DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors

Hardware Reference Manual

Order Number: EC–Q9ZUA–TE

Revision/Update Information: This is a preliminary manual.

Digital Equipment Corporation
Maynard, Massachusetts

June 1994

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

3 Instruction Set

4 Privileged Architecture Library Code

5 Internal Processor Registers

6 External Interface

7 Electrical Data

8 Thermal Management

9 Signal Integrity

10 Mechanical Data and Packaging Information

A Designing a System with the 21064

B Technical Support, Ordering, and Associated Literature

Glossary

Index

Figures

Tables

Preface

Audience

This reference manual is for system designers who use the DECchip 21064 or DECchip 21064A Alpha AXP microprocessors, the first implementations of Digital's Alpha AXP architecture.

Manual Organization

The information in this manual is organized into ten chapters and two appendixes.

The internal chip architecture, Alpha AXP instruction set, privileged architecture code (PALcode) and internal registers are described before the external interface. The final chapters of the manual contain electrical, thermal, signal integrity, and mechanical information.

Appendix A contains information for designing systems using the DECchip 21064. Appendix B contains information about technical support and ordering related documentation. A glossary comes next followed by an index and ordering information.

Alpha AXP architecture information is contained in the companion volume to this manual, the *Alpha Architecture Handbook*.

Terminology and Conventions

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

Microprocessor Terms

The term 21064/21064A will be used where information applies to both the DECchip 21064 and the DECchip 21064A Alpha AXP microprocessors. The terms **21064** or **21064A** will be used where information applies to only one of the Alpha AXP microprocessors.

Numbering

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

Security Holes

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

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

UNPREDICTABLE and UNDEFINED

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

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

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

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

UNPREDICTABLE

- Results or occurrences specified as UNPREDICTABLE may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. Software can never depend on results specified as UNPREDICTABLE.
- An UNPREDICTABLE result may acquire an arbitrary value subject to a few constraints. Such a result may be an arbitrary function of the input operands or of any state information that is accessible to the process in its current access mode. UNPREDICTABLE results may be unchanged from their previous values.

Operations that produce UNPREDICTABLE results may also produce exceptions.

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

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

  Also, operations that may produce UNPREDICTABLE results must not:

  - Write or modify the contents of memory locations or registers to which the current process in the current access mode does not have access.
  - Halt or hang the system or any of its components.

  For example, a security hole would exist if some UNPREDICTABLE result depended on the value of a register in another process, on the contents of processor temporary registers left behind by some previously running process, or on a sequence of actions of different processes.

UNDEFINED

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

Ranges and Extents

Ranges are specified by a pair of numbers separated by a ".." 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 brackets separated by a colon and are inclusive. For example, bits [7:3] specify an extent of bits including bits 7, 6, 5, 4, and 3.

ALIGNED and UNALIGNED

In this manual the terms ALIGNED and NATURALLY ALIGNED are used interchangeably to refer to data objects that are powers of two in size. An aligned datum of size 2**N is stored in memory at a byte address that is a multiple of 2**N, that is, one that has N low-order zeros. Thus, an aligned 64-byte stack frame has a memory address that is a multiple of 64.

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

Must Be Zero (MBZ)

Fields specified as Must Be Zero (MBZ) must never be filled by software with a non-zero value. If the processor encounters a non-zero value in a field specified as MBZ, a Reserved Operand exception occurs.

Should Be Zero (SBZ)

Fields specified as Should Be Zero (SBZ) should be filled by software with a zero value. Non-zero values in SBZ fields produce UNPREDICTABLE results and may produce extraneous instruction-issue delays.

Read As Zero (RAZ)

Fields specified as Read As Zero (RAZ) return a zero when read.

Ignore (IGN)

Fields specified as Ignore (IGN) are ignored when written.

Register Format Notation

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

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

Table 1 Register Field Type Notation

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

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

Table 2 Register Field Notation

Notation	Description
RAZ	Denotes a register bit(s) that is read as a zero.
IGN	Denotes a register bit(s) that is ignored on write and UNPREDICTABLE when read if not otherwise specified.
MBZ	Denotes a register bit(s) that must be a zero value.

DECchip 21064 and DECchip 21064A Differences Sections

The DECchip 21064 and DECchip 21064A are alike in most ways but they have some differences. Throughout this manual the bold labels **21064** and **21064A** are used to indicate that the feature or operation only applies to one of the microprocessors.

The sections, figures, and tables where these differences occur are listed here:

- Parity and ECC features in Section 1.3
- Backward compatibility of the **21064A** in Section 1.4
- **21064** Block Diagram in Figure 2–1 and **21064A** Block Diagram in Figure 2–2
- Branch prediction in Section 2.3.1.1 and Section 2.3.1.2
- Internal cache hit signals in Section 2.3.4
- Resetting the write buffer counter in Section 2.5.4
- Fbox inexact flag in Section 2.6
- Inexact disable bit added to FPCR. See Figure 2–3 and Figure 2–4
- Inexact (INE) part of IEEE floating-point conformance in Section 2.7
- Primary cache differences in Section 2.8
- ABOX_CTL Register [15:12] in Figure 5–22 and Table 5–10
- BIU_CTL Register [44,39,37,7:4] in Figure 5–26 and Table 5–12
- Cache status registers in Section 5.3.16 and Section 5.3.17
- Microprocessors logic symbols in Figure 6–1 and Figure 6–2
- **dInvReq_h** in Table 6–2
- **tagAdr_h**, **tagEq_l** and **dMapWE_h** in Table 6–3
- **irq_h** and **sysClkDiv_h** in Table 6–5
- **icMode_h** in Table 6–6
- **resetSClk_h** in Table 6–7
- Fast lock mode signals in Table 6–8
- Reset signal states in Section 6.4.1

- LDL_L/LDQ_L and STL_C/STQ_C transactions in Section 6.4.10
- System clock divisor and assertion delay in Section 6.5.1
- Primary cache invalidates in Section 6.5.3
- Fast lock mode effect on LDL_L/LDQ_L in Section 6.5.4
- Tristate driver note in Section 6.5.4.4
- **tagOK** synchronization in Section 6.5.4.5
- Check bits during reads in Section 6.5.5.2
- **21064A** data protection mode selection in Section 6.5.9.3
- **21064A** byte parity data protection in Table 6–24
- **20164A** cache parity errors in Section 6.6.5
- Maximum electrical ratings in Table 7–1
- Reference voltage for **tagOK_h** and **tagOK_l** in Section 7.2.1 and Section 7.4.1.
- Input clock timing in Table 7–4 and Table 7–5
- Subtable with input setup relative to **sysClkOut1_h** in Section 7.4.5
- READ_BLOCK Timing in Figure 7–6 and Figure 7–7
- WRITE_BLOCK Timing in Figure 7–8 and Figure 7–9
- BARRIER Timing in Figure 7–10 and Figure 7–11
- FETCH/FETCH_M Timing in Figure 7–12 and Figure 7–13
- **tagEq_l** in Section 7.4.6
- **tagOK_h** and **tagOK_l** synchronization in Section 7.4.7 and Section 7.4.8
- Power considerations in Section 8.2.2
- Thermal characteristics and parameters with heat sink in a forced-air environment in Table 8–1 and Table 8–4
- Pin List differences in Table 10–1

1

Introduction to the 21064/21064A

1.1 Introduction

This chapter introduces the 21064/21064A. The descriptions and lists are meant to familiarize the reader with the microprocessors but are not in great detail or depth. The chapter is organized as follows:

- Architecture
- Chip features
- Backward compatibility

1.2 The Architecture

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

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

The Alpha AXP architecture supports the following data types:

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

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

The 21064/21064A uses a set of subroutines, called privileged architecture library code (PALcode), that is specific to a particular Alpha AXP operating system implementation and hardware platform. These subroutines provide operating system primitives for context switching, interrupts, exceptions, and memory management. These subroutines can be invoked by hardware or CALL_PAL instructions. CALL_PAL instructions use the function field of the instruction to vector to a specified subroutine. PALcode is written in standard machine code with some implementation-specific extensions to provide direct access to low-level hardware functions. PALcode supports optimizations for multiple operating systems, flexible memory management implementations, and multi-instruction atomic sequences.

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

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

1.3 Chip Features

The DECchip 21064/21064A microprocessors are some of the first in a family of chips implementing the Alpha AXP architecture. The 21064/21064A are CMOS super-scalar super-pipelined microprocessors using dual instruction issue.

The 21064/21064A and associated PALcode implements IEEE single and double precision, VAX F_floating and G_floating datatypes and supports longword (32-bit) and quadword (64-bit) integers. Byte (8-bit) and word (16-bit) support is provided by byte manipulation instructions. Limited hardware support is provided for the VAX D_floating datatype.

Other 21064/21064A features include:

- **21064** peak instruction execution rate of
  - 300 million operations per second at 150 MHz clock rate
  - 332 million operations per second at 166 MHz clock rate
  - 400 million operations per second at 200 MHz clock rate
- **21064A** peak instruction execution rate of
  - 466 million operations per second at 233 MHz clock rate
  - 550 million operations per second at 275 MHz clock rate

- An internal clock generator providing a high-speed chip clock and a pair of programmable system clocks with a frequency of
  - CPU clock/2 to CPU clock/8 for **21064**
  - CPU clock/2 to CPU clock/17 for **21064A**
- Flexible external interface supporting a complete range of system sizes and performance levels while maintaining peak CPU execution speed
  - Selectable data bus width of 64 bit or 128 bit
  - Selectable data bus speed. For example 75 MHz to 18.75 MHz bus speed at 150 MHz CPU clock rate
- Support for external secondary cache including programmable cache size and speed
- An on-chip write buffer with four 32-byte entries
- An on-chip pipelined floating-point unit
- **21064** on-chip cache
  - An 8K byte instruction cache
  - An 8K byte data cache
- **21064A** on-chip cache
  - A 16K byte instruction cache
  - A 16K byte data cache
- An on-chip demand paged memory management unit consisting of:
  - A 12-entry I-stream translation buffer (ITB) with 8 entries for 8K pages and 4 entries for 4 MB pages
  - A 32-entry D-stream translation buffer (DTB) with each entry able to map a single 8K, 64K, 512K, or 4 MB page
- Parity and ECC
  - **21064** provides on-chip support for data bus parity and ECC
  - **21064A** provides parity for on-chip Icache and Dcache as well as on-chip support for data bus parity and ECC
- Chip and module level test support
- 3.3-volt power supply with interface to 5-volt logic

See Chapter 7 for the 21064/21064A electrical characteristics (dc and ac).

1.4 Backward Compatibility

The **21064A** is backward compatible with the **21064**. The compatibility includes pin layout, PALcode, and application programs.

The following restrictions apply to the compatibility between the **21064A** and **21064**:

- The **21064A** has internal pulldown resistors on inputs which are unused spare pins on the **21064**. If these spare pins are unconnected on a module designed for the **21064**, then there will be no migration problem with these pins.
- Two pins have been reallocated for other uses. If these pins were not used on a module designed for the **21064**, then there will be no migration problem with these pins. On the **21064** the two pins are **tagEq_l** and **tagAdr_h 17**; on the **21064A** they are **lockWE_h** and **lockFlag_h** respectively.

Note

See Table 10–1 for a list of pin differences between the **21064** and the **21064A**.

- The behavior of the **tagOK** protocol on the **21064A** differs from that of the **21064**. Designers should investigate the effect of the change if this protocol is used in existing **21064** modules.

2

Internal Architecture

2.1 Introduction

This chapter gives a system designer's view of the 21064/21064A micro-architecture. The chapter describes the hardware with minimal forward references to the pipeline, discussed in Section 2.9. The scheduling and dual issue rules are defined in Section 2.10 (the 21064/21064A processor can issue two instructions in a single cycle). This chapter is not intended to be a detailed hardware description of the chip.

The combination of the 21064/21064A micro-architecture and PALcode defines the chip's implementation of the Alpha AXP architecture. Many hardware design decisions were based on specific PALcode functionality. PALcode is described in Chapter 4. If a certain piece of hardware seems to be "architecturally incomplete", the missing functionality is implemented in PALcode. The chapter is organized as follows:

- Overview
- Ibox
- Ebox
- Abox
- Fbox
- IEEE Floating-point Conformance
- Cache Organization
- Pipeline Organization
- Scheduling and Issuing Rules
- PALcode

Figure 2–1 shows a block diagram of the **21064** chip.

Figure 2–1 Block Diagram of the 21064

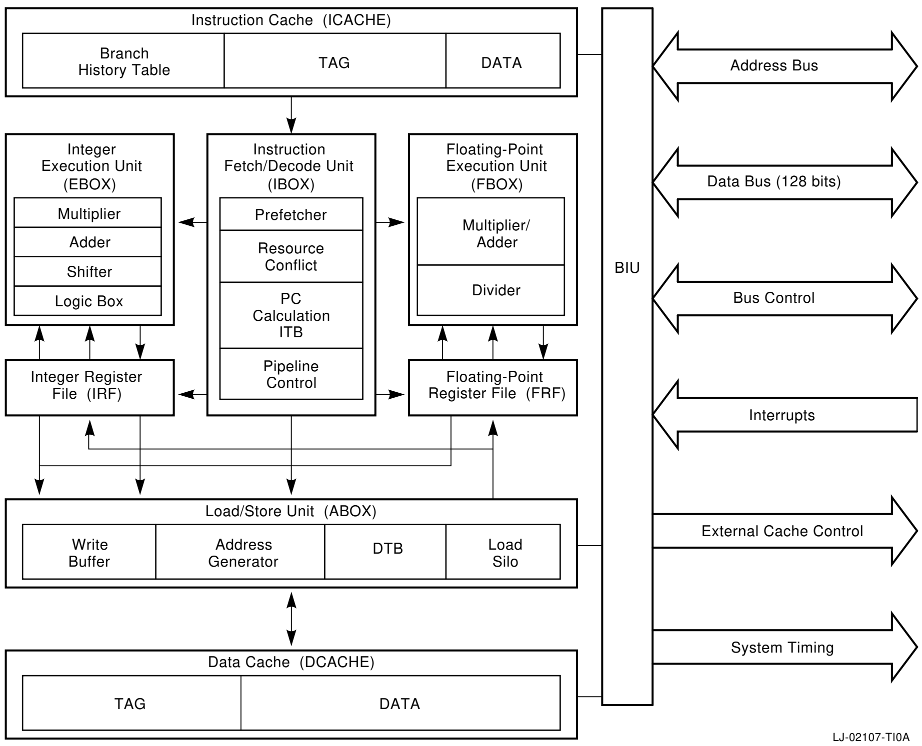
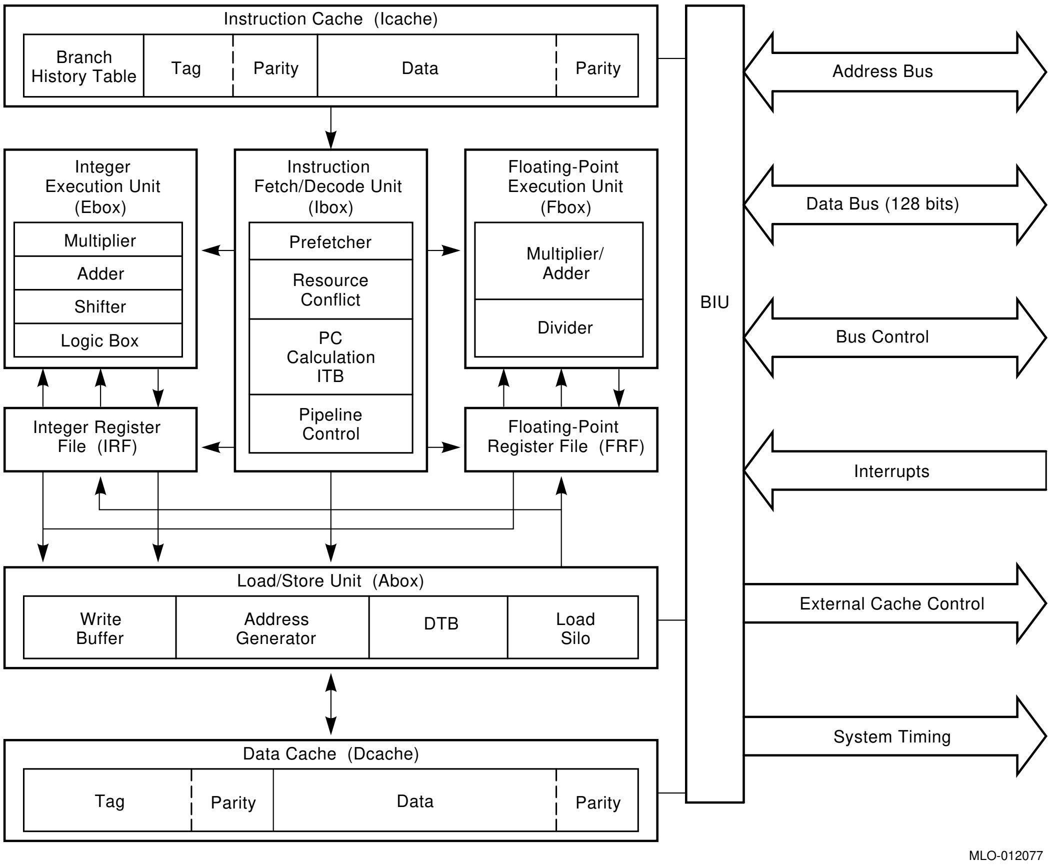


Figure 2–2 shows a block diagram of the **21064A** chip.

Figure 2–2 Block Diagram of the 21064A



2.2 21064/21064A Overview

The 21064/21064A has a central control unit referred to as the Ibox. It issues instructions, maintains the pipeline, and performs program counter (PC) calculations.

The 21064/21064A contains on-chip instruction and data caches (Icache and Dcache).

The 21064/21064A also contains four independent execution units:

- The integer execution unit (Ebox)
- The address generation, load/store and bus interface unit (Abox)
- The floating-point unit (Fbox)
- The branch logic

Each execution unit can accept at most one instruction per cycle; however if code is correctly scheduled, the 21064/21064A can issue two instructions to two independent units in a single cycle.

2.3 Ibox

The primary functions of the Ibox are to:

- Issue instructions
- Fetch instructions
- Decode instructions
- Pipeline control

The Ibox issues instructions to the Ebox, Abox, and Fbox. To provide those instructions, the Ibox contains:

- The prefetcher
- PC pipeline
- ITB
- Abort logic
- Register conflict or dirty logic
- Exception logic

The Ibox decodes two instructions in parallel and checks that the required resources are available for both instructions.

If resources are available then both instructions are issued. See Section 2.10.5 for details on instructions that can be dual issued. The Ibox does *not* issue instructions out of order; if the resources are available for the second instruction, but not for the first instruction, then the Ibox issues neither. The resources for the first instruction must be available before the Ibox issues any instructions. If the Ibox issues only the first of a pair of instructions, the Ibox does not advance another instruction to attempt dual issue again. Dual issue is only attempted on aligned quadword pairs.

2.3.1 Branch Prediction Logic

The Ibox contains the branch prediction logic. The 21064/21064A offers a choice of three branch prediction strategies selectable through the ICCSR internal processor register (IPR). The three strategies are:

- Branch will not be taken
- Branch taken is dependent on the sign of the instruction branch displacement.
- Branch taken is dependent on the branch history table

2.3.1.1 21064 Branch Prediction Logic

The prediction for the first execution of a branch instruction is based on the sign of the displacement field within the branch instruction itself. The branch is taken if the bit is negative (1) and is not taken if the bit is positive (0).

The **21064** Icache records the outcome of branch instructions in a single bit branch history table provided for each instruction location in the Icache. The bit is set when the branch is taken and cleared when the branch is not taken.

The **21064** consults the branch history table when executing the branch instruction.

