.4 pll_filter

To maintain stable operation, the 0.01-µF capacitor connected between the **pll_filter** pin and **Vss** sets the feedback-loop time constant needed to regulate the speed with which the PLL responds to changes in frequency or operating conditions (Figure 7–2).

Figure 7–2 PLL Reference Current and Filter Circuit



7.3.5 pll_i_ref

The constant current that flows in the 5.1-kΩ resistor connected between the **pll_i_ref** pin and **Vss** provides the reference to the analog PLL circuits. This reduces the variations in the speed of CMOS devices over a wide range of process and operating conditions (Figure 7–2).

7.3.6 pll_5v

The clean +5 V supplied to the **pll_5v** pin is regulated internally to source the nominal +3.3 V used by the PLL and associated logic. This onchip isolation is required to reduce phase jitter. Decoupling capacitors for the +5 V should be connected as close as possible to the **pll_5v** and **Vss** pins.

7.3.7 test_clk_out

The **test_clk_out** signal is an output reference clock to be used only for testing the chip. Its rising edge into a 40-pF load nominally coincides with the start of an internal microcycle. The relationship between the **pll_clk_in** and **test_clk_out** signals can be determined following the second deassertion of the **reset_in_l** signal after either a power-up or a change in the clock frequency ratio.

For the 21066A, when **pll_bypass** = 0, **test_clk_out** is the internal clock divided by 4; when **pll_bypass** = 1, **test_clk_out** is the same frequency as the input clock.

This pin should not be used to drive module-level logic.

7.3.8 reset_in_l

The **reset_in_l** signal is the master reset input for the chip and should be asserted when power is first applied to the chip. When the **reset_in_l** signal is asserted, certain internal chip logic is immediately initialized.

Note

Some internal state is not reset and must be handled by software when the chip boots.

The deassertion of the **reset_in_l** signal is synchronized with the internal clock. Internal chip activity starts 31 cycles after the **reset_in_l** signal is deasserted.

8
JTAG Test Port

This chapter describes the chip's implementation of a joint testing action group (JTAG) test port according to IEEE standard *1149.1, Standard Test Access Port and Boundary-Scan Architecture.* Information that is included in the standard is not duplicated in this document.

8.1 Overview

The chip contains a serial-scan test port that conforms to IEEE standard 1149.1.The port consists of the 5-wire test access port (TAP), a TAP controller, an instruction register (IR), a bypass register (BPR), and a boundary scan register (BSR).

Note

The JTAG test access port is to be used only while the CPU is not operating. JTAG operations can cause the chip to reset.

8.2 JTAG Signal Pins

This chapter describes the JTAG pins listed in Table 8–1.

Table 8–1 JTAG Pins

Pin	Type	Description
tck	I	Boundary scan clock
tdi	I	Serial boundary scan data in
tdo	O	Serial boundary scan data out
tms	I	JTAG test mode select
trst_l	I	JTAG TAP reset

8.3 Test Access Port Controller

The TAP controller is a state machine that interprets the IEEE 1149.1 protocols received on the **tms** pin and generates the appropriate clocks and control signals for the test features that it controls.

The **tms** signal changes the controller state on the rising edge of the **tck** signal. In each state, the controller generates clocks and control signals that control the operation of the test features. Test feature operations are initiated on the rising edge of the first **tck** signal after entry into a state.

8.4 Instruction Register

The 5-bit instruction register (IR) resides on a scan path. It selects the test modes and features. The IR bits are interpreted as instructions as shown in Table 8–2. The instructions select and control the operation of the boundary scan and bypass registers. During the capture-IR state, the shift-register stage of the IR is loaded with 00001_2.

Table 8–2 describes the test modes and features selected by the IR.

Table 8–2 JTAG Instruction Register

IR Contents	Instruction Name (Test Mode or State)	Test Register Selected	Operation
00000	EXTEST*	BSR	External test (drives pins from the BSR)
00001	SAMPLE	BSR	Samples I/O
00010	Reserved	—	—
00011	Reserved	—	—
00100	CLAMP*	BPR	Drives pins from the BSR and selects the BPR for shifts
00101	HIGHZ*	BPR	Tristates all output and I/O pins except the **tdo** pin
00110	Reserved	—	—
00111	Reserved	—	—
01000 through 11110	Private/ICMODE	BPR	Used for Icache initialization and internal chip testing (Table 8–3)
11111	BYPASS	BPR	Selects the BPR for shifts (Table 8–3)

*When the JTAG test port is in the EXTEST, HIGHZ, or CLAMP mode, the chip's internal reset is asserted. The chip remains in the reset state until the port leaves the mode.

Table 8–3 describes the test functions available when the IR contains an ICMODE or BYPASS instruction.

Table 8–3 JTAG ICMODE Functions

IR <4:3>	IR <2:0>	ICMODE Function
1X	000..011	Reserved.
1X	100	Icache test read mode.
1X	101	Icache test write mode.
1X	110	Disable serial interface.
1X	111	Enable SROM Icache initialization mode.
10	100..111	Select test mode of operation for IOC synchronizers. This mode bypasses the IOC synchronizers to make it easier to adjust the **pci_clk_in** and **pll_clk_in** signals for synchronous operation. Bits IR<2:0> specify one of the Icache or test modes listed in this table.
11	100..111	Select normal mode of operation for the IOC synchronizers. Bits IR<2:0> specify one of the Icache or test modes listed in this table.

X = 1 or 0

The IR also controls the function of the **lock_l** and **instr_ref** pins during reset.

8.4.1 lock_l

During test mode, this pin is used in testing the 24-bit retry timeout counter. A high on this pin enables testing the lower half of the counter; a low on this pin enables testing the upper half of the counter. This test mode is programmed by clearing IR bit 3 (IR<3> = 0). See Chapter 6 for more information about this pin.

8.4.2 instr_ref

During reset, if IR<3> = 0 and **lock_l** is deasserted, this pin is driven with a delayed noninverted PCI clock; otherwise, it is driven with a delayed, inverted core clock. See Appendix B for more information about this pin.

8.5 Bypass Register

The 1-bit bypass register (BPR) is a shift register that implements a single-bit serial connection through the port (chip) when no other test path is selected. During a scan-shift operation, the BPR provides a 1-bit scan route through the chip. It provides a way to bypass the chip's boundary scan during module-level and system-level testing.

8.6 Boundary Scan Register

The boundary scan register (BSR) primarily facilitates module interconnection testing during module manufacture or service.

The BSR is a single-shift register formed by boundary scan cells placed at most of the chip's signal pins. The register is accessed through the **tdi** and **tdo** pins.

8.6.1 Cells

The function of the BSR cells is determined by the associated pins, as follows:

- Input-only pins—The boundary scan cell is basically a 1-bit shift register. The cell supports sample and shift functions.
- Output-only pins—The boundary scan cell comprises a 1-bit shift register and an output multiplexer. The cell supports the sample, shift, and drive output functions.
- Bidirectional pins—The boundary scan cell is identical to the output-only pin cell, but it captures test data from the incoming data line. The cell supports sample, shift, drive output, and hold output functions. It is used at all I/O pins.

8.6.2 Organization

Table 8–4 lists all the signal pins in the order of the scan chain from **mem_dtoe_l** through **memack_l**. The **mem_dtoe_l** pin is the first entry in the scan chain and is the last entry scanned out. The boundary scan register on the chip is 179 bits long.