- If the sign bit is negative, the instruction prefetcher predicts the conditional branch to be taken.
- If the sign is positive, the instruction prefetcher predicts the conditional branch not to be taken.

2.3.1.2 21064A Branch Prediction Logic

The **21064A** Icache records the outcome of branch instructions in a 2-bit branch history table provided for each instruction location in the Icache. The two history bits are used as a counter: incremented each time a branch is taken (stopping at 11_2) and decremented each time a branch is not taken (stopping at 00_2).

The **21064A** consults the branch history table when executing the branch instruction.

- If the higher bit is set, the instruction prefetcher predicts the conditional branch to be taken.
- If the higher bit is clear, the instruction prefetcher predicts the conditional branch not to be taken.

2.3.1.3 21064/21064A Subroutine Return Stack

The 21064/21064A provides a four-entry subroutine return stack (JSR stack) that is controlled by the hint bits in the BSR, HW_REI, and jump to subroutine instructions (JMP, JSR, RET, or JSR_COROUTINE). The chip also provides a means of disabling all branch prediction hardware. Table 2–1 lists the hint bits for the JSR stack.

Table 2–1 Architected JSR Hint Bits

disp [15:14]	Meaning	Predicted Target [15:0]	JSR Stack Action
00	JMP	PC + {4*disp[13:0]}	–
01	JSR	PC + {4*disp[13:0]}	push PC
10	RET	Prediction stack	pop
11	JSR_COROUTINE	Prediction stack	pop, push PC

To control a branch, use the BHE, JSE, and BPE bits of the ICCSR IPR. See Table 5–1.

2.3.2 Instruction Translation Buffers (ITBs)

The Ibox contains two ITBs.

- An eight-entry, fully associative translation buffer that caches recently used instruction-stream page table entries for 8K byte pages
- A four-entry, fully associative translation buffer that supports the largest granularity hint option (512 $*$ 8K byte pages) as described in the *Alpha Architecture Reference Manual*.

Both translation buffers use a not-last-used replacement algorithm. They are hereafter referred to as the small-page and large-page ITBs, respectively.

In addition, the ITB includes support for an extension called the super page, which can be enabled by the MAP bit in the ICCSR IPR. Super page mappings provide one-to-one virtual PC [33:13] to physical PC [33:13] translation when virtual address bits [42:41] = 2. When translating through the super page, the PTE[ASM] bit used in the Icache is always set. Access to the super page mapping is only allowed while executing in kernel mode.

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

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

The 21064/21064A ITB supports a single address space number (ASN) by way of the PTE [ASM] bit. Each PTE entry in the ITB contains an address space match (ASM) bit. Writes to the ITBASM IPR invalidate all entries that do not have their ASM bit set. This provides a simple method of preserving entries that map operating system regions while invalidating all others.

2.3.3 Interrupt Logic

The 21064/21064A chip supports three sources of interrupts.

- Hardware

  There are six level-sensitive hardware interrupts sourced by pins.

- Software

  There are fifteen software interrupts sourced by an on-chip IPR (SIRR).

- Asynchronous system trap (AST)

  There are four AST interrupts sourced by a second internal IPR (ASTRR).

All interrupts are independently maskable by on-chip enable registers to support a software-controlled mechanism for prioritization. In addition, AST interrupts are qualified by the current processor mode and the current state of SIER [2].

By providing distinct enable bits for each independent interrupt source, a software-controlled interrupt priority scheme can be implemented by PALcode or the operating system with maximum flexibility.

For example, the 21064/21064A can support a six-level interrupt priority scheme through the six hardware interrupt request pins. This is done by defining a distinct state of the hardware interrupt enable register (HIER) for each interrupt priority level (IPL). The state of the HIER determines the current interrupt priority. The lowest interrupt priority level is produced by enabling all six interrupts, for example bits [6:1]. The next is produced by enabling bits [6:2] and so on, to the highest interrupt priority level that is produced by enabling only bit [6], and disabling bits [5:1]. When all interrupt enable bits are cleared, the processor can not be interrupted from the hardware interrupt request register (HIRR). Each state, ([6:1], [6:2], [6:3], [6:4], [6:5], [6]) represents an individual IPL. If these states are the only states allowed in the HIER, a six-level hardware interrupt priority scheme can be controlled entirely by PALcode software.

The scheme is extensible to provide multiple interrupt sources at the same interrupt priority level by grouping enable bits. Groups of enable bits must be set and cleared together to support multiple interrupts of equal priority level. This method reduces the total available number of distinct levels.

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

> Four AST interrupts are provided, one for each processor mode. AST interrupt requests are qualified such that AST requests corresponding to a given mode are blocked whenever the processor is in a higher mode regardless of the state of the AST interrupt enable register. In addition, all AST interrupt requests are qualified in the 21064/21064A with SIER [2].

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

Note

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

2.3.4 Performance Counters

The 21064/21064A contains a performance recording feature. The implementation of this feature provides a mechanism to count various hardware events and cause an interrupt upon counter overflow. Interrupts are triggered six cycles after the event, and therefore, the exception program counter may not reflect the exact instruction causing counter overflow. Two counters are provided to allow accurate comparison of two variables under a potentially non-repeatable experimental condition.

Counter inputs include:

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

In addition, the 21064/21064A provides one chip pin input to each counter to measure external events at a rate determined by the selected system clock speed.

Note

These counters are controlled by the ICCSR IPR bits PCMUX1, PCMUX0, PC1, and PC0. See Table 5–1.

The **21064A** contains a mode in which **dMapWE_h [1:0]** are asserted during both Icache and Dcache read operations. This makes it possible to build external logic to record the frequency of Icache and Dcache block access. The user may base Bcache allocation on this information to improve overall system performance.

The mode will be entered when BIU_CTL [IMAP_EN] is set. When BIU_CTL [IMAP_EN] is set **dMapWE_h [1:0]** will be asserted during Icache reads as well as the usual assertion during Dcache reads.

When in this mode the **21064A** asserts **dMapWE_h 0** or **dMapWE_h 1** for D-stream Bcache reads. Which signal is asserted depends upon which half of the 16K byte Dcache was addressed by VA 13:

- **dMapWE_h 0** when VA 13 equals zero
- **dMapWE_h 1** when VA 13 equals one

When in this mode the **21064A** asserts either **dMapWE_h 0** or **dMapWE_h 1** when there is an I-stream Bcache read. Which of the two signals is asserted is UNPREDICTABLE.

2.4 Ebox

The Ebox contains the 64-bit integer execution data path.

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

The integer multiplier retires four bits per cycle. The Ebox also contains the 32-entry 64-bit integer register file (IRF) as shown in Figure 2–1 and Figure 2–2. The register file has four read ports and two write ports that allow reading operands from and writing operands (results) to both the integer execution data path and the Abox.

2.5 Abox

The Abox contains six major sections.

- Address translation data path
- Load silo
- Write buffer
- Dcache interface
- Internal processor registers (IPRs)
- External bus interface unit (BIU)

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

2.5.1 Data Translation Buffer (DTB)

The 21064/21064A contains a 32-entry, fully associative, translation buffer (DTB) that caches recently used data-stream page table entries and supports all four variants of the granularity hint option, as described in the *Alpha Architecture Reference Manual*.

The 21064/21064A provides an extension referred to as the super page, which can be enabled using ABOX_CTL [5:4]. Super page mappings provide virtual to physical address translation for two regions of the virtual address space. The first enables super page mapping when virtual address bits [42:41] = 2. In this mode, the entire physical address space is mapped multiple times to one quadrant of the virtual address space defined by VA [42:41] = 2. The second super page mode maps a 30-bit region of the total physical address space defined by PA [33:30] = 0 into a single corresponding region of virtual space defined by VA [42:30] = 1FFE(Hex). Super page translation is only allowed in kernel mode. The operating system by way of PALcode is responsible for ensuring that translation buffer entries, including super page regions, do not map overlapping virtual address regions at the same time.

The 21064/21064A DTB supports a single address space number (ASN) with the PTE [ASM] bit. Each PTE entry in the DTB contains an address space match (ASM) bit. Writes to the DTBASM IPR invalidate all entries that do not have their ASM bit set. This provides a simple method of preserving entries that map operating system regions while invalidating all others.

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 PFN and protection bits for the page that contains the address are used by the Abox to complete the address translation and access checks.

The DTB is filled and maintained by PALcode. Chapter 4 details the DTB miss flow. The DTB can also be filled in kernel mode by first setting the HWE bit in the ICCSR IPR before executing the HW_MTPR instruction.

2.5.2 Bus Interface Unit (BIU)

The BIU controls the interface to the 21064/21064A pin bus. (Chapter 6 describes the pin bus). The BIU responds to three classes of CPU-generated requests:

- Dcache fills
- Icache fills
- Write buffer-sourced commands

The BIU resolves simultaneous internal requests 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.

The BIU contains logic to directly access an external cache to service internal cache fill requests and writes from the write buffer. The BIU services reads and writes that do not hit in the external cache with help from external logic.

Internal data transfers between the CPU and the BIU are made through a 64-bit bidirectional bus. Since the internal cache fill block size is 32 bytes, cache fill operations result in four data transfers across this bus from the BIU to the appropriate cache. Also, because each write buffer entry is 32 bytes wide, write transactions may result in four data transfers from the write buffer to the BIU.

2.5.3 Load Silos

The Abox contains a memory reference pipeline that can accept a new load or store instruction every cycle until a Dcache fill is required. Since the Dcache lines are only allocated on load misses, the Abox can accept a new instruction every cycle until a load miss occurs. When a load miss occurs the Ibox stops issuing all instructions that use the load port of the register file or are otherwise handled by the Abox.

These instructions include LDL_L/LDQ_L, STL_C/STQ_C, HW_MTPR, HW_MFPR, FETCH, FETCH_M, RPCC, RS, RC, and MB. It also includes all memory format branch instructions, JMP, JSR, JSR_COROUTINE, and RET.

However, a JSR with a destination of R31 may be issued.

Because the result of each Dcache lookup is known late in the pipeline (stage [6]) and instructions are issued in pipe stage [3], there can be two instructions in the Abox pipeline behind a load instruction that misses the Dcache. These two instructions are handled as follows:

- Loads that hit the Dcache are allowed to complete—hit under miss.
- Load misses are placed in a silo and replayed in order 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 the silo and presented to the write buffer in order with respect to load misses.

To improve performance, the Ibox is allowed to restart the execution of Abox directed instructions before the last pending Dcache fill is complete. Dcache fill transactions result in four data transfers from the BIU to the Dcache. These transfers can each be separated by one or more cycles depending on the characteristics of the external cache and memory subsystems. The BIU attempts to send the quadword of the fill block that the CPU originally requested in the first of these four transfers (it is always able to accomplish this for reads that hit in the external cache). Therefore, the pending load instruction that requested the Dcache fill can complete before the Dcache fill finishes. Dcache fill data accumulates one quadword at a time into a "pending fill" latch, rather than being written into the cache array as it is received from the BIU. When the load miss silo is empty and the requested quadword for the last outstanding load miss is received, the Ibox resumes execution of Abox-directed instructions despite the still-pending Dcache fill. When the entire cache line has been received from the BIU, it is written into the Dcache data array whenever the array is not busy with a load or a store.

2.5.4 Write Buffer

The Abox contains a write buffer for two purposes.

- To minimize the number of CPU stall cycles by providing a high bandwidth (but finite) resource for receiving store data.

  This is required since the 21064/21064A can generate store data at the peak rate of one quadword every CPU cycle, which is greater than the rate at which the external cache subsystem can accept the data.

- To attempt to aggregate-store data into aligned 32-byte cache blocks to maximize the rate at which data may be written from the BIU into the external cache.

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

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

The write buffer has four entries; each has storage for up to 32 bytes. The buffer has a "head" pointer and "tail" pointer. The buffer puts new commands into empty tail entries and takes commands out of nonempty head entries. The head pointer increments when an entry is unloaded to the BIU, and the tail pointer increments when new data is put into the tail entry. The head and tail pointers only point to the same entry when the buffer has zero or four nonempty entries. If no writes ever merge with existing nonempty entries, the ordering of writes with respect to other writes will be maintained. The write buffer never reorders writes except to merge them into nonempty entries. Once a write merges into a nonempty slot, its "programmed" order is lost with respect to both writes in the same slot and writes in other slots.

The write buffer attempts to send its head entry off-chip by requesting the BIU 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 8-bit counter is cleared when one of the following conditions is met.
  - The write buffer is empty.
  - The write buffer unloads an entry.
  - **21064 only**—A write-merge operation is executed.
- The write buffer contains an MB, STQ_C or STL_C instruction.
- A load miss is pending to an address currently valid in the write buffer that requires the write buffer to be flushed. The write buffer is completely flushed regardless of which entry matches the address.

2.6 Fbox

The Fbox is on-chip, pipelined, and capable of executing both VAX and IEEE floating-point instructions. IEEE floating-point datatypes S_floating and T_floating are supported with all rounding modes except round to $+/-$ infinity, which can be provided in software. VAX floating-point datatypes F_floating and G_floating are fully supported with limited support for D_floating format.

The Fbox contains:

- A 32-entry, 64-bit floating-point register file (FRF in Figure 2–1)
- A user-accessible control register, FPCR, containing:
  - Dynamic Rounding Mode controls
  - Exception flag information

The Fbox can accept an instruction every cycle, with the exception of floating-point divide instructions. The latency for data dependent, non-divide instructions is six cycles. For detailed information on instruction timing, refer to Section 2.9.

21064 Inexact Flag

For divide instructions, the **21064** Fbox does not compute the inexact flag. Consequently, the INE exception flag in the FPCR register is never set for IEEE floating-point divide using the inexact enable (/I) qualifier. To deliver IEEE conforming exception behavior to the user, **21064** FPU hardware always traps on DIVS/SI and DIVT/SI instructions. The intent is for 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 to the user. The exception associated with DIV/SI and DIVT/SI is imprecise. Software must follow the rules specified by the Alpha AXP architecture associated with the software completion modifier to ensure that the trap handler can deliver correct behavior to the user.

21064A Inexact Flag

For divide instructions, the **21064A** Fbox computes the inexact flag setting FPCR [INE] if appropriate. The **21064A** traps on DIV/SI instructions only when the result is really inexact.

For IEEE compliance issues, see Section 2.7 and the *Alpha Architecture Reference Manual*.

2.6.1 Fbox Exception Handling

Exceptions generated by the Fbox are recorded in two places: the architecturally defined FPCR and the EXC_SUM register. The FPCR records the occurrence of all exceptions that are detected (except for software completion [SWC]), independent of whether the corresponding trap is enabled. This register can be cleared only by way of an explicit clear command, a write using MT_FPCR. The exception information it records is a summary of all exceptions that occurred since the last clear command.

If any exception is detected and the trap is enabled for that exception, the Fbox informs the Ibox. The Ibox records this information in the EXC_SUM register and initiates an arithmetic trap.

The FPCR contains an additional field called the Dynamic Rounding Mode (DYN) field. The Dynamic Rounding Mode field provides an alternate method for selecting the rounding mode used for IEEE-type instructions. If the rounding mode selected by the opcode is /D, then the rounding mode specified by the FPCR [59:58] is used.

Figure 2–3 shows the format of the FPCR implemented by the **21064** while Figure 2–4 shows the FPCR used by the **21064A**.

Figure 2–3 21064 Floating-Point Control Register (FPCR) Format

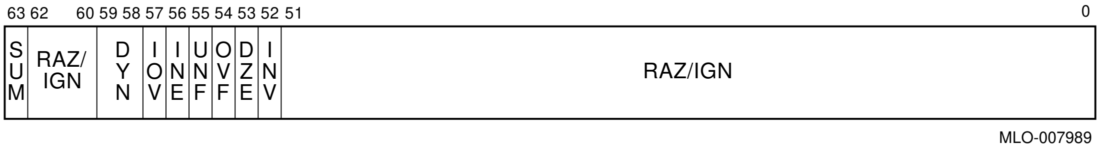


Figure 2–4 21064A Floating-Point Control Register (FPCR) Format

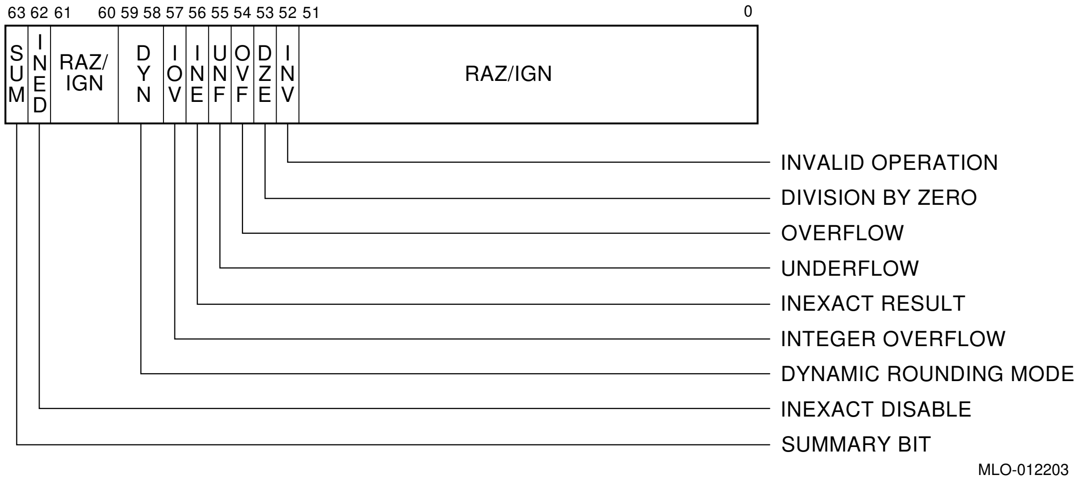


Table 2–2 lists the bit descriptions for the FPCR.

Table 2–2 Floating-Point Control Register Bit Descriptions

Bit	Description
63	Summary bit (SUM). Records bitwise OR of FPCR exception bits (FPCR bits [57:52]).
62	**21064A only**. Inexact Disable (INED). If this bit is set and a floating-point operation which enables trapping on inexact results generates an inexact value the trap is suppressed.
[61:60]	Reserved. Read as zero; ignored when written.
[59:58]	Dynamic rounding mode (DYN). Indicates the rounding mode to be used by an IEEE floating-point operate instruction when the instruction's function field specifies dynamic mode (/D). Assignments are:

DYN	IEEE Rounding Mode Selected
00	Chopped
01	Minus infinity
10	Normal rounding (nearest even)
11	Plus infinity

Bit	Description
57	Integer overflow (IOV). An integer arithmetic operation or a conversion from floating to integer overflowed the destination precision.
56	Inexact result (INE). A floating arithmetic or conversion operation gave a result that differed from the mathematically exact result.
55	Underflow (UNF). A floating arithmetic or conversion operation underflowed the destination exponent.
54	Overflow (OVF). A floating arithmetic or conversion operation overflowed the destination exponent.
53	Division by zero (DZE). An attempt was made to perform a floating divide operation with a divisor of zero.
52	Invalid operation (INV). Attempt was made to perform a floating arithmetic, conversion, or comparison operation, and one or more of the operand values were illegal.
[51:0]	Reserved. Read as zero; ignored when written.

2.7 IEEE Floating-Point Conformance

The 21064/21064A supports the IEEE floating-point operations as defined by the Alpha AXP architecture. Support for a complete implementation of the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Standard 754-1985) is provided by a combination of hardware and software as described in the *Alpha Architecture Reference Manual*.

Additional information that provides guidelines for writing code supporting precise exception handling (necessary for complete conformance to the Standard) is in the *Alpha Architecture Reference Manual*.

Information specific to the 21064/21064A follows:

- Invalid operation (INV)

  The invalid operation trap is always enabled. If the trap occurs, the destination register is UNPREDICTABLE. This exception is signaled if any VAX architecture operand is non-finite (reserved operand or dirty zero) and the operation can take an exception (that is, certain instructions, such as CPYS, never take an exception). This exception is signaled if any IEEE operand is non-finite (NAN, INF, denorm) and the operation can take an exception. This trap is also signaled for an IEEE format divide of +/- 0 divided by +/- 0. If the exception occurs, FPCR [INV] is set and the trap is signaled to the Ibox.

- Divide by zero (DZE)

  The divide-by-zero trap is always enabled. If the trap occurs, the destination register is UNPREDICTABLE. For VAX architecture format, this exception is signaled whenever the numerator is valid and the denominator is zero. For IEEE format, this exception is signaled whenever the numerator is valid and non-zero, with a denominator of +/- 0. If the exception occurs, FPCR [DZE] is set and the trap is signaled to the Ibox.

  For IEEE format divides, 0/0 signals INV, not DZE.

- Floating overflow (OVF)

  The floating overflow trap is always enabled. If the trap occurs, the destination register is UNPREDICTABLE. The exception is signaled if the rounded result exceeds in magnitude the largest finite number which can be represented by the destination format. This applies only to operations whose destination is a floating-point data type. If the exception occurs, FPCR [OVF] is set and the trap is signaled to the Ibox.

- Underflow (UNF)

  The underflow trap can be disabled. If underflow occurs, the destination register is forced to a true zero, consisting of a full 64 bits of zero. This is done even if the proper IEEE result would have been -0. The exception is signaled if the rounded result is smaller in magnitude than the smallest finite number that can be represented by the destination format. If the exception occurs, FPCR [UNF] is set. If the trap is enabled, the trap is signaled to the Ibox.

- Inexact (INE)

  The inexact trap can be disabled. The destination register always contains the properly rounded result, whether the trap is enabled. The exception is signaled if the rounded result is different from what would have been produced if infinite precision (infinitely wide data) were available. For floating-point results, this requires both an infinite precision exponent and fraction. For integer results, this requires an infinite precision integer. If the exception occurs, FPCR [INE] is set. If the trap is enabled, the trap is signaled to the Ibox.

  The IEEE-754 specification allows INE to occur concurrently with either OVF or UNF. Whenever OVF is signaled, (if the inexact trap is enabled) then INE is also signaled. Whenever UNF is signaled (if the inexact trap is enabled), then INE is also signaled. The inexact trap also occurs concurrently with integer overflow. All valid opcodes that enable INE also enable both overflow and underflow.

  If a CVTQL results in an integer overflow (IOV), FPCR [INE] is automatically set. (The INE trap is never signaled to the Ibox because there is no CVTQL opcode that enables the inexact trap.)

  DIVx/I behavior is slightly different. If the DIVx/I instruction does not take an input exception (that is, no INV or DZE), then the Fbox calculates and stores the correct rounded result.

  For the 21064—For DIVx without the /I qualifier FPCR [INE] is never set.

  For DIVx with the /I qualifier, FPCR [INE] is never set and an INE trap is always signaled to the Ibox regardless of whether the result is exact or inexact.
  For the 21064A—The Fbox calculates the inexact flag, setting FPCR [INE] if appropriate, and trapping on DVIx/SI instructions only when the result is really inexact.

- Integer overflow (IOV)

  The integer overflow trap can be disabled. The destination register always contains the low order bits ([64] or [32]) of the true result (not the truncated bits). Integer overflow can occur with CVTTQ, CVTGQ or CVTQL. In conversions from floating to quadword integer or to longword integer, an integer overflow occurs if the rounded result is outside the range -2^{63} ..$2^{63} - 1$. In conversions from quadword integer to longword integer, an integer overflow occurs if the result is outside the range -2^{31} ..$2^{31} - 1$. If the exception occurs, the appropriate bit in the FPCR is set. If the trap is enabled, the trap is signaled to the Ibox.

- Software completion (SWC)

  The software completion signal is not recorded in the FPCR. The state of software completion is recorded in the Exception Summary Register, EXC_SUM[SWC], described in Section 5.2.12.

Floating-point exceptions generated by the 21064/21064A are recorded in two places:

– The FPCR, as defined in the Alpha AXP architecture and accessible by the MT/MF_FPCR instructions, records the occurrence of all exception that are detected (except SWC), whether the corresponding trap is enabled (through the instruction modifiers). This register can only be cleared through an explicit clear command (MT_FPCR) so that the exception information it records is a summary of all exceptions that have occurred since the last clear.

– In addition, if an exception is detected and the corresponding trap enabled, the 21064/21064A records the condition in the EXC_SUM register and initiates an arithmetic trap.

  For the 21064—As a special case, to support inexact exception behavior with the DIVS/I and DIVT/I instructions, the **21064** always sets EXC_SUM [INE] during these instructions, although FPCR [INE] is never set. This behavior allows software emulation of the division instruction with accurate reporting of potential inexact exceptions.

  For the 21064A—The Fbox will calculate the inexact flag, setting FPCR [INE] if appropriate, and trapping on DVIx/SI instructions only when the result is really inexact.

Input exceptions always take priority over output exceptions. If both exception types occur, only the input exception is recorded in the FPCR and only the input exception is signaled to the Ibox.

2.8 Cache Organization

The 21064/21064A includes two on-chip caches, an instruction cache (Icache) and a data cache (Dcache). All memory cells in both Icache and Dcache are fully static six transistor CMOS structures.

2.8.1 21064/21064A Instruction Cache (Icache)

The instruction caches for the **21064** and the **21064A** are different so both are described here.

2.8.1.1 21064 Instruction Cache (Icache)

The **21064** Icache is an 8-KB, physical direct-mapped cache. Icache blocks, or lines, contain 32-bytes of instruction stream data with associated tag, plus a 6-bit ASN field (from the ICCSR IPR), a 1-bit ASM field (from the ITB_PTE IPR), and an 8-bit branch history field per block. It does not contain hardware for maintaining coherency with memory and is unaffected by the invalidate bus.

2.8.1.2 21064A Instruction Cache (Icache)

The **21064A** Icache is a 16-KB, physical direct-mapped cache that is addressed using VA 13 and **adr_h [12:5]**. An Icache block, or line, contains 32-bytes of instruction stream data with 8 data parity bits and a tag with one tag parity bit. The block also contains a 6-bit ASN field (from the ICCSR IPR), a 1-bit ASM field (from the ITB_PTE IPR), a 16-bit branch history field and a no data parity (Nodp) bit. It does not contain hardware for maintaining coherency with memory and is unaffected by the invalidate bus.

2.8.1.3 21064/21064A Icache Stream Buffer

The 21064/21064A also contains a single-entry Icache stream buffer that, together with its supporting logic, reduces the performance penalty due to Icache misses incurred during in-line instruction processing. Stream buffer prefetch requests never cross physical page boundaries, but instead wrap around to the first block of the current page.

2.8.2 21064/21064A Data Cache (Dcache)

The data caches for the **21064** and the **21064A** are different so both are described here.

2.8.2.1 21064 Data Cache (Dcache)

The **21064** Dcache contains 8 KB. It is a write-through, direct mapped, read allocate physical cache and has 32-byte blocks. System components can keep the Dcache coherent with memory by using the invalidate bus described in Section 6.5.3.

2.8.2.2 21064A Data Cache (Dcache)

The **21064A** 16 KB Dcache is a write-through, direct mapped, read allocated, physical tagged cache. Each block has 32-bytes and is addressed by VA 13 and **adr_h [12:5]**. External logic can keep the Dcache coherent with memory by using the invalidate bus described in Section 6.5.3.

The **21064A** Dcache has parity protection. Each cache line includes 8 data parity bits (one per LW) and a tag parity bit.

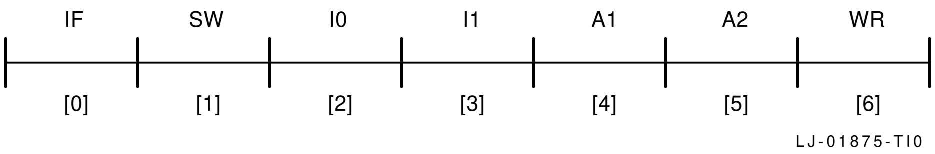
The **21064A** has both 8K byte and 16K byte Dcache modes. The mode is selected using ABOX_CTL [DC_16K].

2.9 Pipeline Organization

The 21064/21064A has a seven-stage pipeline for integer operate and memory reference instructions. Floating-point operate instructions progress through a ten-stage pipeline. The Ibox maintains state for all pipeline stages to track outstanding register writes, and determine Icache hit/miss.

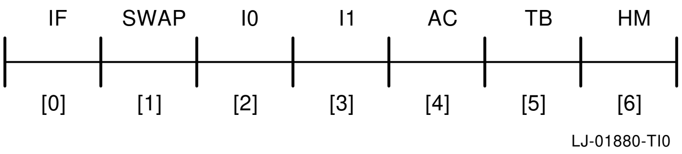
Figure 2–5 through Figure 2–7 show the integer operate, memory reference, and the floating-point operate pipelines for the Ibox, Ebox, Abox, and Fbox. The first four cycles are executed in the Ibox and the last stages are box specific. There are bypasses in all of the boxes that allow the results of one instruction to be used as operands of a following instruction without having to be written to the register file. Section 2.10 describes the pipeline scheduling rules.

Figure 2–5 Integer Operate Pipeline



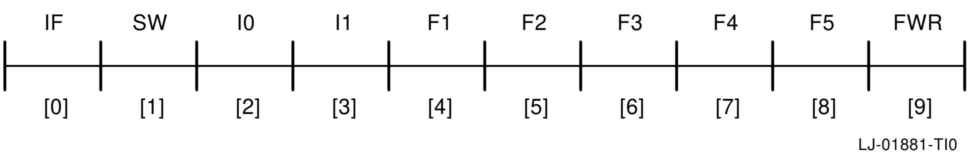
Stage 0: Instruction Fetch
Stage 1: Swap Dual Issue Instruction /Branch Prediction
Stage 2: Decode
Stage 3: Register file(s) access / Issue check
Stage 4: Computation cycle 1 / Ibox computes new PC
Stage 5: Computation cycle 2 / ITB look-up
Stage 6: Integer register file write / Icache Hit/Miss

Figure 2–6 Memory Reference Pipeline



Stages 0–3: Same
Stage 4: Abox calculates the effective Dstream address
Stage 5: DTB look-up
Stage 6: Dcache hit/miss and load data register file write

Figure 2–7 Floating-Point Operate Pipeline



Stages 0–3: Same
Stage 4–8: Floating-point calculate pipeline
Stage 9: Floating-point register file write

2.9.1 Static and Dynamic Stages

The 21064/21064A integer pipeline divides instruction processing into four static and three dynamic stages of execution. The 21064/21064A floating-point pipeline maintains the first four static stages and adds six dynamic stages of execution. The first four stages consist of:

- Instruction fetch
- Swap
- Decode
- Issue logic

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

Dynamic stages always advance state and are unaffected by any stall in the pipeline. (Pipeline stalls are also referred to as pipeline freezes.) A pipeline freeze may occur while zero instructions issue, or while one instruction of a pair issues and the second is held at the issue stage. A pipeline freeze implies that a valid instruction or instructions are presented to be issued but cannot proceed.

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

2.9.2 Aborts

Aborts can result from a number of causes. In general, they are grouped into two classes:

- Exceptions (including interrupts)
- Non-exceptions

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

The non-exception case, however, does not need to drain the pipeline of all outstanding instructions ahead of the aborting instruction. The pipeline can be immediately restarted at a redirected address. Examples of non-exception abort conditions are branch mispredictions, subroutine call/return mispredictions, and instruction cache misses. Data cache 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 as described. Due to the nature of some error conditions, this can occur as late as the write cycle. Next, the address of the excepting instruction is latched in the EXC_ADDR IPR. When the pipeline is fully drained, the processor begins instruction execution at the address given by the PALcode dispatch. The pipeline is drained when:

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

2.9.3 Non-Issue Conditions

There are two basic reasons for non-issue conditions.

- A pipeline freeze when a valid instruction or pair of instructions are prepared to issue but cannot due to a resource conflict

  This type of non-issue cycle can be minimized through code scheduling.

- Pipeline bubbles when there is no valid instruction in the pipeline to issue

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

2.10 Scheduling and Issuing Rules

Scheduling and issuing rules are covered in the sections that follow.

2.10.1 Instruction Class Definition

The scheduling and dual issue rules covered in this section are only performance related. There are no functional dependencies related to scheduling or dual issuing. The scheduling and issuing rules are defined in terms of instruction classes. Table 2–3 specifies all of the instruction classes and the box that executes the particular class.

Table 2–3 Producer-Consumer Classes

Class Name	Box	Instruction List
LD	Abox	All loads, (HW_MFPR, RPCC, RS, RC, STC producers only), (FETCH consumer only)
ST	Abox	All stores, HW_MTPR
IBR	Ebox	Integer conditional branches
FBR	Fbox	Floating-point conditional branches
JSR	Ebox	Jump to subroutine instructions JMP, JSR, RET, or JSR_ COROUTINE, (BSR, BR producer only)
IADDLOG	Ebox	ADDL ADDL/V ADDQ ADDQ/V SUBL SUBL/V SUBQ SUBQ /V S4ADDL S4ADDQ S8ADDL S8ADDQ S4SUBL S4SUBQ S8SUBL S8SUBQ LDA LDAH AND BIS XOR BIC ORNOT EQV
SHIFTCM	Ebox	SLL SRL SRA EXTQL EXTLL EXTWL EXTBL EXTQH EXTLH EXTWH MSKQL MSKLL MSKWL MSKBL MSKQH MSKLH MSKWH INSQL INSLL INSWL INSBL INSQH INSLH INSWH ZAP ZAPNOT CMOVEQ CMOVNE CMOVLT CMOVLE CMOVGT CMOVGE CMOVLBS CMOVLBC
ICMP	Ebox	CMPEQ CMPLT CMPLE CMPULT CMPULE CMPBGE
IMULL	Ebox	MULL MULL/V
IMULQ	Ebox	MULQ MULQ/V UMULH
FPOP	Fbox	Floating-point operates except divide
FDIV	Fbox	Floating-point divide

2.10.2 Producer-Consumer Latency

The 21064/21064A enforces the following issue rules regarding producer-consumer latencies.

The scheduling rules are described as a producer-consumer matrix, shown in Figure 2–8. Each row and column in the matrix is a class of Alpha AXP instructions. A number 1 in the Producer-Consumer Latency Matrix indicates one cycle of latency. A one cycle latency means that if instruction B uses the results of instruction A, then instruction B can be issued *one* cycle after instruction A is issued.

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

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

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

The SUBQ instruction consumes the result (R5) produced by the SRA instruction. The latency associated with a shift-iadd producer-consumer pair, as specified by the matrix, is two. That means that if the SRA was issued in cycle n, the SUBQ could be issued in cycle $n + 2$. The Ibox injects one NOP cycle in the pipeline for this case.

The final case has the STQ instruction consuming the result (R7) produced by the SUBQ instruction. The latency associated with an iadd-st producer-consumer pair, when the result of the iadd is the store data, is zero. This means that the SUBQ and STQ instruction pair can be dual-issued, if they were fetched in the same quadword.

Figure 2–8 Producer-Consumer Latency Matrix

Consumer \ Producer	LD (1)	JSR	IADDLOG	SHIFTCM	ICMP	IMULL (3)	IMULQ (3)	FPOP	FDIV F/S (4)	FDIV G/T (4)
LD	3	3	2	2	2	21	23	X	X	X
ST (2)	3	3	2/0	2/0	2/0	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	21	23	X	X	X
IADDLOG	3	3	1	2	2	21	23	X	X	X
SHIFTCM	3	3	1	2	2	21	23	X	X	X
ICMP	3	3	1	2	2	21	23	X	X	X
IMUL	3	3	1	2	2	21/19	23/21	X	X	X
FBR	3	X	X	X	X	X	X	6	34	63
FPOP	3	X	X	X	X	X	X	6	34	63
FDIV	3	X	X	X	X	X	X	6	34/30	63/59

LJ-01973-TI0

Notes:

1. For loads, Dcache hit is assumed. The latency for a Dcache miss is dependent on the system configuration.
2. For ST consumer class, some table entries contain 2 values in the form A/D. The A represents the latency for base address of store and the D represents the latency for store data. (D is known.) Floating-point results cannot be used as the base address for load or store operations.
3. For IMULL or IMULQ followed by IMUL in the form of Y/N, the Y represents the latency with data dependency; that is, the IMUL (N) uses the result from Y. N is the multiply latency without data dependencies (e.g. multiplier unit resource contention).
4. For FDIV followed by FDIV, there are two latencies given. The first represents the latency with data dependency; the second FDIV uses the result from the first. The second is the division latency without data dependencies.

X indicates an impossible state, or a state not encountered under normal circumstances. For example, a floating-point branch cannot consume data from an integer compare.

2.10.3 Producer-Producer Latency

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

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

2.10.4 Instruction Issue Rules

The following conditions prevent instruction issue:

- No instruction can be issued until all of its source and destination registers are clean; in other words, all outstanding writes to the destination register are guaranteed to complete in issue order and there are no outstanding writes to the source registers or those writes can be bypassed.
- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR, or BSR, BR, JSR (with destination other than R31) 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.

- No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR or BSR, BR, JSR (with destination other than R31) instruction can be issued when the Abox is busy due to a load miss or write buffer overflow. For more information see Section 2.5.3.
- No FDIV instruction can be issued if the floating-pointer divider is busy.
- No floating-point operate instruction can be issued exactly five or exactly six cycles before a floating-point divide completes.

2.10.5 Dual Issue Table

Table 2–4 can be used to determine instruction pairs that can issue in a single cycle. Instructions are dispatched using two internal data paths or buses. For more information about instructions and their opcodes and definitions, refer to the *Alpha Architecture Reference Manual.*

The buses are referred to in Table 2–4 as IB0, IB1, and IBx.

Any instruction identified with IB0 in the table can be issued in the same cycle as any instruction identified with IB1. An instruction that is identified as IBx 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 (Figure 2–8) and the following requirements are met:

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

Note

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

Table 2–4 Opcode Summary with Instruction Issue Bus

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

Key to Opcode Summary with Instruction Issue Bus

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

Table 2–4 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_{16}. The rows beneath the leftmost column supply the individual hex number to resolve that granularity.

If an instruction column has a 0 in the right (low) hex 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 additional information.

2.11 PALcode

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

The five opcodes (PAL19, PAL1B, PAL1D, PAL1E, PAL1F) are provided by the Alpha AXP architecture as implementation-specific privileged instructions. These instructions are defined independently for each Alpha AXP hardware implementation to provide PALcode software routines with access to specific hardware state and functions.

2.11.1 Architecturally Reserved PALcode Instructions

The hardware-specific instructions listed in Table 2–5 are executed in the PALcode environment. They produce OPCDEC exceptions (see Section 4.5 for a definition of OPCDEC) if executed while not in the PALcode environment. These instructions are mapped using the architecturally reserved opcodes (PAL19, PAL1B, PAL1D, PAL1E, PAL1F). They can only be used while executing chip-specific PALcode. See Section 4.8 for further information.

Table 2–5 Reserved PALcode Instructions (21064/21064A Specific)

Opcode	Mnemonic	Operation
PAL19	HW_MFPR	Move data from processor register
PAL1B	HW_LD	Load data from memory
PAL1D	HW_MTPR	Move data to processor register
PAL1E	HW_REI	Return from PALmode exception
PAL1F	HW_ST	Store data in memory

3
Instruction Set

3.1 Scope

This chapter provides information about the 21064/21064A instruction set. See the *Alpha AXP Architecture Handbook* for further information.

3.1.1 Instruction Summary

Table 3–1 provides the instruction format and opcode notation used in Table 3–2. All values are in hexadecimal radix.