Table 8–4 also shows the four JTAG boundary scan groups. A 1-bit group control register (GCR) is associated with each group. The GCRs are physically located in the chain positions shown in Table 8–4.

The following list identifies the pins associated with each GCR:

GCR1 controls scan pins 14 through 69.
GCR2 controls scan pins 71 through 89.
GCR3 controls scan pins 91 through 137.
GCR4 controls scan pins 139 through 179.

When a GCR is set to one, the bidirectional (I/O) pins in the associated group are simultaneously tristated. At the same time, the output signals that belong to the group and are indicated by an asterisk (*) are also tristated.

Table 8–4 lists the boundary scan chain entries and signals.

Table 8–4 Boundary Scan Chain Order

Entry	Signal Pin or Register
1	**mem_dtoe_l**
2	**vrefresh_l**
3	**vframe_l**
4	**mem_dsf**
5	**mem_rasa_l3**
6	**mem_rasa_l2**
7	**mem_rasa_l1**
8	**mem_rasa_l0**
9	**mem_rasb_l3**
10	**mem_rasb_l2**
11	**mem_rasb_l1**
12	**mem_rasb_l0**
13	Group control register 1
14	**mem_ecc0**
|	|
21	**mem_ecc7**
22	**mem_data31**
|	|
53	**mem_data0**
54	**mem_rd_oe**
55	**mem_write_l**
56	**mem_wr_oe_l**
57	**mem_cas_l**
58	**mem_addr11** *
|	|
69	**mem_addr0** *
70	Group control register 2
71	**bc_index** *
72	**bc_idx_tag4**
|	|
76	**bc_idx_tag0**
77	**bc_parity**
78	**bc_dirty**
79	**bc_tag0**
|	|
86	**bc_tag7**
87	**bc_oe_l**
88	**bc_cs_l**
89	**bc_we_l**
90	Group control register 3
91	**ad0**
|	|
98	**ad7**
99	**c_be_l0**
|	|
102	**c_be_l3**
103	**lock_l**
104	**irdy_l**
105	**trdy_l**
106	**stop_l**
107	**devsel_l**
108	**perr_l**
109	**frame_l**
110	**par**
111	**gnt_l**
112	**req_l** *
113	**rst_l** *
114	**ad8**
|	|
137	**ad31**
138	Group control register 4
139	**mem_data32**
|	|
170	**mem_data63**
171	**reset_in_l**
172	**sromd**
173	**sromoe_l**
174	**sromclk**
175	**irq2**
176	**irq1**
177	**irq0**
178	**memreq_l**
179	**memack_l** *

* Output signals tristated when the associated GCR is set.

8.7 Initialization

The TAP controller and the instruction register output latches are initialized when the **trst_l** input is asserted. The TAP controller is forced to enter the test-logic reset state. The IR value is forced to 11111_2 and interpreted as follows:

- BYPASS instruction
- Select asynchronous (normal) mode of operation for the IOC
- Enable SROM Icache initialization mode

During test-logic reset state, all JTAG logic, including the boundary scan register, is in an inactive state; that is, the chip performs normal system functions. The boundary scan logic is set to a passive sample (observe) mode. The TAP controller leaves this state only when a JTAG test operation is desired and the appropriate sequence is sent on the **tms** and **tck** pins.

9 SROM Interface and Icache Initialization

This chapter describes instruction cache (Icache) initialization using the serial ROM (SROM). It also describes how the SROM port can be used as a serial terminal port after the Icache is initialized.

9.1 Overview

When the microprocessor is reset, it loads its initial Istream from a compact SROM to start the bootstrap procedure, rather than jumping to a fixed I/O address. The SROM interface can also be used for chip and module-level testing. Furthermore, by using PALcode and the serial line transmit and receive registers, the SROM port can be used to implement a diagnostic terminal port.

9.2 Instruction Cache Initialization

The microprocessor implements several Icache initialization modes to support normal initialization as well as chip and module-level testing. The value in the JTAG instruction register (Section 8.4) determines which initialization mode is used following microprocessor reset (Table 9–1). Bit 3 of the JTAG instruction register determines whether the IOC mode of operation is synchronous or asynchronous (normal).

When the enable SROM mode is selected, the microprocessor loads the Icache from the external SROM before it executes its first instruction. The SROM can contain enough code to complete the external interface configuration (for example, setting the timing on the external cache RAMs) and diagnose the path between the CPU chip and the memory components. After the **reset_in_l** signal is deasserted, the microprocessor is in PALmode, which makes all of the visible state within the chip accessible to the code loaded into the Icache.

Table 9–1 describes the test functions available when the JTAG instruction register (IR) contains an ICMODE or BYPASS instruction. See Chapter 8 for more information about the JTAG instruction register. (Table 9–1 duplicates part of Table 8–3.)

Table 9–1 Icache Test Modes

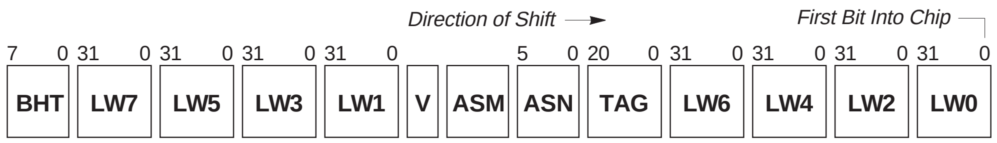
IR <4:3>	IR <2:0>	Mode
1X	000..011	Reserved.
1X	100	Icache test read mode.
1X	101	Icache test write mode.
1X	110	Disable serial interface.
1X	111	Enable SROM Icache initialization mode.
10	100..111	Select synchronous mode of operation for the IOC. Bits IR<2:0> specify one of the Icache or test modes previously listed.
11	100..111	Select asynchronous (normal) mode of operation for the IOC. Bits IR<2:0> specify one of the Icache or test modes previously listed.

X = 1 or 0

All Icache bits are loaded through the SROM interface, including each cache block's tag, address space number (ASN), address space match (ASM), valid (V) bit, and branch history table (BHT) bits. The Icache blocks are loaded in sequential order starting with block zero and ending with block 255. The bits within each block are serially loaded in the order shown in Figure 9–1.

In Figure 9–1, the most significant bit of each field is on the left. The serial chain shifts right, making bit 0 of longword 0 (LW0) the first bit shifted into the chip.

Figure 9–1 Icache Load Order



The valid and ASM bits in each cache block must be set. The tag field must be written with zero. Any value can be written into the BHT and ASN fields. The Icache load timing is described in Section 9.2.1 and the SROM signals are described in Section 9.4.

9.2.1 Icache Load Timing

When the microprocessor detects the deassertion of the **reset_in_l** signal (low-to-high transition), it loads bits from the external SROM into Icache, according to the value of the JTAG instruction register.

9.2.1.1 First Bit Samples

Figure 9–2 shows the timing for the first bit samples. The sequence is as follows:

1. The **reset_in_l** signal is asserted, the **sromoe_l** signal is deasserted, and the **sromclk** signal is asserted.
2. The microprocessor's internal reset signal remains asserted at least 20 CPU cycles after the **reset_in_l** signal is deasserted, when the **sromoe_l** signal is asserted.
3. The first rising edge of the **sromclk** signal occurs 128 CPU cycles after the **sromoe_l** signal is asserted, and every 126 CPU cycles after that.
4. The microprocessor samples the **sromd** signal in the last half of each CPU cycle that occurs before the rising edge of the **sromclk** signal.