Table 3–1 Instruction Format and Opcode Notation

Instruction Format	Format Symbol	Opcode Notation	Meaning
Branch	Bra	oo	oo is the 6-bit opcode field.
Floating-point	F-P	oo.fff	oo is the 6-bit opcode field. fff is the 11-bit function code field.
Memory	Mem	oo	oo is the 6-bit opcode field.
Memory/ func code	Mfc	oo.ffff	oo is the 6-bit opcode field. ffff is the 16-bit function code in the displacement field.
Memory/ branch	Mbr	oo.h	oo is the 6-bit opcode field. h is the high-order two bits of the displacement field.
Operate	Opr	oo.ff	oo is the 6-bit opcode field. ff is the 7-bit function code field.
PALcode	Pcd	oo	oo is the 6-bit opcode field; the particular PALcode instruction is specified in the 26-bit function code field.

Table 3–2 shows architecture instructions. Table 3–3 shows qualifiers for IEEE floating-point instructions and Table 3–4 shows qualifiers for VAX floating-point instructions.

Table 3–2 Architecture Instructions

Mnemonic	Format	Opcode	Description
ADDF	F-P	15.080	Add F_floating
ADDG	F-P	15.0A0	Add G_floating
ADDL	Opr	10.00	Add longword
ADDL/V		10.40	
ADDQ	Opr	10.20	Add quadword
ADDQ/V		10.60	
ADDS	F-P	16.080	Add S_floating
ADDT	F-P	16.0A0	Add T_floating
AND	Opr	11.00	Logical product
BEQ	Bra	39	Branch if $=$ zero
BGE	Bra	3E	Branch if $\geq$ zero
BGT	Bra	3F	Branch if $>$ zero
BIC	Opr	11.0	Bit clear
BIS	Opr	11.20	Logical sum
BLBC	Bra	38	Branch if low bit clear
BLBS	Bra	3C	Branch if low bit set
BLE	Bra	3B	Branch if $\leq$ zero
BLT	Bra	3A	Branch if $<$ zero
BNE	Bra	3D	Branch if $\neq$ zero
BR	Bra	30	Unconditional branch
BSR	Mbr	34	Branch to subroutine
CALL_PAL	Pcd	00	Trap to PALcode
CMOVEQ	Opr	11.24	CMOVE if $=$ zero
CMOVGE	Opr	11.46	CMOVE if $\geq$ zero
CMOVGT	Opr	11.66	CMOVE if $>$ zero
CMOVLBC	Opr	11.16	CMOVE if low bit clear
CMOVLBS	Opr	11.14	CMOVE if low bit set
CMOVLE	Opr	11.64	CMOVE if $\leq$ zero
CMOVLT	Opr	11.44	CMOVE if $<$ zero
CMOVNE	Opr	11.26	CMOVE if $\neq$ zero
CMPBGE	Opr	10.0F	Compare byte
CMPEQ	Opr	10.2D	Compare signed quadword equal
CMPGEQ	F-P	15.0A5	Compare G_floating equal

(continued on next page)

Table 3–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
CMPGLE	F-P	15.0A7	Compare G_floating less than or equal
CMPGLT	F-P	15.0A6	Compare G_floating less than
CMPLE	Opr	10.6D	Compare signed quadword less than or equal
CMPLT	Opr	10.4D	Compare signed quadword less than
CMPTEQ	F-P	16.0A5	Compare T_floating equal
CMPTLE	F-P	16.0A7	Compare T_floating less than or equal
CMPTLT	F-P	16.0A6	Compare T_floating less than
CMPTUN	F-P	16.0A4	Compare T_floating unordered
CMPULE	Opr	10.3D	Compare unsigned quadword less than or equal
CMPULT	Opr	10.1D	Compare unsigned quadword less than
CPYS	F-P	17.020	Copy sign
CPYSE	F-P	17.022	Copy sign and exponent
CPYSN	F-P	17.021	Copy sign negate
CVTDG	F-P	15.09E	Convert D_floating to G_floating
CVTGD	F-P	15.0AD	Convert G_floating to D_floating
CVTGF	F-P	15.0AC	Convert G_floating to F_floating
CVTGQ	F-P	15.0AF	Convert G_floating to quadword
CVTLQ	F-P	17.010	Convert longword to quadword
CVTQF	F-P	15.0BC	Convert quadword to F_floating
CVTQG	F-P	15.0BE	Convert quadword to G_floating
CVTQL	F-P	17.030	Convert quadword to longword
CVTQL/SV		17.530	
CVTQL/V		17.130	
CVTQS	F-P	16.0BC	Convert quadword to S_floating
CVTQT	F-P	16.0BE	Convert quadword to T_floating
CVTST	F-P	16.2AC	Convert S_floating to T_floating
CVTTQ	F-P	16.0AF	Convert T_floating to quadword
CVTTS	F-P	16.0AC	Convert T_floating to S_floating
DIVF	F-P	15.083	Divide F_floating
DIVG	F-P	15.0A3	Divide G_floating
DIVS	F-P	16.083	Divide S_floating
DIVT	F-P	16.0A3	Divide T_floating

(continued on next page)

Table 3–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
EQV	Opr	11.48	Logical equivalence
EXCB	Mfc	18.0400	Exception barrier
EXTBL	Opr	12.06	Extract byte low
EXTLH	Opr	12.6A	Extract longword high
EXTLL	Opr	12.26	Extract longword low
EXTQH	Opr	12.7A	Extract quadword high
EXTQL	Opr	12.36	Extract quadword low
EXTWH	Opr	12.5A	Extract word high
EXTWL	Opr	12.16	Extract word low
FBEQ	Bra	31	Floating branch if = zero
FBGE	Bra	36	Floating branch if $\geq$ zero
FBGT	Bra	37	Floating branch if > zero
FBLE	Bra	33	Floating branch if $\leq$ zero
FBLT	Bra	32	Floating branch if < zero
FBNE	Bra	35	Floating branch if $\neq$ zero
FCMOVEQ	F-P	17.02A	FCMOVE if = zero
FCMOVGE	F-P	17.02D	FCMOVE if $\geq$ zero
FCMOVGT	F-P	17.02F	FCMOVE if > zero
FCMOVLE	F-P	17.02E	FCMOVE if $\leq$ zero
FCMOVLT	F-P	17.02C	FCMOVE if < zero
FCMOVNE	F-P	17.02B	FCMOVE if $\neq$ zero
FETCH	Mfc	18.8000	Prefetch data
FETCH_M	Mfc	18.A000	Prefetch data, modify intent
INSBL	Opr	12.0B	Insert byte low
INSLH	Opr	12.67	Insert longword high
INSLL	Opr	12.2B	Insert longword low
INSQH	Opr	12.77	Insert quadword high
INSQL	Opr	12.3B	Insert quadword low
INSWH	Opr	12.57	Insert word high
INSWL	Opr	12.1B	Insert word low
JMP	Mbr	1A.0	Jump
JSR	Mbr	1A.1	Jump to subroutine
JSR_COROUTINE	Mbr	1A.3	Jump to subroutine return
LDA	Mem	08	Load address
LDAH	Mem	09	Load address high
LDF	Mem	20	Load F_floating
LDG	Mem	21	Load G_floating

(continued on next page)

Table 3–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
LDL	Mem	28	Load sign-extended longword
LDL_L	Mem	2A	Load sign-extended longword locked
LDQ	Mem	29	Load quadword
LDQ_L	Mem	2B	Load quadword locked
LDQ_U	Mem	0B	Load unaligned quadword
LDS	Mem	22	Load S_floating
LDT	Mem	23	Load T_floating
MB	Mfc	18.4000	Memory barrier
MF_FPCR	F-P	17.025	Move from FPCR
MSKBL	Opr	12.02	Mask byte low
MSKLH	Opr	12.62	Mask longword high
MSKLL	Opr	12.22	Mask longword low
MSKQH	Opr	12.72	Mask quadword high
MSKQL	Opr	12.32	Mask quadword low
MSKWH	Opr	12.52	Mask word high
MSKWL	Opr	12.12	Mask word low
MT_FPCR	F-P	17.024	Move to FPCR
MULF	F-P	15.082	Multiply F_floating
MULG	F-P	15.0A2	Multiply G_floating
MULL	Opr	13.00	Multiply longword
MULL/V		13.40	
MULQ	Opr	13.20	Multiply quadword
MULQ/V		13.60	
MULS	F-P	16.082	Multiply S_floating
MULT	F-P	16.0A2	Multiply T_floating
ORNOT	Opr	11.28	Logical sum with complement
RC	Mfc	18.E000	Read and clear
RET	Mbr	1A.2	Return from subroutine
RPCC	Mfc	18.C000	Read process cycle counter
RS	Mfc	18.F000	Read and set
S4ADDL	Opr	10.02	Scaled add longword by 4
S4ADDQ	Opr	10.22	Scaled add quadword by 4
S4SUBL	Opr	10.0B	Scaled subtract longword by 4
S4SUBQ	Opr	10.2B	Scaled subtract quadword by 4
S8ADDL	Opr	10.12	Scaled add longword by 8

(continued on next page)

Table 3–2 (Cont.) Architecture Instructions

Mnemonic	Format	Opcode	Description
S8ADDQ	Opr	10.32	Scaled add quadword by 8
S8SUBL	Opr	10.1B	Scaled subtract longword by 8
S8SUBQ	Opr	10.3B	Scaled subtract quadword by 8
SLL	Opr	12.39	Shift left logical
SRA	Opr	12.3C	Shift right arithmetic
SRL	Opr	12.34	Shift right logical
STF	Mem	24	Store F_floating
STG	Mem	25	Store G_floating
STS	Mem	26	Store S_floating
STL	Mem	2C	Store longword
STL_C	Mem	2E	Store longword conditional
STQ	Mem	2D	Store quadword
STQ_C	Mem	2F	Store quadword conditional
STQ_U	Mem	0F	Store unaligned quadword
STT	Mem	27	Store T_floating
SUBF	F-P	15.081	Subtract F_floating
SUBG	F-P	15.0A1	Subtract G_floating
SUBL	Opr	10.09	Subtract longword
SUBL/V		10.49	
SUBQ	Opr	10.29	Subtract quadword
SUBQ/V		10.69	
SUBS	F-P	16.081	Subtract S_floating
SUBT	F-P	16.0A1	Subtract T_floating
TRAPB	Mfc	18.0000	Trap barrier
UMULH	Opr	13.30	Unsigned multiply quadword high
WMB	Mfc	18.44	Write memory barrier
XOR	Opr	11.40	Logical difference
ZAP	Opr	12.30	Zero bytes
ZAPNOT	Opr	12.31	Zero bytes not

3.1.2 IEEE Floating-Point Instructions

Table 3–3 lists the hexadecimal value of the 11-bit function code field for the IEEE floating-point instructions, with and without qualifiers. The opcode for these instructions is 16_{16}.

Table 3–3 IEEE Floating-Point Instruction Function Codes

Mnemonic	None	/C	/M	/D	/U	/UC	/UM	/UD
ADDS	080	000	040	0C0	180	100	140	1C0
ADDT	0A0	020	060	0E0	1A0	120	160	1E0
CMPTEQ	0A5	–	–	–	–	–	–	–
CMPTLT	0A6	–	–	–	–	–	–	–
CMPTLE	0A7	–	–	–	–	–	–	–
CMPTUN	0A4	–	–	–	–	–	–	–
CVTQS	0BC	03C	07C	0FC	–	–	–	–
CVTQT	0BE	03E	07E	0FE	–	–	–	–
CVTST	See below							
CVTTQ	See below							
CVTTS	0AC	02C	06C	0EC	1AC	12C	16C	1EC
DIVS	083	003	043	0C3	183	103	143	1C3
DIVT	0A3	023	063	0E3	1A3	123	163	1E3
MULS	082	002	042	0C2	182	102	142	1C2
MULT	0A2	022	062	0E2	1A2	122	162	1E2
SUBS	081	001	041	0C1	181	101	141	1C1
SUBT	0A1	021	061	0E1	1A1	121	161	1E1

(continued on next page)

Table 3–3 (Cont.) IEEE Floating-Point Instruction Function Codes

Mnemonic	/SU	/SUC	/SUM	/SUD	/SUI	/SUIC	/SUIM	/SUID
ADDS	580	500	540	5C0	780	700	740	7C0
ADDT	5A0	520	560	5E0	7A0	720	760	7E0
CMPTEQ	5A5	–	–	–	–	–	–	–
CMPTLT	5A6	–	–	–	–	–	–	–
CMPTLE	5A7	–	–	–	–	–	–	–
CMPTUN	5A4	–	–	–	–	–	–	–
CVTQS	–	–	–	–	7BC	73C	77C	7FC
CVTQT	–	–	–	–	7BE	73E	77E	7FE
CVTTS	5AC	52C	56C	5EC	7AC	72C	76C	7EC
DIVS	583	503	543	5C3	783	703	743	7C3
DIVT	5A3	523	563	5E3	7A3	723	763	7E3
MULS	582	502	542	5C2	782	702	742	7C2
MULT	5A2	522	562	5E2	7A2	722	762	7E2
SUBS	581	501	541	5C1	781	701	741	7C1
SUBT	5A1	521	561	5E1	7A1	721	761	7E1

Mnemonic	None	/S
CVTST	2AC	6AC

Mnemonic	None	/C	/V	/VC	/SV	/SVC	/SVI	/SVIC
CVTTQ	0AF	02F	1AF	12F	5AF	52F	7AF	72F

Mnemonic	D	/VD	/SVD	/SVID	/M	/VM	/SVM	/SVIM
CVTTQ	0EF	1EF	5EF	7EF	06F	16F	56F	76F

3.1.3 VAX Floating-Point Instructions

Table 3–4 lists the hexadecimal value of the 11-bit function code field for the VAX floating-point instructions, with and without qualifiers. The opcode for these instructions is 15_{16}.

Table 3–4 VAX Floating-Point Instruction Function Codes

Mnemonic	None	/C	/U	/UC	/S	/SC	/SU	/SUC
ADDF	080	000	180	100	480	400	580	500
CVTDG	09E	01E	19E	11E	49E	41E	59E	51E
ADDG	0A0	020	1A0	120	4A0	420	5A0	520
CMPGEQ	0A5	–	–	–	4A5	–	–	–
CMPGLT	0A6	–	–	–	4A6	–	–	–
CMPGLE	0A7	–	–	–	4A7	–	–	–
CVTGF	0AC	02C	1AC	12C	4AC	42C	5AC	52C
CVTGD	0AD	02D	1AD	12D	4AD	42D	5AD	52D
CVTGQ	See below							
CVTQF	0BC	03C	–	–	–	–	–	–
CVTQG	0BE	03E	–	–	–	–	–	–
DIVF	083	003	183	103	483	403	583	503
DIVG	0A3	023	1A3	123	4A3	423	5A3	523
MULF	082	002	182	102	482	402	582	502
MULG	0A2	022	1A2	122	4A2	422	5A2	522
SUBF	081	001	181	101	481	401	581	501
SUBG	0A1	021	1A1	121	4A1	421	5A1	521

Mnemonic	None	/C	/V	/VC	/S	/SC	/SV	/SVC
CVTGQ	0AF	02F	1AF	12F	4AF	42F	5AF	52F

3.1.4 Required PALcode Function Codes

The opcodes listed in Table 3–5 are required for all Alpha AXP implementations. The notation used is oo.ffff, where oo is the hexadecimal 6-bit opcode and ffff is the hexadecimal 26-bit function code.

Table 3–5 Required PALcode Function Codes

Mnemonic	Type	Function Code
DRAINA	Privileged	00.0002
HALT	Privileged	00.0000
IMB	Unprivileged	00.0086

3.1.5 Opcodes Reserved for PALcode

The opcodes listed in Table 3–6 are reserved by the Alpha AXP architecture to be implementation specific. They are used by the 21064/21064A to implement PALcode as listed in Table 3–6. See Section 4.8 for more information.

Table 3–6 Opcodes Specific to the 21064/21064A

Architecture Mnemonic	Opcode	21064/21064A Mnemonic	Architecture Mnemonic	Opcode	21064 /21064A Mnemonic
PAL19	19	HW_MFPR	PAL1B	1B	HW_LD
PAL1D	1D	HW_MTPR	PAL1E	1E	HW_REI
PAL1F	1F	HW_ST	–	–	–

3.1.6 Opcodes Reserved for Digital

Table 3–7 lists the opcodes that are reserved for Digital.

Table 3–7 Opcodes Reserved for Digital

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
OPC01	01	OPC02	02	OPC03	03
OPC04	04	OPC05	05	OPC06	06
OPC07	07	OPC0A	0A	OPC0C	0C
OPC0D	0D	OPC0E	0E	OPC14	14

4 Privileged Architecture Library Code

4.1 Introduction

This chapter describes the 21064/21064A privileged architecture library code (PALcode). The chapter is organized as follows:

- Introduction
- PALcode
- PALmode Environment
- Invoking PALcode
- PALcode Environment Entry Points
- PALmode Restrictions
- Memory Management
- 21064/21064A Implementation of the Architecturally Reserved Opcodes Instructions

4.2 PALcode

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

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

- There are functions that must run atomically, yet involve long sequences of instructions that may need complete access to all the underlying computer hardware. An example of this is the sequence that returns from an exception or interrupt.
- There are some instructions that are necessary for backward compatibility or ease of programming; however, these are not used often enough to dedicate them to hardware, or are so complex that they would jeopardize the overall performance of the computer. For example, an interlocked memory access instruction might be familiar to someone used to programming on a CISC machine, but is not included in the Alpha AXP architecture. Another case is the emulation of an instruction that has no direct hardware support in a particular chip implementation.

In each of these cases, PALcode routines are used to provide the function. The routines are nothing more than programs invoked at specified times, and read in as I-stream code in the same way that all other Alpha AXP code is read. Once invoked, however, PALcode runs in a special mode.

4.3 PALmode Environment

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

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

One important aspect of PALcode is that it uses normal Alpha AXP instructions for most of its operations; that is, the same instruction set that non-privileged Alpha AXP programmers use. There are a few extra instructions that are only available in PALmode, and will cause a dispatch to the OPCDEC PALcode entry point if attempted while not in PALmode. The Alpha AXP architecture allows some flexibility in what these special PALmode instructions do. On the 21064/21064A the special PALmode-only instructions perform the following functions:

- Read or write internal processor registers (HW_MFPR, HW_MTPR)

- Perform memory load or store operations without invoking the normal memory management routines (HW_LD, HW_ST)
- Return from an exception or interrupt (HW_REI)

Refer to Section 4.8 for detailed information on these special PALmode instructions.

When executing in PALmode, there are certain restrictions for using the privileged instructions because PALmode gives the programmer complete access to many of the internal details of the 21064/21064A.

Caution

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

Refer to Section 4.6 for additional information on PALmode restrictions.

4.4 Invoking PALcode

PALcode is invoked at specific entry points, under certain well-defined conditions. These entry points provide access to a series of callable routines, with each routine indexed as an offset from a base address. The base address of the PALcode is programmable (stored in the PAL_BASE internal processor register), and is normally set by the system reset code. Refer to Section 4.5 for additional information on PALcode entry points.

When an event occurs that needs to invoke PALcode, the 21064/21064A first drains the pipeline. The current PC is loaded into the EXC_ADDR internal processor register, and the appropriate PALcode routine is dispatched. These operations occur under direct control of the chip hardware, and the machine is now in PALmode.

To exit PALcode a HW_REI instruction is executed at the end of the PALcode routine causing the hardware to execute a jump to the address contained in the EXC_ADDR internal processor register. The LSB is used to indicate PALmode to the hardware. Generally, upon return from a PALcode routine, the LSB is clear, in which case the hardware will load the new PC, enable interrupts, enable I-stream memory mapping, and dispatch back to the user.