This sequence continues until the Icache is loaded. There are 256 blocks in the Icache. Each block contains 293 bits, for a total of 75,008 bits, which results in 75,008 rising edges of the **sromclk** signal.

Although the 21066A contains additional bits in each block for parity and branch history, these extra bits are passed over while loading the Icache from SROM.

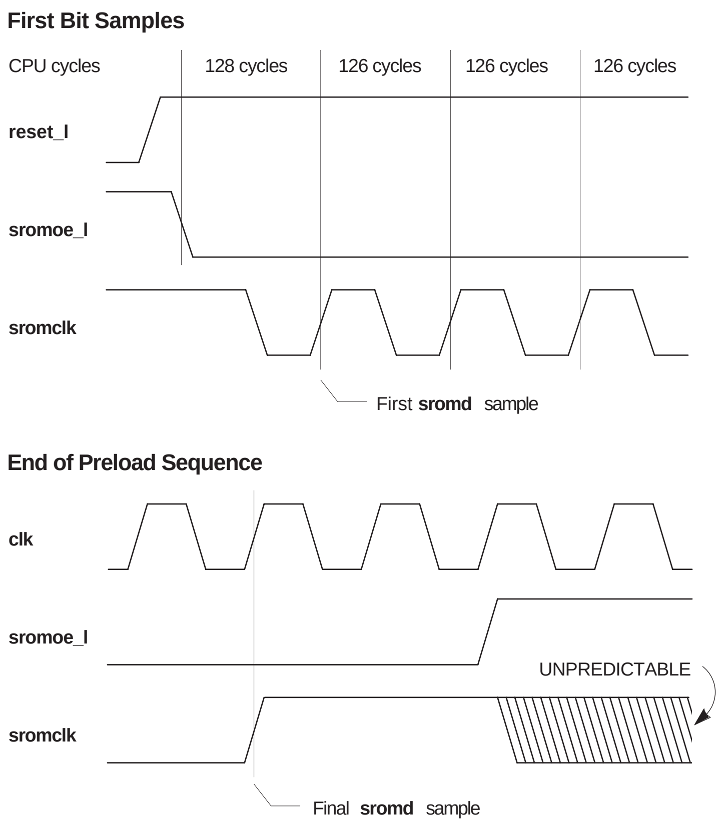
9.2.1.2 End of Preload Sequence

Figure 9–2 also shows the timing for the end of the preload sequence. At the end of the Icache preload sequence, the shaded area where the **sromclk** signal goes low indicates UNPREDICTABLE behavior. The **clk** signal represents the microprocessor's internal CPU clock and is shown as cycle reference. The sequence is as follows:

1. The microprocessor samples the final SROM bit when the **sromclk** signal goes high.
2. Two CPU cycles later, the microprocessor deasserts the **sromoe_l** signal and drives the **sromclk** pin with the value from the transmit (TMT) bit of the serial line transmit register (SL_XMIT, Section 4.1.12). Because the XMT bit is not initialized by chip reset, the value driven on the **sromclk** pin is UNPREDICTABLE.

Figure 9–2 shows the Icache load timing.

Figure 9–2 Icache Load Timing



9.2.2 Icache Test Modes

The Icache test-read and test-write modes give chip-tester hardware full read and write access to the Icache. Icache test-write mode works exactly like SROM enable mode except that bits are loaded into the Icache at a faster rate. Icache test-read mode allows the contents of the Icache to be read in a bit-serial way from the **sromoe_l** pin. These two test modes are available only to chip-tester hardware.

To correctly reset internal pointers to the Icache after writing the Icache in Icache test-write mode, the chip must be reset before using the Icache test-read mode.

9.3 Serial Line Interface

After the data from the SROM has been loaded into the Icache, the three SROM signals (Section 9.4) become simple parallel I/O pins that can be used to drive a diagnostic terminal.

When the SROM is not being read, the **sromoe_l** output signal is deasserted (high). This means that the **sromoe_l** pin can be wired to the active high enable of an RS422 receiver driving onto the **sromd** pin and to the active high enable of an RS422 driver driving from the **sromclk** pin. The CPU allows PALcode to read from the **sromd** pin and write to the **sromclk** pin. This level of hardware support is sufficient to implement a software-driven serial interface. See the serial line receive and serial line transmit register descriptions for more information (SL_RCV, Section 4.1.11 and SL_XMIT, Section 4.1.12).

9.4 SROM Interface Signals

Sections 9.4.1 through 9.4.3 describe the SROM interface signals.

9.4.1 sromclk

The **sromclk** output signal is the clock that causes the SROM to advance to the next bit.

9.4.2 sromd

The microprocessor reads SROM data from the **sromd** input pin. The read data is written to the Icache at a rate of 1 bit every 126 CPU cycles (Figure 9–2).

9.4.3 sromoe_l

The **sromoe_l** signal is the SROM output enable signal. This signal serves as both an output enable and a reset.

A
21066A Differences

This appendix describes the new functionality of the 21066A and the differences between the 21066A and previous implementations of the Alpha architecture.

Two of the three versions of the 21066A microprocessors run at significantly higher clock frequencies than their original counterparts while all three incorporate more extensive power management features. They also implement internal cache parity protection, new floating-point divider hardware, improved branch prediction, and revised write-buffer unload logic.

A.1 Power Management

The 21066A supports thermal and idleness-based power management through a programmable internal clock divider. Power dissipated by the chip is a linear function of the internal clock frequency. The internal clock frequency is selected at power up and the operating clock is generated by the internal phase-locked loop (PLL) clock generator.[1] This divider divides the operating clock frequency by the divisor specified in the power management register (PMR, Section A.6.1).

Software can specify a *primary divisor* and an *override divisor*. Each divisor divides the operating clock by a factor of 1, 1.5, 2, 4, 8, or 16. The primary divisor is the clock divider default and the override divisor is selected when the 21066A detects an enabled override event. The capability to select an override divisor allows thermal management software to operate independently of idleness-based power management software.

Latency-critical external events such as direct memory access (DMA) and interrupt requests are expedited by selecting the override divisor. When the clock divider detects an external hardware interrupt (regardless of masking in the HIER [see Section 4.1.13]) or DMA request, it switches from the primary to the override divisor. Interrupt and DMA requests are independently enabled.

[1] See Section 7.2 for more information about the PLL clock generator.

Upon completion of an overriding DMA access, the clock divisor is switched from override to primary. On the other hand, upon completion of an overriding interrupt request, the clock divisor is not switched—the override divisor remains selected until the PMR is written.

Rather than instantaneously switching from one divisor to another, the divider provides a gradual ramp of divisors between the current divisor and the target divisor (Section A.1.1). This mitigates potential problems with the power supply grid that might occur when the operating frequency is drastically changed.

The DRAM refresh interval and maximum RAS assertion time values programmed into the global timing register (GTR—for more information, see Section 5.6.5) should be based on the maximum operating frequency (the divide-by-1 frequency) of the 21066A. The refresh and RAS assertion counter logic automatically scales the programmed counts according to the currently selected divisor. The counter scaling logic assumes an instantaneous switch of divisors and does not account for the ramping process. This worst-case error between two refresh requests is no more than 2500 ns (at 200 MHz) greater than the programmed value.