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

- When the 21064/21064A is reset, it enters PALmode and executes the RESET PALcode. The system will remain in PALmode until a HW_REI instruction is executed with EXC_ADDR [0] clear. It then continues execution in non-PALmode (native mode), as just described. It is during this initial RESET PALcode execution that the rest of the low level system initialization is performed, including any modification to the PALcode base register.
- When a system hardware error is detected by the 21064/21064A, it invokes one of several PALcode routines, depending upon the type of error. Errors such as machine checks, arithmetic exceptions, reserved or privileged instruction decode, and data fetch errors are handled in this manner.
- When the 21064/21064A senses an interrupt, it dispatches to a PALcode routine that does the necessary information gathering, then handles the situation appropriately for the given interrupt.
- When a D-stream or I-stream translation buffer miss occurs, one of several PALcode routines is called to perform the TB fill. The memory management algorithms or even the existence of a virtual to physical page mapping is flexible. In the simplest case, this could be an automatic one-to-one translation from virtual to physical address. On a normal operating system these routines would consult page tables and perform the translation and fill based upon its contents.

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

4.4.1 CALL_PAL Instruction

As well as being invoked by hardware events, PALcode can be invoked under software control through the CALL_PAL instruction. This is a special instruction which causes an hardware exception. The hardware then dispatches to PALcode at a specific entry point using the same set of steps as the hardware-activated PALcode. That is, the pipeline is drained, the PC is saved, and the appropriate dispatch to an offset into the PALcode base is performed. The only difference is that the CALL_PAL instruction causes $PC + 4$ to be placed in the EXC_ADDR IPR. PALcode invoked by a CALL_PAL instruction does not increment the EXP_ADDR register before executing a HW_REI instruction to return to native mode. This feature requires special handling in the arithmetic trap and machine check PALcode flows. See Section 5.2.5, EXC_ADDR for more complete information.

The CALL_PAL instruction format includes a single parameter, the function field, that defines which PALcode routine to invoke. Only a subset of all the possible CALL_PAL function values are supported with hardware dispatches, in the 21064/21064A. These dispatches are described in Section 4.5. If a CALL_PAL instruction is executed and its function field is not supported by the 21064/21064A dispatch hardware, an OPDEC exception is taken.

There is a subtle difference between the two basic uses of PALcode: hardware-dispatched and CALL_PAL-dispatched. The hardware-invoked PALcode functions are necessary in some form for almost any useful computer system. For example, when the 21064/21064A detects a serious system error, it will dispatch to the machine check (MCHK) PALcode entry point. The exact PALcode that resides at this entry point can do whatever is reasonable, based upon system needs. In contrast, the functions invoked by CALL_PAL instructions are largely optional and based upon what the system implementation needs. CALL_PAL routines can perform different functions for different operating systems running on the 21064/21064A.

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

The 21064/21064A supports hardware dispatch for both privileged and non-privileged CALL_PAL instructions. That is, some of the functions that are passed to the CALL_PAL instruction are considered special. The designation of privileged or non-privileged refers to whether the user can call that particular CALL_PAL, and not the mode that it eventually runs in. Without exception, every CALL_PAL instruction will dispatch to PALcode that runs in PALmode. Privileged CALL_PAL instructions can only be successfully executed in kernel mode.

Privileged and non-privileged CALL_PAL instructions are dispatched in exactly the same way. When executed, they enter PALmode, do their function and return to the caller. Before execution a check is made to determine if the caller is in the correct mode. If code running in non-kernel mode attempts to execute a privileged CALL_PAL instruction, an OPCDEC PALcode routine is run instead of the CALL_PAL function.

4.5 PALcode Entry Points

Table 4–1 prioritizes entry points from highest to lowest priority; the first row in the table (reset) has the highest priority. The table defines only the entry point offset, bits [13:0]. The high-order bits of the new PC (bits [33:14]) come from the PAL_BASE register. The PAL_BASE register value at powerup is equal to zero.

Note

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

Table 4–1 PALcode Entry Points

Entry Name	Time	Offset(Hex)	Cause
RESET	anytime	0000	
MCHK	anytime	0020	Uncorrected hardware error.
ARITH	anytime	0060	Arithmetic exception.
INTERRUPT	anytime	00E0	Includes corrected hardware error.
D-stream errors	pipe_stage 6	01E0, 08E0, 09E0, 11E0	See Table 4–2.
ITB_MISS	pipe_stage 5	03E0	ITB miss.
ITB_ACV	pipe_stage 5	07E0	I-stream access violation.
CALL_PAL	pipe_stage 5	2000,2040,2080, 20C0 through 3FC0	128 locations based on instruction bits 7, 5..0.
OPCDEC	pipe_stage 5	13E0	Reserved or privileged opcode. Reserved opcodes are listed in Table 2–4 and marked RES. Privileged opcodes include both HW_x instructions and the privileged CALL_PAL instructions attempted when the processor is not in kernel mode (PS<CM1:CM0> not equal to 0).
FEN	pipe_stage 5	17E0	FP op attempted with: FP instructions disabled by way of the ICCSR FPE bit FP IEEE round to +/- infinity FP IEEE with datatype field other than S,T,Q

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

The 21064/21064A provides the first 64 privileged and 64 unprivileged CALL_PAL instructions with regions of 64 bytes. This produces hardware PALcode entry points described as follows:

```
Privileged CALL_PAL Instructions [00000000:0000003F]
Offset(Hex) = 2000 + ([5:0] shift left 6)
Unprivileged CALL_PAL Instructions [00000080:000000BF]
Offset(Hex) = 3000 + ([5:0] shift left 6)
```

The CALL_PAL instructions that do not fall within the ranges [00000000:0000003F] or [00000080:000000BF] result in an OPCDEC exception. CALL_PAL instructions that fall within the range [00000000:0000003F] while the 21064/21064A is not executing in kernel mode will result in an OPCDEC exception.

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

- Bad virtual address (incorrect sign extension)
- DTB miss
- Alignment error
- Access violation (ACV), Fault on read (FOR), Fault on write (FOW)

These errors get mapped into four PALcode entry points:

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

Table 4–2 lists the priority of these entry points as a group with respect to each of the other entry points. A particular D-stream memory reference may generate errors that fall into more than one of the four error classes which the hardware recognizes.

Table 4–2 D-stream Error PALcode Entry Points

BAD_VA	DTB_MISS	UNALIGN	PAL	Other	Offset(Hex)
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

4.6 PALmode Restrictions

Many of the PALmode restrictions involve waiting ''n'' cycles before using the results of a PALcode instruction. Inserting ''n'' instructions between the two time-sensitive instructions is the typical method of waiting for ''n'' cycles. Because the 21064/21064A can dual issue instructions it is possible to write code that requires $2 * n + 1$ instructions to wait ''n'' cycles. Due to the resource requirements of individual instructions and the 21064/21064A hardware design, multiple copies of the same instruction cannot be dual issued. This characteristic is used in some of the following examples. The following is a list of PALmode restrictions:

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

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

   - Multiple reads
   - Multiple writes
   - Read followed by write

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

   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 VPC, (VPC $[0] = 0$). By the previous rule, it is implied that at least three cycles of delay must be included after writing the ITB before executing a HW_REI instruction to exit PALmode.

Exceptions:

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

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

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

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

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

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

Note

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

Code for this operation is:

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

3. At least one cycle of delay must occur after a HW_MTPR TB_CTL before a HW_MTPR ITB_PTE or a HW_MFPR ITB_PTE. This must be done to allow setup of the ITB large page or small page decode.

4. The first cycle (the first one or two instructions) at all PALcode entry points can not execute a conditional branch instruction or any other instruction that uses the JSR stack hardware. This includes instructions:
   - JSR
   - JMP
   - RET
   - JSR_COROUTINE
   - BSR
   - HW_REI
   - All Bxx opcodes except BR

5. Table 4–3 lists the number of cycles required after a HW_MTPR instruction before a subsequent HW_REI instruction for the specified IPRs. These cycles can be ensured by inserting one HW_MFPR R31,0 instruction or other appropriate instruction(s) for each cycle of delay required after the HW_MTPR.

Table 4–3 HW_MTPR Restrictions

IPR	Cycles Between HW_MTPR and HW_REI
DTBIS,ASM,ZAP	0
ITBIS,ASM,ZAP	2
xIER	3
xIRR	3
ICCSR [FPE]	4
PS	5

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

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

8. Three cycles of delay are required between:
   - HW_MTPR xIER and HW_MFPR xIRR
   - HW_MTPR xIRR 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
9. The following operations are disabled in the cycle immediately following a HW_REI instruction:
   - HW_MxPR ITB_TAG
   - HW_MxPR ITB_PTE
   - HW_MxPR ITB_PTE_TEMP

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

   - An exception interrupts flow of the ITB miss routine and attempts to REI back.

- The return address begins with a HW_MxPR instruction to an ITB register.
- The REI is predicted correctly to avoid any delay between the two instructions.

The code for this operation is:

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

10. The following registers can only be accessed in PALmode:

    - TB_TAG
    - ITB_PTE
    - ITB_PTE_TEMP

    If the instruction HW_MTPR or HW_MFPR is to or from the previous mentioned registers while not in PALmode by setting the HWE (hardware enable) bit of the ICCSR, the instructions will be ignored.

11. When writing the PAL_BASE register, exceptions must be prevented. An exception occurring simultaneously with a write to the PAL_BASE can leave the register in a metastable state. All asynchronous exceptions but reset can be avoided under the following conditions:

```
PALmode .................... blocks all interrupts
machine checks disabled ..... blocks I/O error exceptions
(by way of the ABOX_CTL reg or MB isolation)
Not under trap shadow ....... avoids arithmetic traps

The trap shadow is defined as:

less than  3 cycles after a non-mul integer operate that may
overflow
less than 22 cycles after a MULL/V instruction
less than 24 cycles after a MULQ/V instruction
less than  6 cycles after a non-div fp operation that may cause
a trap
less than 34 cycles after a DIVF or DIVS that may cause a trap
less than 63 cycles after a DIVG or DIVT that may cause a trap
```

12. The sequence HW_MTPR PTE, HW_MTPR TAG is not allowed. At least two null cycles must occur between HW_MTPR xxx_PTE and HW_MTPR TB_TAG.

13. 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.

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

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

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

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

16. The content of the destination register for HW_MFPR Rx,DTB_PTE or HW_MFPR Rx,ITB_PTE is UNPREDICTABLE.

17. Two HW_MFPR DTB_PTE instructions cannot be issued in consecutive cycles. This implies that more than one instruction can be necessary between the HW_MFPR instructions if dual issue is possible. Similar restrictions apply to the ITB_PTE register.

18. Reading the EXC_SUM and BC_TAG registers require special timing. Refer to Section 5.2.12 and Section 5.3.22 for specific information.

19. DMM errors occurring one cycle before HW_MxPR instructions to the ITB_PTE will not stop the TB pointer from incrementing to the next TB entry even though the HW_MxPR instruction will be aborted by the DMM error. This restriction only affects performance and not functionality.

20. PALcode that writes multiple ITB entries must write the entry that maps the address contained in the EXC_ADDR register last.

21. HW_STC instructions cannot be followed, for two cycles, by any load instruction that may miss in the Dcache.

22. Updates to the ASN field of the ICCSR IPR require at least 10 cycles of delay before entering native mode that can reference the ASN during Icache access. If the ASN field is updated in kernel mode by way of the HWE bit of the ICCSR IPR, it is sufficient that all I-stream references during this time be made to pages with the ASM bit set to avoid use of the ASN.

23. HW_MTPR instructions that update the TB_CTL register cannot follow a HW_MTPR instruction that updates the DTB_PTE or ITB_PTE register by one cycle.

24. The HW_MTPR instructions that update the following IPRs require delays after the instruction as shown in Table 4–4:

    - ICCSR (ASN field)
    - FLUSH_IC
    - FLUSH_IC_ASM

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

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

Table 4–4 HW_MTPR Cycle Delay

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

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

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

Machine check service routines should check for MxPR instructions at the return address and return to the instruction following the HW_MxPR.

Note

When writing PALcode, the PALcode Violation Checker (PVC) software tool should be used to verify that the PALcode does not violate any of these restrictions.

4.7 Memory Management

Memory management is supported in PALcode. Hardware support for memory management includes the Data Translation Buffer (DTB) and ITB, which are each capable of up to four protection modes. Hardware support consists of virtual (up to 43 bits) to physical (up to 34 bits) address translation.

4.7.1 TB Miss Flows

This section describes hardware specific details to aid the PALcode programmer in writing ITB and DTB fill routines. These flows highlight the tradeoffs and restrictions between PALcode and hardware. The PALcode source that is released with the 21064/21064A should be consulted before any new flows are written. A working knowledge of the Alpha AXP memory management architecture is assumed. Refer to the *Alpha Architecture Reference Manual* for additional information about the Alpha AXP memory management architecture. Also see Section 5.3.4.

4.7.1.1 ITB Miss

When the Ibox encounters an ITB miss it:

1. Latches the VPC of the target instruction-stream reference in the EXC_ADDR IPR.
2. Flushes the pipeline of any instructions following the instruction which caused the ITB miss.
3. Waits for any other instructions which may be in progress to complete.

4. Enters PALmode.
5. Jumps to the ITB miss PALcode entry point.

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

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

Note

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

8. Restore the contents of any modified integer registers from the PAL_TEMP register file.
9. Restart the instruction stream using the HW_REI instruction.

4.7.1.2 DTB Miss

When the Abox encounters a DTB miss it:

1. Latches the referenced virtual address in the VA IPR and other information about the reference in the MM_CSR IPR.

   Locks the VA and MM_CSR registers against further modifications.

   Latches the PC of the instruction that generated the reference in the EXC_ADDR register.

2. Drains the machine as described in Section 4.7.1.1 (steps 2 and 3).
3. Jumps to the DTB miss PALcode entry point.

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

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

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

Note

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

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

4.8 21064/21064A Implementation of the Architecturally Reserved Opcodes Instructions

PALcode uses the Alpha AXP instruction set for most of its operations. The 21064/21064A maps the architecturally reserved opcodes PAL19, PAL1B, PAL1D, PAL1E, and PAL1F to:

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

These instructions are described further in Table 4–5. They produce an OPCDEC exception if executed while not in the PALmode environment. If the HWE bit of the ICCSR internal processor register (IPR) is set, these instructions can be executed in kernel mode.

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

Table 4–5 Instructions Specific to the 21064/21064A

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

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

Note

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

4.8.1 HW_MFPR and HW_MTPR Instructions

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

Caution

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

Coding restrictions (see Section 4.6) are associated with accessing various registers. Separate bits are used to access the:

- Abox IPRs
- Ibox IPRs
- PAL_TEMPs

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

Figure 4–1 shows the HW_MFPR and HW_MTPR instruction format. Table 4–6 lists the instruction fields.

Figure 4–1 HW_MFPR and HW_MTPR Instruction Format

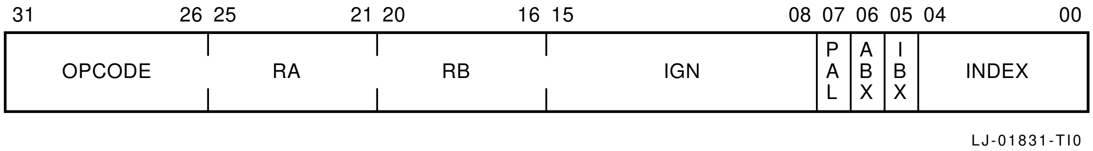


Table 4–6 HW_MFPR and HW_MTPR Format Description

Field	Description
OPCODE	Is either 25 (HW_MFPR) or 29 (HW_MTPR).
RA/RB	Contain the source (HW_MTPR) or destination (HW_MFPR) register number. The RA and RB fields must always be identical.
PAL	If set, this HW_MFPR or HW_MTPR instruction is referencing a PAL temporary register, PAL_TEMP.
ABX	If set, this HW_MFPR or HW_MTPR instruction is referencing a register in the Abox.
IBX	If set, this HW_MFPR or HW_MTPR instruction is referencing a register in the Ibox.

Table 4–7 indicates how the PAL, ABX, IBX, and INDEX fields are set to access the IPRs. Setting the PAL, ABX, and IBX fields to zero generates a NOP.

Table 4–7 Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
TB_TAG	x	x	1	0	W	PALmode only
ITB_PTE	x	x	1	1	R/W	PALmode only
ICCSR	x	x	1	2	R/W	
ITB_PTE_TEMP	x	x	1	3	R	PALmode only
EXC_ADDR	x	x	1	4	R/W	

(continued on next page)

Table 4–7 (Cont.) Internal Processor Register Access

Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
SL_RCV	x	x	1	5	R	
ITBZAP	x	x	1	6	W	PALmode only
ITBASM	x	x	1	7	W	PALmode only
ITBIS	x	x	1	8	W	PALmode only
PS	x	x	1	9	R/W	
EXC_SUM	x	x	1	10	R/W	
PAL_BASE	x	x	1	11	R/W	
HIRR	x	x	1	12	R	
SIRR	x	x	1	13	R/W	
ASTRR	x	x	1	14	R/W	
HIER	x	x	1	16	R/W	
SIER	x	x	1	17	R/W	
ASTER	x	x	1	18	R/W	
SL_CLR	x	x	1	19	W	
SL_XMIT	x	x	1	22	W	
TB_CTL	x	1	x	0	W	
DTB_PTE	x	1	x	2	R/W	
DTB_PTE_TEMP	x	1	x	3	R	
MM_CSR	x	1	x	4	R	
VA	x	1	x	5	R	
DTBZAP	x	1	x	6	W	
DTBASM	x	1	x	7	W	
DTBIS	x	1	x	8	W	
BIU_ADDR	x	1	x	9	R	
BIU_STAT	x	1	x	10	R	
DC_STAT	x	1	x	12	R	
FILL_ADDR	x	1	x	13	R	

(continued on next page)

Table 4–7 (Cont.) Internal Processor Register Access

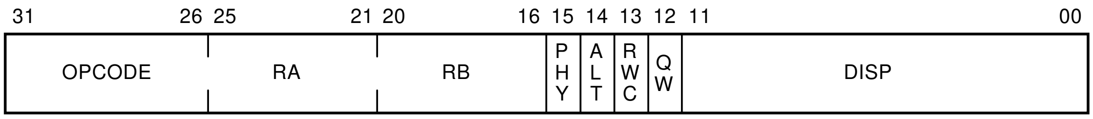
Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
ABOX_CTL[1]	x	1	x	14	W	
ALT_MODE	x	1	x	15	W	
CC	x	1	x	16	W	
CC_CTL	x	1	x	17	W	**21064 only**
DC_CTL	x	1	x	17	W	**21064A only**
BIU_CTL[1]	x	1	x	18	W	
FILL_SYNDROME	x	1	x	19	R	
BC_TAG	x	1	x	20	R	
FLUSH_IC	x	1	x	21	W	
FLUSH_IC_ASM	x	1	x	23	W	
PAL_TEMP [31..0]	1	x	x	31-00	R/W	

[1]Versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2 did not implement ABOX_CTL [9:7] and BIU_CTL [43, 42:40, 38]. PALcode for these processors is upward compatible if the PALcode did not set these bits.

4.8.2 HW_LD and HW_ST Instructions

PALcode uses the HW_LD and HW_ST instructions to access memory outside of the realm of normal Alpha AXP memory management. Figure 4–2 shows the HW_LD and HW_ST instructions format. Table 4–8 lists the instruction fields.