A readable 32-bit counter counts all of the cycles spent in override. This counter facilitates fine-grained elapsed time measurements using the cycle counter (CC—for more information, see Section 4.2.13) register reported by the RPCC instruction.

A.1.1 Divider Ramping

In the following code sequence, the destination of the STQ instruction is the power management register:

```
MB
STQ Rn,MCTL_PMR
MB
```

Table A–1 shows the worst-case time between issue of the second memory barrier (MB) instruction and the resulting adjusted clock frequency.

Table A–1 Clock Divider Latency

Maximum Frequency (MHz)	Current Divisor	Target Divisor	Clock[*] Change Latency (Cycles)	Clock Change Latency (ns)	Override† Latency (ns)
PLL Enabled					
233	1	2	96	484	419
233	2	1	72	546	417
233	1	16	108	733	668
233	16	1	84	3161	2129
PLL Disabled (Clock Bypass Mode)					
233	1	2	176	893	828
233	2	1	128	955	826
233	1	16	196	1194	1129
233	16	1	124	3429	2397

[*]The 21066A operating cycles are based on the output of the clock divider circuitry.

†The latency associated with override clock divisor changes.

A.1.2 PCI Clock Frequency

The PCI clock input is asynchronous to the internal CPU clock. For correct operation, the PCI clock frequency must equal the lesser of the following:

- Less than or equal to the CPU post-divider internal operating frequency
- 33 MHz

A.2 Internal Cache Parity

The internal instruction and data caches (Icache and Dcache) are longword parity protected. Each cache line contains an even tag-parity bit and eight longword even data-parity bits. The Icache tag-parity bit is calculated across the address space number (ASN) field and address space match (ASM) bit in addition to the tag address. The memory controller generates data parity during fills of either cache. Each cache generates its tag parity.

Dcache and Icache parity errors set bits <4> and <5>, respectively, in the cache status register (C_STAT, Section A.6.3), and generate a machine check if so enabled in the load and store unit control register (ABOX_CTL, Section A.6.2). Setting ABOX_CTL register bit <14> disables primary cache parity checking.

Icache parity errors are recoverable—the PALcode machine-check handler can flush the Icache and return. Dcache parity errors are not recoverable.

Diagnostic code can generate bad parity by setting bit <15> in the error status register (ESR, Section A.6.4). This causes the memory controller to generate inverse (odd) data parity on subsequent cache fills. Internal cache tag-parity diagnostics can set ABOX_CTL register bit <13> to generate incorrect tag parity for both Icache and Dcache fills.

A.3 New Floating-Point Divider

The 21066A includes new floating-point divide hardware that implements a nonrestoring, normalizing, variable-shift (maximum of 4 bits per cycle) algorithm that retires an average of 2.4 bits per cycle. The average overall divide latency, including pipeline overhead, is 29 cycles for double-precision and 19 cycles for single-precision (compared to 63 and 34 cycles, respectively, in previous implementations).

Additionally, to avoid the noncompliant (IEEE) divide behavior of previous implementations, the new divider calculates the inexact flag, setting the inexact (INE) bit in the floating-point control register (FPCR) if appropriate, and trapping on DIVx/SI instructions only when the result is really inexact.[3]

The inexact trap disable bit (INED) has also been added to the FPCR.

[3] See the *Alpha Architecture Reference Manual* for more information about the FPCR.

A.4 Improved Branch Prediction

The 21066A supports an improved branch prediction scheme that uses a $2K \times 2$-bit history table. The table is indexed by the same bits that index the Icache. Each 2-bit table entry behaves as a counter that increments on branches taken (stopping at 11_2) and decrements on branches not-taken (stopping at 00). If the upper bit of the counter is set, the branch is predicted taken. The contents of the table are not disturbed by Icache fills. The 21066A also supports a static branch-prediction mode that uses the sign bit of the branch displacement (as in the 21066).

A.5 Revised Write-Buffer Unload Logic

The Alpha architecture requires that write operations will not be buffered indefinitely. The write-buffer in previous implementations does not fully comply with this requirement. In the previous implementations, the write buffer attempts to send a buffered write offchip when one of the following conditions is met:

- The write buffer contains two or more valid entries.
- The write buffer contains one valid entry and 256 cycles have elapsed since the last write was executed.

  This condition is implemented using an 8-bit counter. Counter overflow signals the write buffer to send a write operation offchip. Any of the following conditions clears the counter:

  - The write buffer is empty.
  - The write buffer unloads an entry.
  - A write operation executes.

    In the 21066A, this condition is removed from the counter’s reset equation because it allows write operations to be buffered indefinitely in the sadistic case of an indefinite stream of write operations that all merge into the same 32-byte buffer entry.
- The write buffer contains an MB or STx/C instruction.
- A load miss hits an entry in the write buffer.

A.6 Register Descriptions

Sections A.6.1 through A.6.4 describe the new 21066A power management register and the 21066A implementation-specific differences of several other registers. (The access codes in the type column of the field description tables are defined in the Conventions section of the Preface.)

A.6.1 Power Management Register

The power management register (PMR) is implemented only in the 21066A. (The PMR is ignored in the 21066; writing to it causes no side effects.) The PMR is physically mapped to 1 2000 0098_{16}.

The PMR specifies the primary and override divisors, enables the interrupt and DMA override divisors, and contains the override cycle counter. The PMR can be accessed only as a quadword; accessing the individual longwords of the register can cause UNPREDICTABLE results.

The override cycle counter (<63:48,31:16>) counts the number of cycles spent in override mode. The value is split into odd-bit and even-bit fields because of implementation constraints. The counter is cleared every time the PMR is written.

If the DMA override enable bit (<7>) is set when the I/O controller sends a request for DMA transfer service to the memory controller, the clock divisor is switched to the value specified by the override divisor field (<5:3>). The clock divisor is switched to the primary divisor value (<2:0>) when the memory controller finishes the DMA request.

If the interrupt override enable bit (<6>) is set when an interrupt is requested through the external interrupt request pins (**irq<2:0>**), the clock divisor is switched to the value specified by the override divisor field (<5:3>). Interrupt masking in the HIER has no effect on enabling the override divisor. The override divisor remains in effect after the interrupt request is deasserted, and continues to be used to divide the operating clock until the PMR is written.

Figure A–1 shows the PMR format, and Table A–2 describes its fields.