Figure 4–2 HW_LD and HW_ST Instructions Format



LJ-01832-TI0

The effective address of these instructions is calculated as follows:

```
addr <- (SEXT(DISP) + RB) AND NOT (QW | 11 (bin))
```

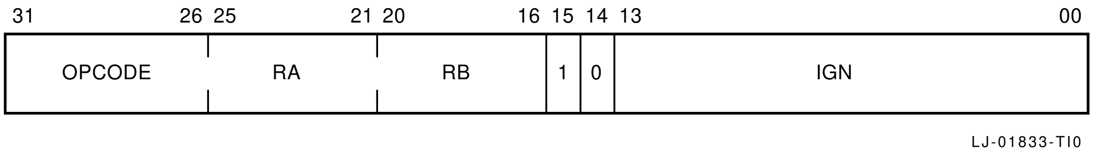
Table 4–8 HW_LD and HW_ST Format Description

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

4.8.3 HW_REI Instruction

The HW_REI instruction uses the address in the Ibox EXC_ADDR register to determine the new virtual program counter (VPC). Bit [0] of the EXC_ADDR determines the state of the PALmode bit on the completion of the HW_REI. If EXC_ADDR bit [0] is clear, then the processor returns to non-PALmode. If EXC_ADDR bit [0] is set, then 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 PALcode instruction flows to take advantage of the D-stream mapping hardware in the 21064/21064A, including traps. Figure 4–3 shows the HW_REI instruction format. Table 4–9 lists the instruction fields.

Figure 4–3 HW_REI Instruction Format



Bits [15:14] contain the branch prediction hint bits. The 21064/21064A pushes the contents of the EXC_ADDR register on the JSR prediction stack. Bit [15] must be set to pop the stack to avoid misalignment.

The next address and PALmode bit are calculated as follows:

```
VPC <- EXC_ADDR AND {NOT 3}

PALmode <- EXC_ADDR[0]
```

Table 4–9 The HW_REI Format Description

Field	Description
OPCODE	The OPCODE field contains 30.
RA/RB	Contain register numbers which should be R31 or a stall may occur.

4.8.4 Required PALcode Instructions

The PALcode instructions listed in Table 4–10 are described in the *Alpha Architecture Reference Manual*.

Table 4–10 Required PALcode Instructions

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

5

Internal Processor Registers

5.1 Introduction

This chapter describes the internal processor registers of the 21064/21064A microprocessor. This chapter is organized as follows:

- Ibox and Abox Internal Processor Registers
- PAL_TEMP Registers
- Lock Registers
- Internal Processor Registers Reset State

5.2 Ibox Internal Processor Registers

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

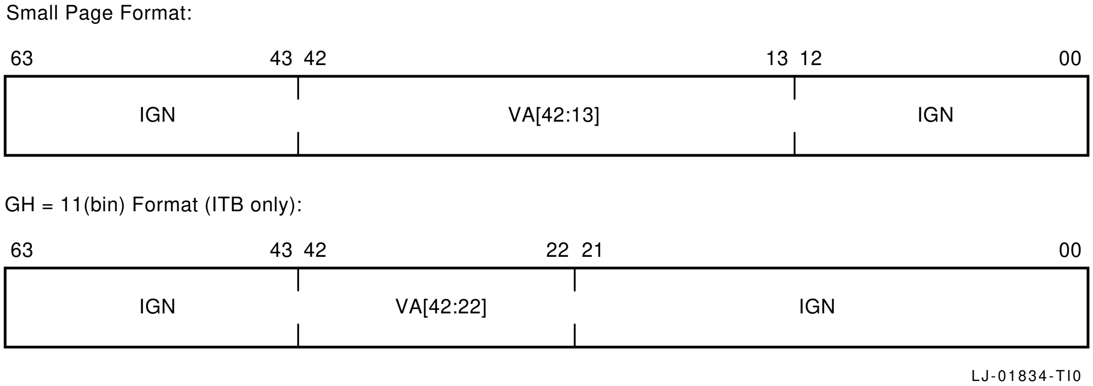
5.2.1 Translation Buffer Tag Register (TB_TAG)

The TB_TAG register is a write-only register that holds the tag for the next translation buffer update operation in the Instruction Translation Buffer (ITB) or the Data Translation Buffer (DTB). The tag is written to a temporary register and not transferred to the ITB or DTB until the Instruction Translation Buffer Page Table Entry (ITB_PTE) or the Data Translation Buffer Page Table Entry (DTB_PTE) register is written. The entry to be written is chosen at the time of the ITB_PTE or DTB_PTE write operation by a not-last-used algorithm, implemented in hardware. Figure 5–1 shows the TB_TAG register format.

Note

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

Figure 5–1 Translation Buffer Tag Register



5.2.2 Instruction Translation Buffer Page Table Entry Register (ITB_PTE)

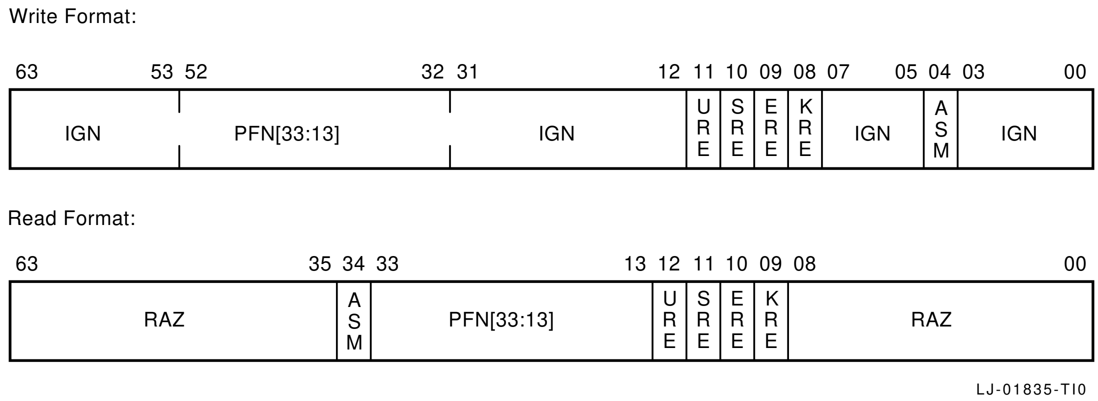
The ITB_PTE register is a read/write register, representing twelve page table entries split into two distinct arrays. The first eight page table entries provide small page (8K byte) translations while the remaining four provide large page (4 MB) translations. The entry to be written is chosen by a not-last-used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

The ITB's tag array is updated simultaneously from the TB_Tag register when the ITB_PTE register is written. Reads of the ITB_PTE register require two instructions. The first instruction sends the PTE data to the Instruction Translation Buffer Page Table Entry Temporary register (ITB_PTE_TEMP) and the second instruction, reading from the ITB_PTE_TEMP register, returns the PTE entry to the register file. Reading or writing the ITB_PTE register increments the TB entry pointer corresponding to the large/small page selection indicated by the TB_CTL, which allows reading the entire set of ITB_PTE register entries. Figure 5–2 shows the ITB_PTE register format.

Note

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

Figure 5–2 Instruction Translation Buffer Page Table Entry Register



5.2.3 Instruction Cache Control and Status Register (ICCSR)

The ICCSR register contains various Ibox hardware enables. The only architecturally defined bit in this register is the floating-point enable (FPE), which enables floating-point instructions. When cleared, all floating-point instructions generate exceptions to the FEN entry point in PALcode (see Table 4–1. Most of this register is cleared by hardware at reset. Fields that are not cleared at reset include ASN, PC0, and PC1.

The hardware enable bit allows the special privileged architecture library code (PALcode) instructions to execute in kernel mode. This bit is intended for diagnostic or operating system alternative PALcode routines only. It does not allow access to the ITB registers if not running in PALmode. Figure 5–3 shows the ICCSR register format.

Table 5–1 lists the ICCSR register fields and a brief description. Table 5–2 lists branch states controlled by the Branch Prediction Enable (BPE) and Branch History Enable (BHE) bits in the ICCSR.

Figure 5–3 ICCSR Register

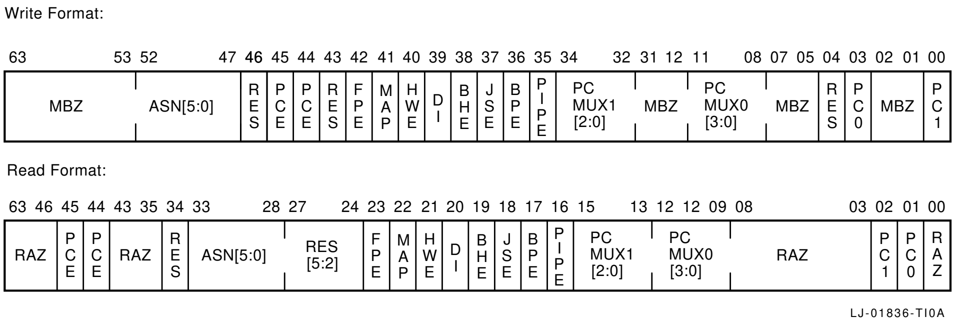


Table 5–1 ICCSR Fields and Description

Field	Type	Description
ASN	RW	The ASN field is used in conjunction with the Icache to further qualify cache entries and avoid some cache flushes. The ASN is written to the Icache during fill operations and compared with the I-stream data on fetch operations. Mismatches invalidate the fetch without affecting the Icache. (See the *Alpha Architecture Reference Manual*.)
RES	RW,0	The RES state bits are reserved by Digital and should not be used by software.
PCE	RW	If both of these bits are clear, they disable both performance counters. If either bit is set, both performance counters will increment in their usual fashion.
FPE	RW,0	If set, floating-point instructions can be issued. If clear, floating-point instructions cause FEN exceptions.
MAP	RW,0	If set, it allows superpage I-stream memory mapping of virtual PC [33:13] directly to Physical PC [33:13] essentially bypassing ITB for virtual PC addresses containing virtual PC [42:41] = 2. Superpage mapping is allowed in kernel mode only. The Icache ASM bit is always set. If clear, superpage mapping is disabled.

(continued on next page)

Table 5–1 (Cont.) ICCSR Fields and Description

Field	Type	Description
HWE	RW,0	If set, it allows the five reserved opcodes (PAL19, PAL1B, PAL1D, PAL1E, and PAL1F) instructions to be issued in kernel mode. If cleared, attempts to execute reserved opcodes instructions while not in PALmode result in OPCDEC exceptions.
DI	RW,0	If set, it enables dual issue. If cleared, instructions can only single issue.
BHE	RW,0	Used in conjunction with BPE. See Table 5–2 for programming information.
JSE	RW,0	If set, it enables the JSR stack to push a return address. If cleared, JSR stack is disabled.
BPE	RW,0	Used in conjunction with BHE. See Table 5–2 for programming information.
PIPE	RW,0	If clear, it causes all hardware interlocked instructions to drain the machine and waits for the write buffer to empty before issuing the next instruction. Examples of instructions that do not cause the pipe to drain include HW_MTPR, HW_REI, conditional branches, and instructions that have a destination register of R31. If set, pipeline proceeds normally.
PCMUX1	RW,0	See Table 5–4 for programming information.
PCMUX0	RW,0	See Table 5–3 for programming information.
PC1	RW	If clear, it enables performance counter 1 interrupt request after 2^{12} events counted. If set, enables performance counter 1 interrupt request after 2^{8} events counted.
PC0	RW	If clear, it enables performance counter 0 interrupt request after 2^{16} events counted. If set, it enables performance counter 0 interrupt request after 2^{12} events counted.

Note

Using the HW_MTPR instruction to update the EXC_ADDR register while in the native mode is restricted to bit [0] being equal to 0. The combination of the native mode and EXC_ADDR bit [0] being equal to one causes UNDEFINED behavior. This combination is only possible through the use of the HWE bit.

Table 5–2 BHE, BPE Branch Prediction Selection (Conditional Branches Only)

BPE	BHE	Prediction
0	X	Not Taken
1	0	Sign of Displacement
1	1	Branch History Table

5.2.3.1 Performance Counters

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

The performance counters may be enabled or disabled using ICCSR [45:44] (PCE [1:0]).

Since the counters continuously accumulate selected events, despite interrupts being enabled, the first interrupt after selecting a new counter input has an error bound as large as the selected overflow range. Some inputs can over count events occurring simultaneously with D-stream errors that abort the actual event very late in the pipeline.

For example, when counting load instructions, attempts to execute a load resulting in a TB miss exception will increment the performance counter after the first aborted execution attempt and again after the TB fill routine when the load instruction reissues and completes.

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

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

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

Table 5–3 lists performance counter 0 inputs and Table 5–4 lists performance counter 1 inputs.

Table 5–3 Performance Counter 0 Input Selection (in ICCSR)

MUX0 [3:0]	Input	Comment
000X	Total Issues/2	Counts total issues divided by 2, dual issue increments count by 1.
001X	Pipeline Dry	Counts cycles where nothing issued due to lack of valid I-stream data. Causes include Icache fill, misprediction, branch delay slots, and pipeline drain for exception.
010X	Load Instructions	Count all Load instructions.
011X	Pipeline Frozen	Counts cycles where nothing issued due to resource conflict.
100X	Branch Instructions	Counts all conditional branches, unconditional branches, JSR, and HW_REI instructions.
1011	PALmode	Counts cycles while executing in PALmode.
1010	Total cycles	Counts total cycles.
110X	Total Non-issues/2	Counts total non-issues divided by 2 ("no issue" increments count by 1).
111X	PERF_CNT_H [0]	Counts external events supplied to a pin at a selected system clock cycle interval.

Table 5–4 Performance Counter 1 Input Selection (in ICCSR)

MUX1 [2:0]	Input	Comment
000	Dcache miss	Counts total Dcache misses.
001	Icache miss	Counts total Icache misses.
010	Dual issues	Counts cycles of Dual issue.
011	Branch Mispredicts	Counts both conditional branch mispredictions and JSR or HW_REI mispredictions. Conditional branch mispredictions cost 4 cycles and others cost 5 cycles of dry pipeline delay.
100	FP Instructions	Counts total floating-point operate instructions, that is, no FP branch, load, or store.
101	Integer Operate	Counts integer operate instructions including LDA and LDAH with destination other than R31.
110	Store Instructions	Counts total store instructions.
111	PERF_CNT_H [1]	Counts external events supplied to a pin at a selected system clock cycle interval.

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

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

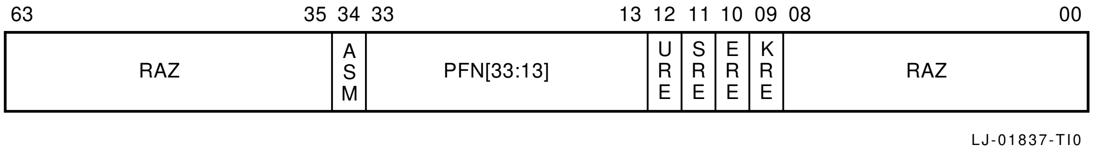
1. Read the ITB_PTE register data to the ITB_PTE_TEMP register.
2. Read the ITB_PTE_TEMP register data to the integer register file.

The ITB_PTE_TEMP register is updated on all ITB accesses, both read and write. A read of the ITB_PTE to the ITB_PTE_TEMP should be followed closely by a read of the ITB_PTE_TEMP to the register file. Figure 5–4 shows the ITB_PTE_TEMP register format.

Note

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

Figure 5–4 ITB_PTE_TEMP Register



5.2.5 Exceptions Address Register (EXC_ADDR)

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

The HW_REI instruction executes a jump to the address contained in the EXC_ADDR register. The EXC_ADDR register is written by hardware after an exception to provide a return address for PALcode.

The instruction pointed to by the EXC_ADDR register did not complete its execution. The LSB of the EXC_ADDR register is used to indicate PALmode to the hardware. When the LSB is clear, the HW_REI instruction executes a jump to native (non-PAL) mode, enabling address translation.

CALL_PAL exceptions load the EXC_ADDR with the PC of the instruction following the CALL_PAL. This function allows CALL_PAL service routines to return without needing to increment the value in the EXC_ADDR register.

This feature requires careful treatment in PALcode. Arithmetic traps and machine check exceptions can preempt CALL_PAL exceptions resulting in an incorrect value being saved in the EXC_ADDR register. In the cases of an arithmetic trap or machine check exception (only in these cases), EXC_ADDR [1] takes on special meaning. PALcode servicing these two exceptions must:

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

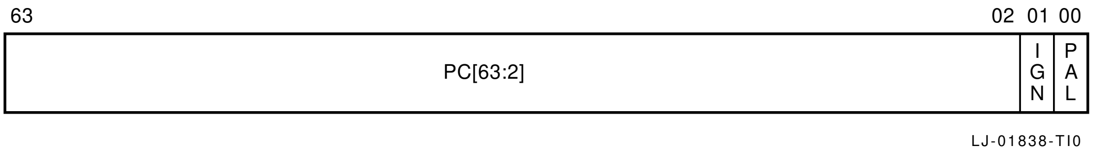
All other PALcode entry points except reset can expect EXC_ADDR [1] to be 0.

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

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

Figure 5–5 shows the exception address register format.

Figure 5–5 Exception Address Register



5.2.6 Clear Serial Line Interrupt Register (SL_CLR)

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

- Serial line interrupt request
- Performance counter interrupt requests
- CRD interrupt request

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

Figure 5–6 Clear Serial Line Interrupt Register

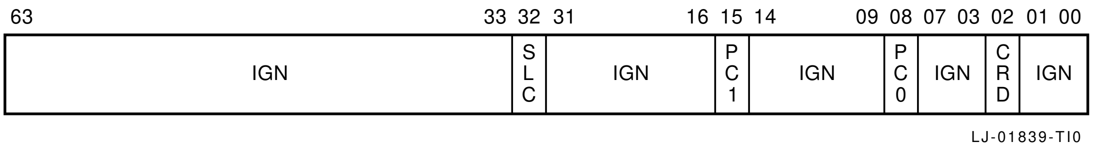


Table 5–5 Clear Serial Line Interrupt Register Fields

Field	Type	Description
CRD	W0C	Clears the correctable read error interrupt request
PC1	W0C	Clears the performance counter 1 interrupt request
PC0	W0C	Clears the performance counter 0 interrupt request
SLC	W0C	Clears the serial line interrupt request

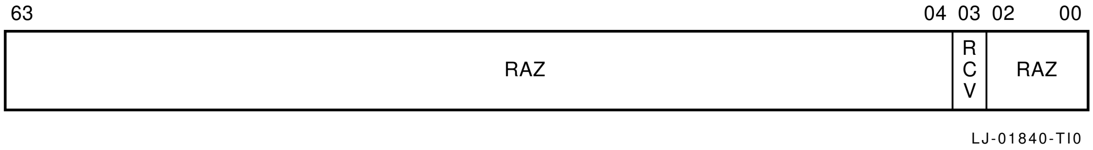
5.2.7 Serial Line Receive Register (SL_RCV)

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

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

Figure 5–7 shows the Serial Line Receive Register format.

Figure 5–7 Serial Line Receive Register



5.2.8 Instruction Translation Buffer ZAP Register (ITBZAP)

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

5.2.9 Instruction Translation Buffer ASM Register (ITBASM)

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

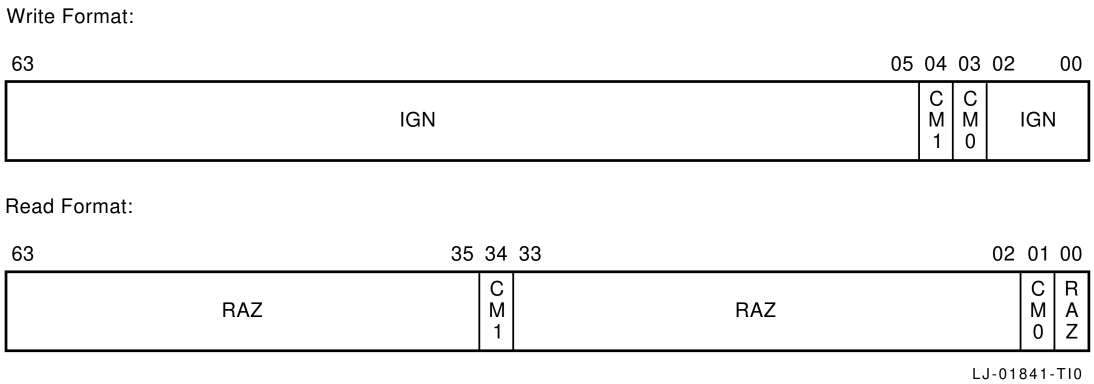
5.2.10 Instruction Translation Buffer IS Register (ITBIS)

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

5.2.11 Processor Status Register (PS)

The PS register is a read/write register containing only the current mode bits of the architecturally defined PS. Figure 5–8 shows the PS register format. See the *Alpha Architecture Reference Manual* for additional information.