Figure A–1 PMR Format

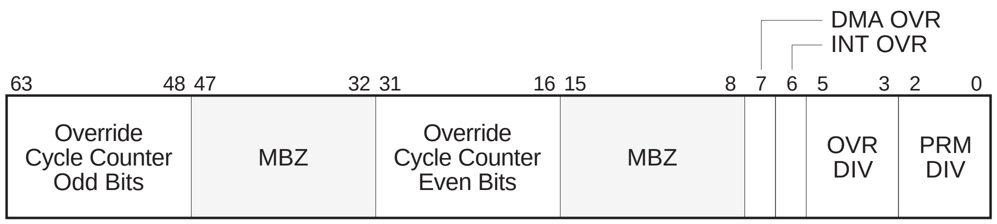

DMA OVR
INT OVR
63
48 47
32 31
16 15
8
7
6
5
3
2
0
Override Cycle Counter Odd Bits
MBZ
Override Cycle Counter Even Bits
MBZ
OVR DIV
PRM DIV

Table A–2 PMR Field Description

Bits	Field	Type	Description
63:48	Override Cycle Counter Odd Bits	WA	The 16 odd-numbered bits (31, 29,..., 1) of the 32-bit override cycle counter. Cleared when written. Undefined at reset.
47:32	MBZ	RW	Must be zero.
31:16	Override Cycle Counter Even Bits	WA	The 16 even-numbered bits (30, 28,..., 0) of the 32-bit override cycle counter. Cleared when written. Undefined at reset.
15:8	MBZ	RW	Must be zero.
7	DMA OVR	RW	DMA override—When set, enables DMA events to cause override divisor selection. When clear, DMA events have no effect on the clock divisor. Cleared at reset.
6	INT OVR	RW	Interrupt override—When set, enables interrupts to cause override divisor selection. When clear, interrupts have no effect on the clock divisor. Cleared at reset.
5:3	OVR DIV	RW	Override divisor—Specifies the value of the override divisor (Table A–3). Cleared at reset.
2:0	PRM DIV	RW	Primary divisor—Specifies the value of the primary divisor (Table A–3). Cleared at reset.

The override divisor field (<5:3>) specifies the value by which the internal operating clock is divided during periods when an enabled overriding operation is being serviced (Table A–3).

The primary divisor field (<2:0>) specifies the default value by which the internal operating clock is divided when an override operation is not being serviced (Table A–3).

Table A–3 shows the coding for the primary and override divisor fields.

Table A–3 Primary and Override Divisor Values

Bits <5:3> <2:0>	Divide Internal Clock By . . .
000	1 (value at reset)
001	1.5
010	2
011	4
100	8
101	16
110	Reserved
111	Reserved

A.6.2 Load and Store Unit Control Register

The 21066A implements two new bits (<14:13>) in the load and store unit control (ABOX_CTL) register; otherwise, the register is the same as in the 21066 (refer to Section 4.2.1).

The write-only ABOX_CTL register is cleared when written with a value of zero.

Figure A–2 shows the ABOX_CTL register format, and Table A–4 describes its fields.

Figure A–2 ABOX_CTL Register Format

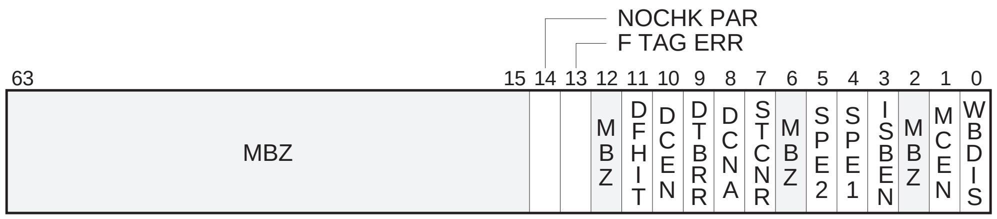


Table A–4 ABOX_CTL Register Field Description

Bits	Field	Type	Description
63:15	MBZ	WO	Must be zero.
14	NOCHK PAR	WO	No check parity—When set, disables Dcache and Icache parity checking. When clear, primary cache parity checking is enabled. Cleared at reset.
			This bit is implemented only in the 21066A.
13	F TAG ERR	WO	Fill tag error—When set, incorrect tag parity is generated on Dcache and Icache fills. When clear, normal tag parity is generated on primary cache fills. Cleared at reset.
			This bit is implemented only in the 21066A.
12	MBZ	WO	Must be zero.

(continued on next page)

Table A–4 (Cont.) ABOX_CTL Register Field Description

Bits	Field	Type	Description
11	DFHIT	WO	Dcache force hit—When set, this bit forces all data stream (Dstream) references to hit in the Dcache. This bit takes precedence over the DCEN bit (<10>). That is, when the DFHIT bit is set and the DCEN bit is clear, all Dstream references hit in the Dcache.
10	DCEN	WO	Dcache enable—When clear, disables and flushes the Dcache. When set, enables the Dcache.
9	DTBRR	WO	DTB round-robin enable—When set, the data translation buffer uses a round-robin replacement algorithm. When clear, a not-last-used (NLU) algorithm is used.
8	DCNA	WO	Dcache, no allocate—Normally, must be zero. When set, the Dcache line associated with a read address is not disturbed.
7	STCNR	WO	Store conditional, no result—Normally, must be zero. When set, it changes the way LDL_L, LDQ_L, STL_C, and STQ_C instructions are handled as follows: The results written in the register specified by the Ra field in STx_C and HW_ST/C instructions are UNPREDICTABLE. This allows the instruction fetch and decode unit to restart the memory reference pipeline when the STx_C is transferred from the write buffer to the memory controller, increasing the repetition rate with which STx/C instructions can be processed. LDx_L, STx_C and HW_ST/C instructions invalidate the Dcache line associated with their generated address. The invalidated lines are not visible to load or store instructions that issue in the two CPU cycles after the LDL_L, LDQ_L, STL_C, STQ_C, or HW_ST/C instruction is issued.
6	MBZ	WO	Must be zero.
5	SPE2	WO	Superpage enable 2—When set, enables one-to-one superpage mapping of Dstream virtual addresses VA<33:13> directly to physical addresses PA<33:13>, when virtual address VA<42:41> = 2. VA<40:34> are ignored in this translation. Access is allowed only in kernel mode.

(continued on next page)

Table A–4 (Cont.) ABOX_CTL Register Field Description

Bits	Field	Type	Description
4	SPE1	WO	Superpage enable 1—When set, enables one-to-one superpage mapping of Dstream virtual addresses with VA<42:30> = 1FFE to physical addresses with PA<33:30> = 0. Access is allowed only in kernel mode.
3	ISBEN	WO	Icache stream buffer enable—When set, enables operation of a single-entry Icache stream buffer.
2	MBZ	WO	Must be zero.
1	MCEN	WO	Machine check enable—When set, the load and store unit generates a machine check when the hardware encounters errors that it cannot correct. When clear, uncorrectable errors do not cause a machine check; however, the cache status register (C_STAT, Section A.6.3) is updated and locked when such errors occur.
0	WBDIS	WO	Write buffer unload disable—When set, prevents the write buffer from sending write data to the memory controller. This bit should be set only for diagnostics.

A.6.3 Cache Status Register

The 21066A implements two new bits (<5:4>) in the read-only cache status (C_STAT) register; otherwise, the C_STAT register is the same as the data cache status (DC_STAT) register in the 21066 (see Section 4.4).

Figure A–3 shows the C_STAT register format, and Table A–5 describes its fields.

Figure A–3 C_STAT Register Format

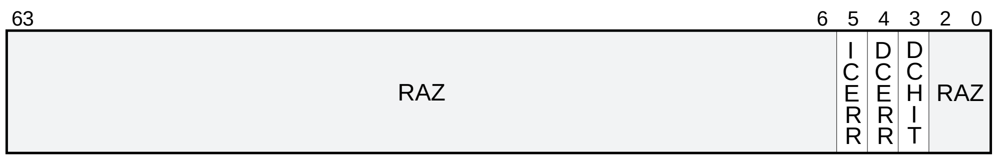


Table A–5 C_STAT Register Field Description

Bits	Field	Type	Description
63:6	RAZ	RO	Read as zero.
5	ICERR	RC	Icache error—When set, indicates an Icache parity error occurred. Cleared when read. This bit is implemented only in the 21066A.
4	DCERR	RC	Dcache error—When set, indicates an Dcache parity error occurred. Cleared when read. This bit is implemented only in the 21066A.
3	DCHIT	RO	Dcache hit—Indicates whether the last load or store instruction processed by the load and store unit hit (DCHIT set) or missed (DCHIT clear) the Dcache. Loads that miss the Dcache can be completed without requiring external reads.
2:0	RAZ	RO	Read as zero.

A.6.4 Error Status Register

The 21066A implements one new bit (<15>) in the error status register (ESR) and changes the interpretation of the chip ID field (<19:16>); otherwise, the ESR is the same as in the 21066 (see Section 5.6.6).

The ESR controls the error detection and correction functions of the memory controller and holds the error status when an error occurs.

The ECC bits are read-only and the write-wrong-ECC (WEC) bits are read and write. The error status flags (CEE, UEE, CTE, MSE, MHE) can be read and are cleared by writing a one to them; writing a zero has no effect. Reset has no effect on this register.

Figure A–4 shows the ESR format, and Table A–6 describes its fields.

Figure A–4 production$:[21066a.art.ps_final]ESR Format

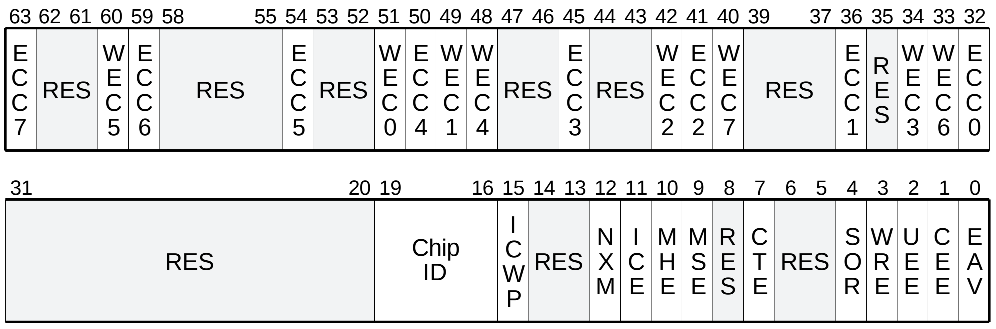


Table A–6 ESR Field Description

Bits	Field	Type	Description
Error Correction Code Bits			
The following bits store the ECC read from the **mem_ecc<7:0>** pins. These bits are frozen when the error address valid (EAV) bit (<0>) is set.			
63	ECC7	RO	
59	ECC6	RO	
54	ECC5	RO	
50	ECC4	RO	
45	ECC3	RO	
41	ECC2	RO	
36	ECC1	RO	
32	ECC0	RO	
Write-Wrong-Error Correction Code Bits			
The following bits are used when testing the ECC detection and correction logic. When set to a one, the corresponding ECC check bit will be written incorrectly to memory or cache. For normal operation, the value written into these bits should be zero.			
60	WEC5	RW	
51	WEC0	RW	
49	WEC1	RW	
48	WEC4	RW	
42	WEC2	RW	
40	WEC7	RW	
34	WEC3	RW	
33	WEC6	RW	
Reserved Bits			
The following bits are reserved:			
62:61	RES	RAX/IGN	
58:55	RES	RAX/IGN	
53:52	RES	RAX/IGN	
47:46	RES	RAX/IGN	
44:43	RES	RAX/IGN	
39:37	RES	RAX/IGN	
35	RES	RAX/IGN	
31:20	RES	RAX/IGN	
14:13	RES	RAX/IGN	
8	RES	RAX/IGN	
6:5	RES	RAX/IGN	

(continued on next page)

Table A–6 (Cont.) ESR Field Description

Bits	Field	Type	Description
Bits <19:15,12:9,7,4:0>			
19:16	Chip ID	RO	Identifies the chip. Its value is 0010 for the 21066A and 0000 for the 21066.
15	ICWP	RW	Internal cache wrong parity—When set, forces the incorrect parity to be written on subsequent Dcache and Icache fills. Cleared at reset. This bit is implemented only in the 21066A.
12	NXM	R/W1C	Nonexistent memory—Set when a DRAM read or write operation is attempted at an address that is not programmed in the bank address mask registers (BMR<3:0>) (see Section 5.6.3)
11	ICE	RW	Ignore correctable errors—When set, prevents the logging of correctable (soft) errors. On a soft error: • The data is corrected before being used. • The CEE and EAV bits (<1:0>) are not set. • The error address register (EAR) is not frozen (see Section 5.6.7).
10	MHE	R/W1C	Multiple hard errors—When set, indicates that an additional hard (uncorrectable) error occurred after the EAV bit (<0>) was set. The secondary error status is not logged.
9	MSE	R/W1C	Multiple soft errors—When set, indicates that an additional soft (correctable) error occurred after the EAV bit (<0>) was set. The secondary error status is not logged.
7	CTE	R/W1C	Cache tag error—When set, indicates that a tag parity error was detected during a Bcache cycle.
4	SOR	RO	Error source—Indicates whether an error occurred while accessing Bcache or memory: 0 = cache, 1 = memory. It is updated on each cycle and frozen when the EAV bit (<0>) is set.

(continued on next page)

Table A–6 (Cont.) ESR Field Description

Bits	Field	Type	Description
3	WRE	RO	Write error—Indicates whether an error occurred during a read or write access: 0 = read, 1 = write. It is updated on each cycle and frozen when the EAV bit (<0>) is set.
2	UEE	R/W1C	Uncorrectable error—Set when an uncorrectable ECC error occurs during a read or read-modify-write operation.
1	CEE	R/W1C	Correctable error—Set when a correctable ECC error occurs during a read or read-modify-write operation.
0	EAV	RO	Error address valid—When set, indicates that the address in the error address register (EAR) is a valid error address. This bit is the logical OR of bits <12,7,2:1>. While this bit is asserted, the memory controller asserts an interrupt to the CPU, and bits <12,7,4:1> are frozen, as is the EAR.