Figure 5–8 Processor Status Register



5.2.12 Exception Summary Register (EXC_SUM)

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

The register containing the result of the operation is recorded in the exception register write mask parameter, as a single bit in a 64-bit shift register specifying registers F31-F0 and I31-I0. The EXC_SUM register provides a one-bit window to the exception register write mask parameter. This is visible only through the EXC_SUM register.

Each read to the EXC_SUM shifts one bit in order F31-F0 then I31-I0. The read also clears the corresponding bit. The EXC_SUM must be read 64 times to extract the complete mask and clear the entire register. If no integer traps are present (IOV=0), only the first 32 corresponding floating-point register bits need to be read and cleared.

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

The Write Mask register bit clears three cycles after a read. Code intended to read the register must allow at least three cycles between reads. This allows the clear and shift operations to complete in order to ensure reading successive bits. Figure 5–9 shows the exception summary register format. Table 5–6 lists the register fields and descriptions.

Figure 5–9 Exception Summary Register

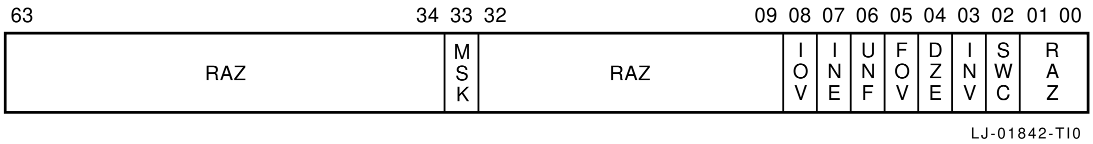


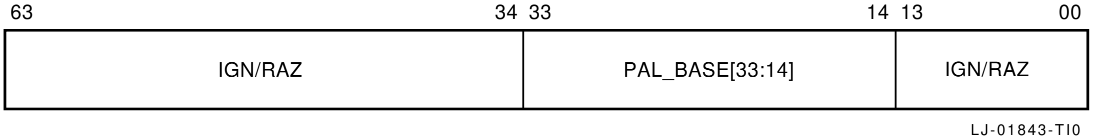
Table 5–6 Exception Summary Register Fields

Field	Type	Description
SWC	WA	Indicates software completion possible. The bit is set after a floating-point instruction containing the /S modifier completes with an arithmetic trap and all previous floating-point instructions that trapped since the last HW_ MTPR EXC_SUM also contained the /S modifier. The SWC bit is cleared whenever a floating-point instruction without the /S modifier completes with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps until the register is written by way of an HW_MTPR instruction. The bit is always cleared upon any HW_MTPR write to the EXC_SUM register.
INV	WA	Indicates invalid operation.
DZE	WA	Indicates divide by zero.
FOV	WA	Indicates floating-point overflow.
UNF	WA	Indicates floating-point underflow.
INE	WA	Indicates floating inexact error.
IOV	WA	Indicates Fbox convert to integer overflow or integer arithmetic overflow.
MSK	RC	Exception Register Write Mask IPR window.

5.2.13 PAL_BASE Address Register (PAL_BASE)

The PAL_BASE register is a read/write register containing the base address for PALcode. This register is cleared by the hardware at reset. Figure 5–10 shows the PAL_BASE address register format.

Figure 5–10 PAL_BASE Address Register



5.2.14 Hardware Interrupt Request Register (HIRR)

The HIRR is a read-only register providing a record of all currently outstanding interrupt requests and summary bits at the time of the read. For each bit of the HIRR [5:0], there is a corresponding bit of the Hardware Interrupt Enable register (HIER) that must be set to request an interrupt.

In addition to returning the status of the hardware interrupt requests, a read of the HIRR returns the state of the software interrupt and AST requests.

Note

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

The register guarantees that the HWR bit reflects the status as shown by the HIRR bits. All interrupt requests are blocked while executing in PALmode. Figure 5–11 shows the hardware interrupt request register format. Table 5–7 lists the register fields and gives a description of each. For additional information on interrupt operations, refer to Section 2.3.3.

Figure 5–11 Hardware Interrupt Request Register

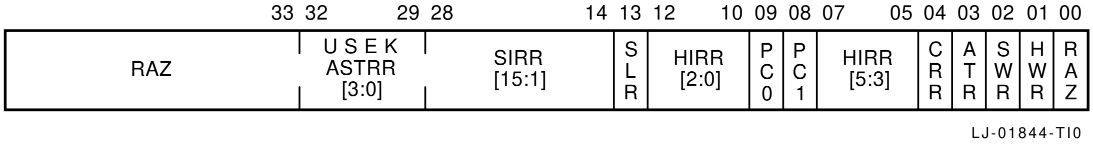


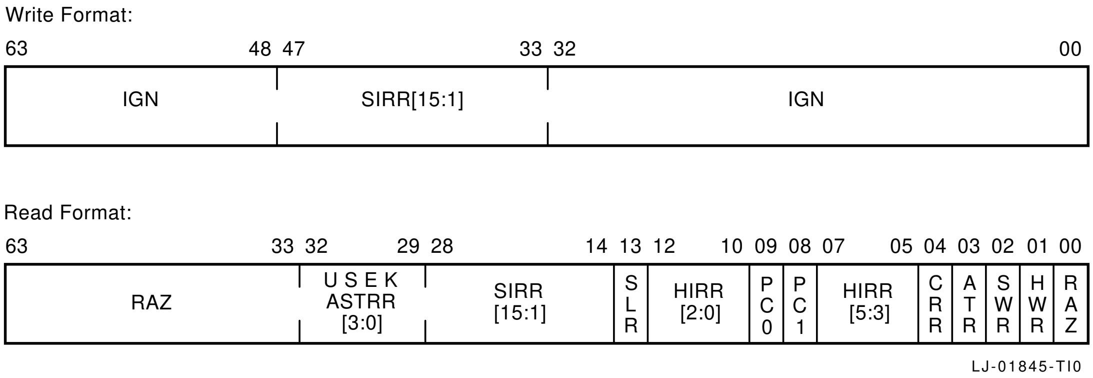
Table 5–7 Hardware Interrupt Request Register Fields

Field	Type	Description
HWR	RO	Is set if any hardware interrupt request and corresponding enable is set
SWR	RO	Is set if any software interrupt request and corresponding enable is set
ATR	RO	Is set if any AST request and corresponding enable is set. This bit also requires that the processor mode be equal to or higher than the request mode. SIER 2 must be set to allow AST interrupt requests.
CRR	RO	CRD correctable read error interrupt request. This interrupt is cleared by way of the SL_CLR register.
HIRR [5:0]	RO	Contains delayed copies of **Irq_h [5:0]** pins
PC1	RO	Performance counter 1 interrupt request
PC0	RO	Performance counter 0 interrupt request
SLR	RO	Serial line interrupt request. Also see SL_RCV, SL_XMIT, and SL_CLR
SIRR [15:1]	RO	Corresponds to software interrupt request 15 through 1
ASTRR [3:0]	RO	Corresponds to AST request 3 through 0 (USEK)

5.2.15 Software Interrupt Request Register (SIRR)

The SIRR is a read/write register used to control software interrupt requests. For each bit of the SIRR, there is a corresponding bit of the Software Interrupt Enable register (SIER) that must be set to request an interrupt. Reads of the SIRR return the complete set of interrupt request registers and summary bits (see Table 5–7 for details). All interrupt requests are blocked while executing in PALmode. Figure 5–12 shows the SIRR format.

Figure 5–12 Software Interrupt Request Register

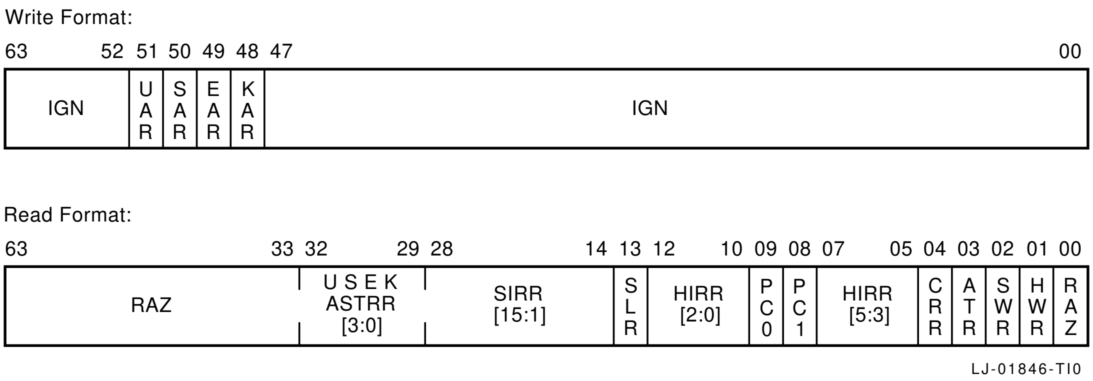


5.2.16 Asynchronous Trap Request Register (ASTRR)

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

All interrupt requests are blocked while executing in PALmode. Reads of the ASTRR return the complete set of interrupt request registers and summary bits. See Table 5–7 for details. Figure 5–13 shows the ASTRR format.

Figure 5–13 Asynchronous Trap Request Register



5.2.17 Hardware Interrupt Enable Register (HIER)

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

- Performance counters
- Serial line
- Correctable read interrupts

There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the HIER return the complete set of interrupt enable registers. See Table 5–7 for details. Figure 5–14 shows the hardware interrupt enable register format. Table 5–8 lists the register fields and a description of each.

Figure 5–14 Hardware Interrupt Enable Register

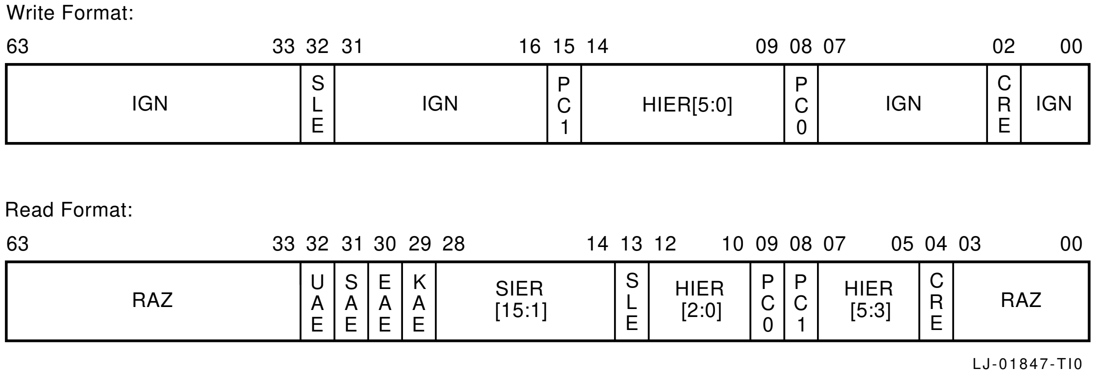


Table 5–8 Hardware Interrupt Enable Register Fields

Field	Type	Description
HIER [5:0]	RW	Interrupt enables for pins **Irq_h [5:0]**
SIER [15:1]	RW	Corresponds to software interrupt requests 15 through 1
ASTER [3:0]	RW	Corresponds to ASTRR enable 3 through 0 (USEK)
PC1	RW	Performance counter 1 interrupt enable

(continued on next page)

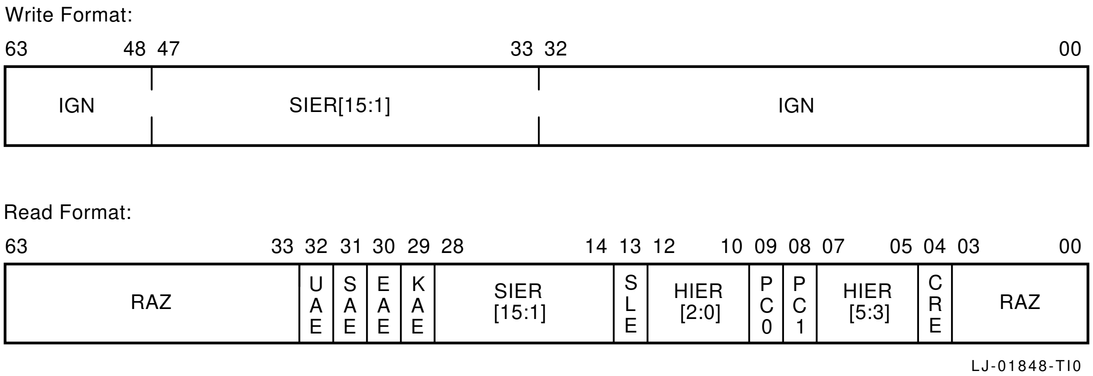
Table 5–8 (Cont.) Hardware Interrupt Enable Register Fields

Field	Type	Description
PC0	RW	Performance counter 0 interrupt enable
SLE	RW	Serial line interrupt enable Also see SL_RCV, SL_XMIT, and SL_CLR
CRE	RW	CRD correctable read error interrupt enable This interrupt request is cleared by way of the SL_CLR register

5.2.18 Software Interrupt Enable Register (SIER)

The SIER is a read/write register. It is used to enable corresponding bits of the SIRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the SIER return the complete set of interrupt enable registers. See Table 5–7 for details. Figure 5–15 shows the software interrupt enable register format.

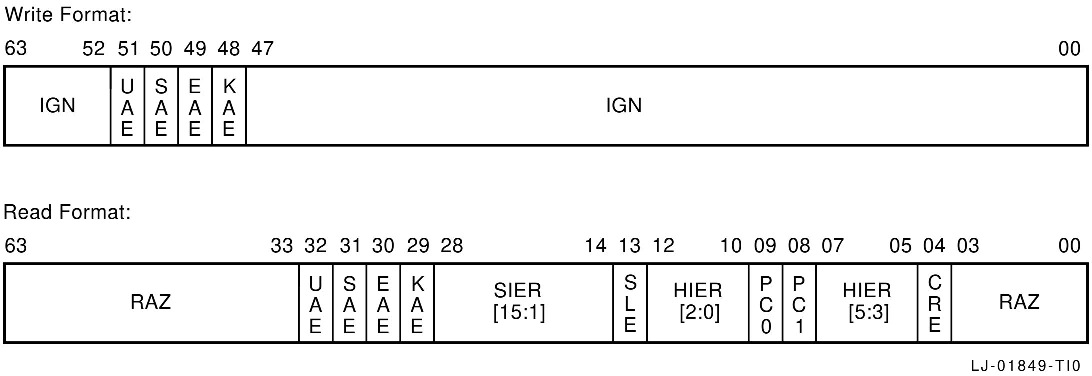
Figure 5–15 Software Interrupt Enable Register



5.2.19 AST Interrupt Enable Register (ASTER)

The ASTER is a read/write register. It is used to enable corresponding bits of the ASTRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits. As with the reads of the interrupt request registers, reads of the ASTER return the complete set of interrupt enable registers. See Table 5–7 for details. Figure 5–16 shows the ASTER format.

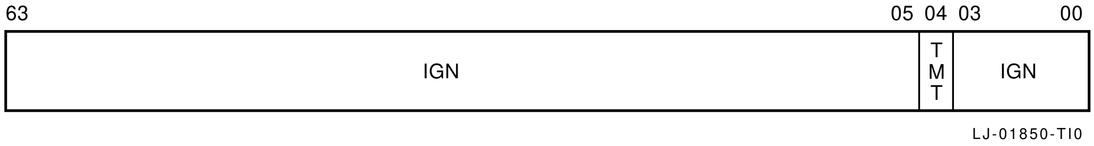
Figure 5–16 AST Interrupt Enable Register



5.2.20 Serial Line Transmit Register (SL_XMIT)

The SL_XMIT register contains a single write-only bit. This bit is used with the interrupt control registers, the **sRomD_h** pin, and the **sRomClk_h** pin to provide an on-chip serial line function. The TMT bit is functionally connected to the **sRomClk_h** pin after the Icache is loaded from the external serial ROM. Writing the TMT bit can be used to transmit data off chip, one bit at a time under a software timing loop. Figure 5–17 shows the SL_XMIT register format.

Figure 5–17 Serial Line Transmit Register



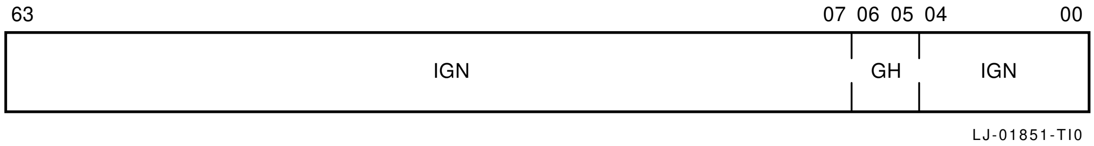
5.3 Abox Internal Processor Registers

The following sections describe the Abox internal processor registers.

5.3.1 Translation Buffer Control Register (TB_CTL)

The granularity hint (GH) field selects between the 21064/21064A TB page mapping sizes. There are two sizes in the ITB and four sizes in the DTB. When only two sizes are provided, the large-page-select (GH=11(bin)) field selects the largest mapping size (512 * 8 KB). All other values select the smallest (8 KB) size. The GH field affects both reads and writes to the ITB and DTB. Figure 5–18 shows the translation buffer control register format. See the *Alpha Architecture Reference Manual* for additional information.

Figure 5–18 Translation Buffer Control Register

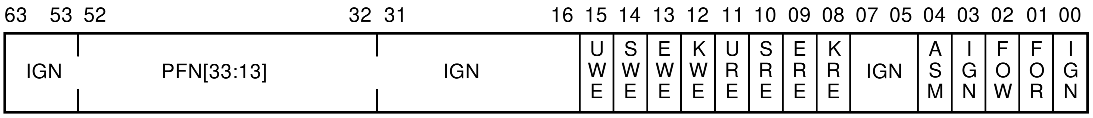


5.3.2 Data Translation Buffer Page Table Entry Register (DTB_PTE)

The DTB_PTE register is a read/write register representing the 32-entry DTB. The entry to be written is chosen by a not-last-used (NLU) algorithm implemented in the hardware. A DTB round robin (DTB_RR) algorithm can be selected by setting ABOX_CTL [9]. Writes to the DTB_PTE use the memory format bit positions as described in the *Alpha Architecture Reference Manual* with the exception that some fields are ignored. The valid bit is not represented in hardware.

The DTB’s tag array is updated simultaneously from the TB_Tag register when the DTB_PTE register is written. Reads of the DTB_PTE require two instructions. The first instruction sends the PTE data to the Data Translation Buffer Page Table Entry Temporary register (DTB_PTE_TEMP). The second instruction, reading from the DTB_PTE_TEMP register, returns the PTE entry to the register file. Reading or writing the DTB_PTE register increments the TB entry pointer of the DTB, which allows reading the entire set of DTB_PTE entries. Figure 5–19 shows the DTB_PTE register format.