B
Pin Summary

Table B–1 is a summary of the chip pins. The pins are listed in alphabetical order within the following groups:

- Memory controller pins
- IOC (PCI bus) pins
- Clock pins
- JTAG pins
- Instruction reference, interrupt, and SROM interface pins

Table B–1 Pin Summary

Name	Qty	Type	Description	Value at Reset
Memory Controller Pins				
bc_cs_l	1	O	Bcache chip select	Driven, asserted
bc_dirty	1	I/O	Bcache valid	Tristate
bc_idx_tag<4:0>	5	I/O	Bcache index or tag	Tristate
bc_index	1	O	Bcache index (bit <12>)	Driven, UNDEFINED
bc_oe_l	1	O	Bcache output enable	Driven, asserted
bc_parity	1	I/O	Bcache tag parity	Tristate
bc_tag<7:0>	8	I/O	Bcache tag	Tristate
bc_we_l	1	O	Bcache write-enable	Driven, deasserted
mem_addr<11:0>	12	O	Row/column address, Bcache index	Driven, UNDEFINED
mem_cas_l	1	O	Column address strobe	Driven, deasserted

Qty = Quantity
NA = Not applicable

(continued on next page)

Table B–1 (Cont.) Pin Summary

Name	Qty	Type	Description	Value at Reset
Memory Controller Pins				
mem_data<63:0>	64	I/O	Memory/Bcache data	Tristate
mem_dsf	1	O	Disable special function	Driven, deasserted
mem_dtoe_l	1	O	Data transfer/output enable	Driven, deasserted
mem_ecc<7:0>	8	I/O	Memory/Bcache error correction code	Tristate
mem_rasa_l<3:0> **mem_rasb_l<3:0>**	8	O	Row address strobes	Driven, deasserted
mem_rd_oe	1	O	Memory read transceiver output enable	Driven, deasserted
mem_write_l	1	O	Write-enable	Driven, deasserted
mem_wr_oe_l	1	O	Memory write transceiver output enable	Driven, asserted
vframe_l	1	I	Load video display pointer and load VRAM shift register	NA
vrefresh_l	1	I	Increment video display pointer and load VRAM shift register	NA
IOC (PCI Bus) Pins				
ad<31:0>	32	I/O	PCI multiplexed address and data bus	Tristate when **gnt_l** is deasserted; otherwise, UNDEFINED
c_be_l<3:0>	4	I/O	PCI multiplexed cycle command and byte enables	Tristate when **gnt_l** is deasserted; otherwise, UNDEFINED
devsel_l	1	I/O	PCI device select	Tristate
frame_l	1	I/O	PCI cycle frame	Tristate
gnt_l	1	I	PCI bus grant	NA
irdy_l	1	I/O	PCI initiator ready	Tristate
lock_l	1	I	PCI lock. This pin is also used in testing the 24-bit retry timeout counter during test mode.	NA

Qty = Quantity
NA = Not applicable

(continued on next page)

Table B–1 (Cont.) Pin Summary

Name	Qty	Type	Description	Value at Reset
IOC (PCI Bus) Pins				
memack_l	1	O	Memory access grant from IOC	Tristate
memreq_l	1	I	PCI request for memory access	NA
par	1	I/O	PCI parity data	Tristate when **gnt_l** is deasserted; otherwise, UNDEFINED
pci_clk_in	1	I	PCI clock input	NA
perr_l	1	I/O	PCI parity error	Tristate
req_l	1	O	PCI bus request	Tristate
rst_l	1	O	PCI reset	Asserted
stop_l	1	I/O	PCI target stop	Tristate
trdy_l	1	I/O	PCI target ready	Tristate
Clock Pins				
pll_bypass	1	I	PLL bypass select	NA
pll_clk_in	1	I	PLL clock input	NA
pll_clk_in_l	1	I	PLL clock input low	NA
pll_filter	1	I	PLL low-pass filter capacitor	NA
pll_i_ref	1	I	PLL reference current	NA
pll_5v	1	I	PLL voltage supply	NA
reset_in_l	1	I	Master reset input	NA
test_clk_out	1	O	Output clock	Driven, clocking
JTAG Pins				
tck	1	I	Boundary scan clock	NA
tdi	1	I	Serial boundary scan data in	NA
tdo	1	O	Serial boundary scan data out	Determined by the state of the JTAG controllers

Qty = Quantity
NA = Not applicable

(continued on next page)

Table B–1 (Cont.) Pin Summary

Name	Qty	Type	Description	Value at Reset
JTAG Pins				
tms	1	I	JTAG test mode select	NA
trst_l	1	I	JTAG TAP reset	NA
Instruction Reference, Interrupt, and SROM Interface Pins				
instr_ref	1	O	This pin has dual functions. During normal operation, it indicates the type of current access (instruction or data) on the memory data bus. A high on this pin indicates Istream reference; a low on this pin indicates Dstream reference. During reset, this pin is driven with a delayed noninverted PCI clock or a delayed, inverted core clock (Section 8.4.2).	Driven
irq<2:0>	3	I	External interrupt request inputs. These pins are also used to program chip internal clock frequency during reset.	NA
sromclk	1	O	SROM clock/transmit serial data	Driven high
sromd	1	I	SROM data/receive serial data	NA
sromoe_l	1	O	SROM output enable	Driven, deasserted

Qty = Quantity
NA = Not applicable

C
Internal Register Summary

This chapter summarizes the microprocessor’s internal registers (internal processor, memory controller, and I/O controller registers).

Table C–1 is a summary of the internal processor registers.

Table C–1 Implementation-Specific Internal Processor Registers

Mnemonic	Register Name	Field[1]	Index[1]	Reset State
Instruction Fetch and Decode Unit Registers				
ASTER	Asynchronous system trap interrupt enable	IBX	18	UNDEFINED
ASTRR	Asynchronous system trap request	IBX	14	UNDEFINED
EXC_ADDR	Exception address	IBX	4	UNDEFINED
EXC_SUM	Exception summary	IBX	10	UNDEFINED
HIER	Hardware interrupt enable	IBX	16	UNDEFINED
HIRR	Hardware interrupt request	IBX	12	UNDEFINED
ICCSR	Instruction cache control and status	IBX	2	Cleared except ASN, PC0, and PC1 bits
ITBASM	Instruction translation buffer address space match	IBX	7	NA[2]
ITBIS	Instruction translation buffer initial state	IBX	8	NA[2]
ITB_PTE	Instruction translation buffer page table entry	IBX	1	UNDEFINED
ITB_PTE_TEMP	Instruction translation buffer page table entry temporary	IBX	3	UNDEFINED
ITBZAP	Instruction translation buffer ZAP	IBX	6	NA[2]
PAL_BASE	PAL base address	IBX	11	Cleared

[1]HW_MFPR and HW_MTPR instruction fields: PAL, ABX, IBX, and Index (<7,6,5,4:0>).

[2]NA = not applicable

(continued on next page)

Table C–1 (Cont.) Implementation-Specific Internal Processor Registers

Mnemonic	Register Name	Field[1]	Index[1]	Reset State
Instruction Fetch and Decode Unit Registers				
PS	Processor status	IBX	9	UNDEFINED
SIER	Software interrupt enable	IBX	17	UNDEFINED
SIRR	Software interrupt request	IBX	13	UNDEFINED
SL_CLR	Clear serial line interrupt	IBX	19	NA[2]
SL_RCV	Serial line receive	IBX	5	UNDEFINED
SL_XMIT	Serial line transmit	IBX	22	UNDEFINED
TB_TAG	Translation buffer tag	IBX	0	UNDEFINED
Load and Store Unit Registers				
ABOX_CTL	Load and store unit control	ABX	14	UNDEFINED
ALT_MODE	Alternate processor mode	ABX	15	UNDEFINED
CC	Cycle counter	ABX	16	UNDEFINED
CC_CTL	Cycle counter control	ABX	17	UNDEFINED
C_STAT[3]	Cache status	ABX	12	UNDEFINED
DC_STAT[4]	Data cache status	ABX	12	UNDEFINED
DTBASM	Data translation buffer address space match	ABX	7	NA[2]
DTBIS	Data translation buffer invalidate single	ABX	8	NA[2]
DTB_PTE	Data translation buffer page table entry	ABX	2	UNDEFINED
DTB_PTE_TEMP	Data translation buffer page table entry temporary	ABX	3	UNDEFINED
DTBZAP	Data translation buffer ZAP	ABX	6	NA[2]
FLUSH_IC	Flush instruction cache	ABX	21	NA[2]
FLUSH_IC_ASM	Flush instruction cache address space match	ABX	23	NA[2]
MM_CSR	Memory management control and status	ABX	4	UNDEFINED
TB_CTL	Translation buffer control	ABX	0	UNDEFINED
VA	Virtual address	ABX	5	UNDEFINED
PAL Temporary Registers				
PAL_TEMP<31:0>	PAL_TEMP internal processor	PAL	31..0	UNDEFINED

[1]HW_MFPR and HW_MTPR instruction fields: PAL, ABX, IBX, and Index (<7,6,5,4:0>).

[2]NA = not applicable

[3]Implemented in the 21066A only

[4]Implemented in the 21066 only

Table C–2 is a summary of the memory controller registers.

Table C–2 Memory Controller Registers

Mnemonic	Register Name	Address (Hexadecimal)	Reset State
BCR0	Bank configuration 0	1 2000 0000	BAV bit = 0, all other bits are UNDEFINED
BCR1	Bank configuration 1	1 2000 0008	BAV bit = 0, all other bits are UNDEFINED
BCR2	Bank configuration 2	1 2000 0010	BAV bit = 0, all other bits are UNDEFINED
BCR3	Bank configuration 3	1 2000 0018	BAV bit = 0, all other bits are UNDEFINED
BMR0	Bank mask 0	1 2000 0020	UNDEFINED
BMR1	Bank mask 1	1 2000 0028	UNDEFINED
BMR2	Bank mask 2	1 2000 0030	UNDEFINED
BMR3	Bank mask 3	1 2000 0038	UNDEFINED
BTR0	Bank timing 0	1 2000 0040	UNDEFINED
BTR1	Bank timing 1	1 2000 0048	UNDEFINED
BTR2	Bank timing 2	1 2000 0050	UNDEFINED
BTR3	Bank timing 3	1 2000 0058	UNDEFINED
GTR	Global timing	1 2000 0060	REN bit = 0, all other bits are UNDEFINED
ESR	Error status	1 2000 0068	UNDEFINED
EAR	Error address	1 2000 0070	UNDEFINED
CAR	Cache control	1 2000 0078	BCE bit = 1, Bcache size = 0 (64 KB), PWR bit = 0, and Bcache read cycle count field = 0 (that is, speed is three cycles). All other bits are UNDEFINED.
VGR	Video and graphics control	1 2000 0080	UNDEFINED
PLM	Plane mask	1 2000 0088	UNDEFINED
FOR	Foreground	1 2000 0090	UNDEFINED
PMR*	Power management	1 2000 0098	DMA_OVR = 0, INT_OVR = 0, OVR_DIV = 0, PRM_DIV = 0, all other bits are UNDEFINED.

*Implemented in 21066A only.

Table C–3 is a summary of the I/O controller registers.

Programming Note

The IOC register address (34-bit) range is 180000000 through 1BFFFFFFF, and the IOC registers are sparsely populated in this range. Although the IOC internally decodes only a subset of the 34-bit address to access the IOC registers, Digital recommends that programs use only the addresses specified in Table C–3 to ensure correct operation in future implementations.

Table C–3 I/O Controller Registers

Mnemonic	Register Name	Address	Reset State and Required Initialization
IOC_HAE	Host address extension	1 8000 0000	UNDEFINED
IOC_CONF	Configuration cycle type	1 8000 0020	UNDEFINED
IOC_STAT0	Status 0	1 8000 0040	UNDEFINED
IOC_STAT1	Status 1	1 8000 0060	UNDEFINED
IOC_TBIA	Translation buffer invalidate all	1 8000 0080	The TLB is UNDEFINED and must be initialized by software.
IOC_TB_ENA	Translation buffer enable	1 8000 00A0	UNDEFINED
IOC_SFT_RST	PCI soft reset	1 8000 00C0	The soft reset bit, and consequently the PCI **rst_l** signal, will be asserted during and after reset until modified by writing to this register.
IOC_PAR_DIS	Parity disable	1 8000 00E0	The parity disable bit is cleared to enable parity checking and reporting after reset.
IOC_W_BASE0	Window base 0	1 8000 0100	UNDEFINED. SROM boot code must initialize the window enable (WEN) and scatter-gather (SG) bits.
IOC_W_BASE1	Window base 1	1 8000 0120	UNDEFINED. SROM boot code must initialize the window enable (WEN) and scatter-gather (SG) bits.
IOC_W_MASK0	Window mask 0	1 8000 0140	UNDEFINED
IOC_W_MASK1	Window mask 1	1 8000 0160	UNDEFINED
IOC_T_BASE0	Translated base 0	1 8000 0180	UNDEFINED

(continued on next page)

Table C–3 (Cont.) I/O Controller Registers

Mnemonic	Register Name	Address	Reset State and Required Initialization
IOC_T_BASE1	Translated base 1	1 8000 01A0	UNDEFINED
IOC_TB_TAG0	Translation buffer tag 0	1 8100 0000	UNDEFINED
IOC_TB_TAG1	Translation buffer tag 1	1 8100 0020	UNDEFINED
IOC_TB_TAG2	Translation buffer tag 2	1 8100 0040	UNDEFINED
IOC_TB_TAG3	Translation buffer tag 3	1 8100 0060	UNDEFINED
IOC_TB_TAG4	Translation buffer tag 4	1 8100 0080	UNDEFINED
IOC_TB_TAG5	Translation buffer tag 5	1 8100 00A0	UNDEFINED
IOC_TB_TAG6	Translation buffer tag 6	1 8100 00C0	UNDEFINED
IOC_TB_TAG7	Translation buffer tag 7	1 8100 00E0	UNDEFINED
IOC_IACK_SC	Interrupt vector and special cycle	*	UNDEFINED

*Any quadword-aligned address in the range 1A0000000..1BFFFFFE0.

D
Technical Support and Ordering Information

D.1 Technical Support

If you need technical support or help deciding which literature best meets your needs, call the Digital Semiconductor Information Line:

United States and Canada	**1–800–332–2717**
Outside North America	**+1–508–628–4760**

D.2 Ordering Digital Semiconductor Products

To order the Alpha 21066 or Alpha 21066A microprocessors, contact your local distributor.

You can order the following semiconductor products from Digital:

Product	Order Number
Alpha 21066–100 Microprocessor	21066–CA
Alpha 21066–166 Microprocessor	21066–AA
Alpha 21066A–233 Microprocessor	21066–AB
Alpha 21066A–100 Microprocessor	21066–CB
Alpha 21066A–266 Microprocessor	21066–DB

D.3 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 21066/21066A Microprocessors Data Sheet	EC–QC4HA–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.

D.4 Ordering Third-Party Literature

You can order the following third-party literature directly from the vendor.

Title	Vendor
PCI Local Bus Specification, Revision 2.0	PCI Special Interest Group 1–800–433–5177 (U.S.) 1–503–797–4207 (International) 1–503–234–6762 (FAX)

Index