Figure 5–19 Data Translation Buffer Page Table Entry Register



LJ-01852-TI0

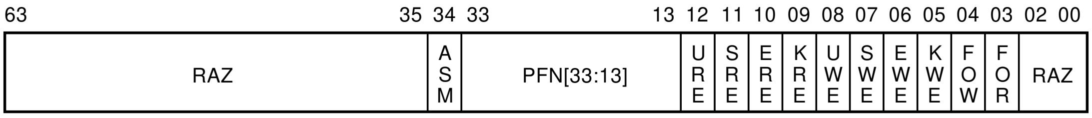
5.3.3 Data Translation Buffer Page Table Entry Temporary Register (DTB_PTE_TEMP)

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

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

Figure 5–20 shows DTB_PTE_TEMP register format.

Figure 5–20 Data Translation Buffer Page Table Entry Temporary Register



LJ-01853-TI0

5.3.4 Memory Management Control and Status Register (MM_CSR)

When D-stream faults occur the information about the fault is latched and saved in the MM_CSR register. The virtual address register (VA) and MM_CSR registers are locked against further updates until the software reads the Virtual Address register. PALcode must explicitly unlock this register whenever its entry point is higher in priority than a DTB miss. The MM_CSR bits are only modified by the hardware when the register is not locked and a memory management error or a DTB miss occurs. The MM_CSR is unlocked after reset. Figure 5–21 shows the MM_CSR register format. Table 5–9 lists the register fields and a brief description.

Figure 5–21 Memory Management Control and Status Register

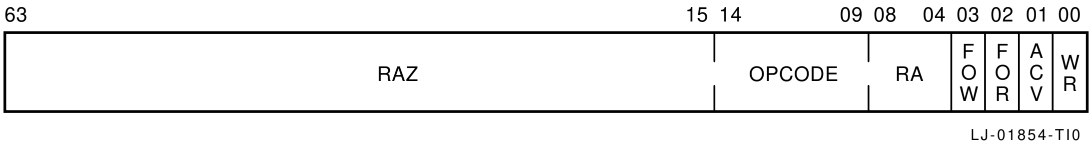


Table 5–9 Memory Management Control and Status Register

Field	Type	Description
WR	RO	Set if reference that caused error was a write.
ACV	RO	Set if reference caused an access violation.
FOR	RO	Set if reference was a read and the PTE’s FOR bit was set.
FOW	RO	Set if reference was a write and the PTE’s FOW bit was set.
RA	RO	RA field of the faulting instruction.
OPCODE	RO	Opcode field of the faulting instruction.

5.3.5 Virtual Address Register (VA)

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

5.3.6 Data Translation Buffer ZAP Register (DTBZAP)

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

5.3.7 Data Translation Buffer ASM Register (DTBASM)

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

5.3.8 Data Translation Buffer Invalidate Single Register (DTBIS)

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

5.3.9 Flush Instruction Cache Register (FLUSH_IC)

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

5.3.10 Flush Instruction Cache ASM Register (FLUSH_IC_ASM)

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

5.3.11 Abox Control Register (ABOX_CTL)

Figure 5–22 shows the Abox control register format. Table 5–10 lists the register fields and descriptions.

Figure 5–22 Abox Control Register [1]

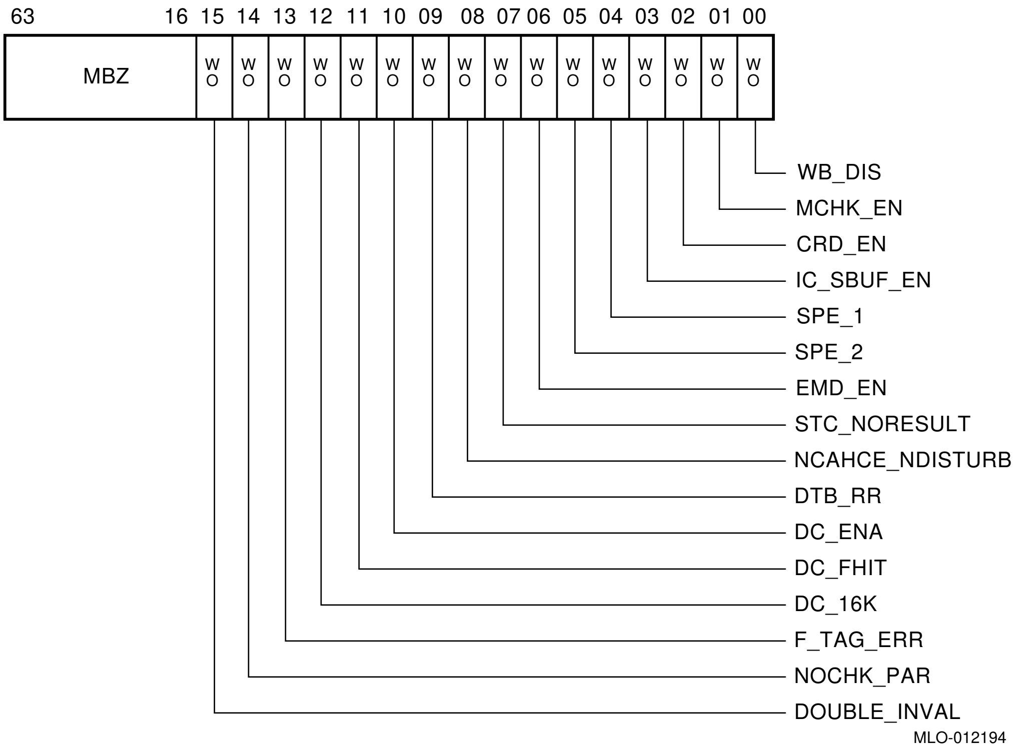


Table 5–10 Abox Control Register Fields

Field	Type	Description
WB_DIS	WO,0	Write Buffer unload Disable. When set, this bit prevents the write buffer from sending write data to the BIU. It should be set for diagnostics only.
MCHK_EN	WO,0	Machine Check Enable. When this bit is set, the Abox generates a machine check when errors (which are not correctable by the hardware) are encountered. When this bit is cleared, uncorrectable errors do not cause a machine check. However, the BIU_STAT, DC_STAT, BIU_ADDR, and FILL_ADDR registers are updated and locked when the errors occur.

(continued on next page)

[1] Versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2 did not implement ABOX_CTL [9:7]. PALcode for these processors is upward compatible if the PALcode did not set ABOX_CTL [15:12] or ABOX_CTL [9:7].

Table 5–10 (Cont.) Abox Control Register Fields

Field	Type	Description
CRD_EN	WO,0	Corrected read data interrupt enable. When this bit is set, the Abox generates an interrupt request whenever a pin bus transaction is terminated with a **cAck_h** code of SOFT_ERROR.
IC_SBUF_EN	WO,0	Icache stream buffer enable. When set, this bit enables operation of a single entry Icache stream buffer.
SPE_1	WO,0	When this bit is set, it enables one-to-one super page mapping of the D-stream virtual addresses with VA [42:30] = 1FFE (Hex) to the physical addresses with PA [33:30] = 0 (Hex). Access is only allowed in kernel mode.
SPE_2	WO,0	When this bit is set, it enables one-to-one super page mapping of the D-stream virtual addresses with VA [33:13] directly to physical addresses PA [33:13], if virtual address bits VA [42:41] = 2. Virtual address bits VA [40:34] are ignored in this translation. Access is only allowed in kernel mode.
EMD_EN	WO,0	Limited hardware support is provided for big endian data formats by way of bit [6] of the ABOX_CTL register. When set, this bit inverts the physical address bit [2] for all D-stream references. It is intended that the chip endian mode be selected during initialization of PALcode only.
STC_NORESULT[2]	WO,0	When clear the 21064/21064A implements lock operation in conformance to Alpha AXP Architecture. When set the the 21064/21064A does not conform to Alpha AXP architecture. These listed items apply. • The result written into the register identified by Ra in STL_C/STQ_C and HW_ST/C instructions is UNPREDICTABLE. This allows the Ibox to restart the memory reference pipeline when the STL_C/STQ_C is transferred from the write buffer to the BIU, and so increases the repetition rate with which STL_C/STQ_C instructions can be processed. • LDL_L/LDQ_L, STL_C/STQ_C and HW_ST/C instructions will invalidate the Dcache line associated with their generated address. These invalidates will not be visible to load or store instructions that issue in the two CPU cycles after the LDL_L/LDQ_L, STL_C/STQ_C or HW_ST/C issues. This bit is cleared by chip reset.

[2]ABOX_CTL [09:07] (DTB_RR, NCACHE_NDISTURB, STC_NORESULT) were not implemented versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2. PALcode for these processors is upward compatible if the PALcode did not set ABOX_CTL [15:12] or ABOX_CTL [09:07].

(continued on next page)

Table 5–10 (Cont.) Abox Control Register Fields

Field	Type	Description
NCACHE_ NDISTURB[2]	WO,0	When this bit is set, it enables a mode which make noncacheable only those external reads for which the 21064/21064A does not probe the external cache. This bit is cleared by chip reset. See Section 6.4.10.3.
DTB_RR[2]	WO,0	When this bit is set, it selects the round robin replacement algorithm in the DTB.
DC_ENA	WO,0	Dcache enable. When clear, this bit disables and flushes the Dcache. When set, this bit enables the Dcache.
DC_FHIT	WO,0	Dcache force hit. When set, this bit forces all D-stream references to hit in the Dcache. This bit takes precedence over DC_ENA. That is, when DC_FHIT is set and DC_ENA is clear all D-stream references hit in the Dcache.
DC_16K[3]	WO,0	**21064A only.** Set to select 16K byte Dcache. Clear to select 8K byte Dcache.
F_TAG_ERR[3]	WO,0	**21064A only.** Set to generate bad Dcache tag parity on fills.
NOCHK_PAR[3]	WO,0	**21064A only.** Set to disable checking of Icache and Dcache parity.
DOUBLE_ INVAL[3]	WO,0	**21064A only.** When set, asserting **dInvReq_h 0** invalidates both Dcache blocks addressed by **iAdr_h [12:5]**.

[2]ABOX_CTL [09:07] (DTB_RR, NCACHE_NDISTURB, STC_NORESULT) were not implemented versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2. PALcode for these processors is upward compatible if the PALcode did not set ABOX_CTL [15:12] or ABOX_CTL [09:07].

[3]ABOX_CTL [15:12] DOUBLE_INVAL, NOCHK_PAR, F_TAG_ERR and DC_16K exist on **21064A** only.

5.3.12 Alternate Processor Mode Register (ALT_MODE)

The ALT_MODE is a write-only register. The AM field specifies the alternate processor mode used by HW_LD and HW_ST instructions that have their ALT bit (bit [14]) set. Figure 5–23 shows the alternate processor mode register format and Table 5–11 lists the register modes.

Figure 5–23 Alternate Processor Mode Register

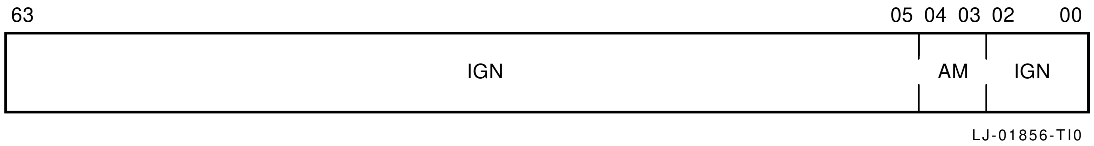


Table 5–11 Alternate Processor Mode Register

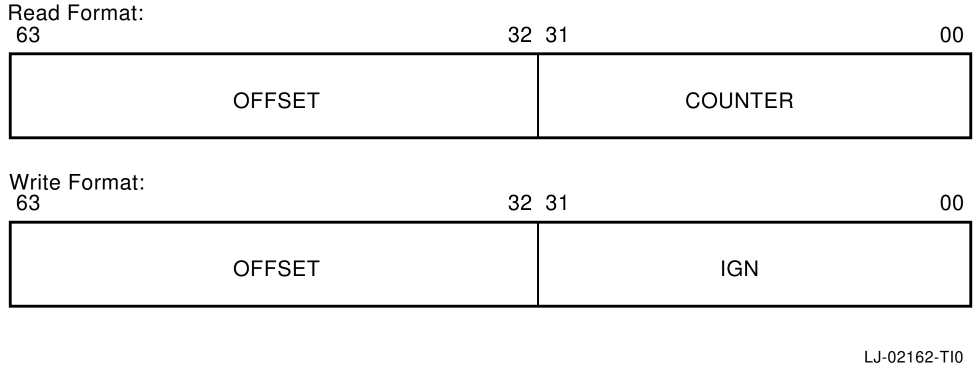
ALT_MODE [4:3]	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

5.3.13 Cycle Counter Register (CC)

The 21064/21064A supports a cycle counter, as described in the *Alpha Architecture Reference Manual*. When enabled, the CC increments once each CPU cycle. The HW_MTPR Rn, CC writes the CC [63:32] with the value held in the Rn [63:32]. The CC [31:0] are not changed. This register is read by the RPCC instruction as defined in the *Alpha Architecture Reference Manual*.

Figure 5–24 shows the register format (top register) when read by the HW_MFPR Rn, CC instruction and when written (bottom register) by the HW_MTPR Rn, CC instruction.

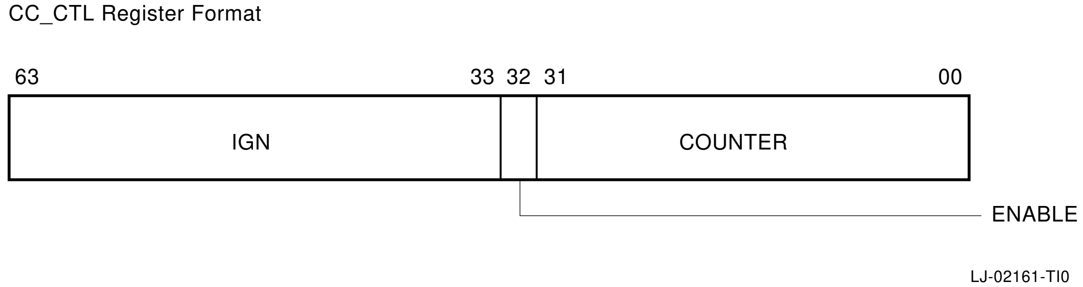
Figure 5–24 Cycle Counter Register



5.3.14 Cycle Counter Control Register (CC_CTL)

The HW_MTPR Rn, CC_CTL writes the CC [31:0] with the value held in Rn [31:0]. The CC register bits [63:32] are not changed. The CC register bits [3:0] must be written with zero. If Rn bit [32] is set, then the counter is enabled, otherwise the counter is disabled. CC_CTL is a write-only register. Figure 5–25 shows the register format when written by the HW_MTPR Rn, CC_CTL instruction.

Figure 5–25 Cycle Counter Control Register



5.3.15 Bus Interface Unit Control Register (BIU_CTL)

Figure 5–26 shows the bus interface unit control register format. Table 5–12 lists the register fields and gives a description of each.

Figure 5–26 21064/21064A Bus Interface Unit Control Register[1]

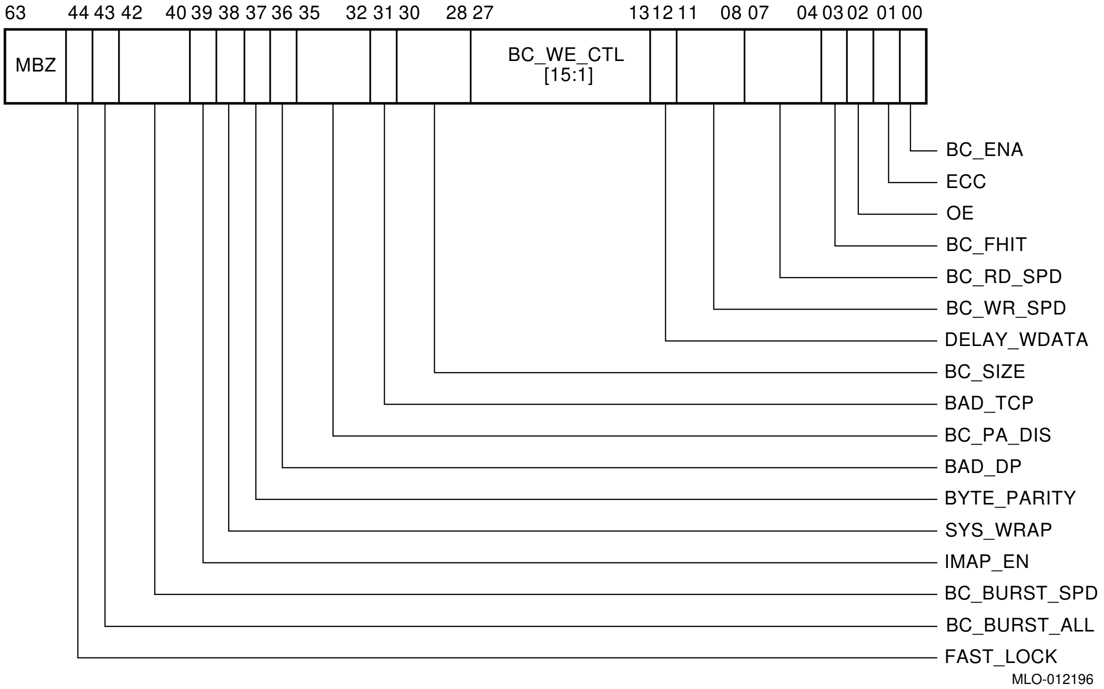


Table 5–12 Bus Interface Unit Control Register Fields

Field	Type	Description
BC_ENA	WO,0	External cache enable. When this bit is cleared, the bit disables the external cache. When the Bcache is disabled, the BIU does not probe the external cache tag store for read/write references; it launches a request on **cReq_h** immediately.

(continued on next page)

[1] Versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2 did not implement BIU_CTL [43, 42:40, 38, 12]. PALcode for these processors is upwards compatible if the PALcode did not set these bits.

Table 5–12 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
ECC	WO,0	When this bit is clear, the 21064/21064A generates/expects parity on four of the **check_h** pins. When this bit is set, the 21064/21064A generates/expects ECC on the **check_h** pins.
OE	WO,0	When this bit is set, the 21064/21064A does not assert its chip enable pins during RAM write cycles, thus enabling these pins to be connected to the output enable pins of the cache RAMs. **Caution** The output enable bit in the BIU_CTL register (BIU_CTL [2]) must be set if the system uses SRAMs in the output enable mode (that is, if the **tagCEOE** and/or **dataCEOE** signals are connected to the output enable input of the SRAM and the 21064 /21064A enable is always enabled). If this bit is inadvertently cleared, the tag and data SRAMs will be enabled during writes, and damage can result.
BC_FHIT	WO,0	External cache force hit. When this bit is set and the BC_ENA bit is also set, all pin bus READ_BLOCK and WRITE_BLOCK transactions are forced to hit in external cache. Tag and tag control parity are ignored. The BC_ENA takes precedence over BC_FHIT. When BC_ENA is cleared and BC_FHIT is set, no tag probes occur and external requests are directed to the **cReq_h** pins. **Note** The BC_PA_DIS field takes precedence over the BC_FHIT bit.

(continued on next page)

Table 5–12 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
BC_RD_SPD	WO,0	External cache read speed. This field indicates to the BIU the read access time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less than the read access time of the external cache RAMs. **21064** access times for reads must be in the range [16:4] CPU cycles, which means the values for the BC_RD_SPD field are in the range of [15:3]. **21064A** access times for reads must be in the range [16:3] CPU cycles, which means the values for the BC_RD_SPD field are in the range of [15:2].
BC_WR_SPD	WO,0	External cache write speed. This field indicates to the BIU the write cycle time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less than the write cycle time of the external cache RAMs. The access times for writes must be in the range [16:2] CPU cycles, which means the values for the BC_WR_SPD field are in the range of [15:1].
DELAY_WDATA[2]	WO,0	When this bit is set, it changes the timing of the data bus during external cache writes. See Section 6.4.4.
BC_WE_CTL	WO,0	External cache write enable control. This field is used to control the timing of the write enable and chip enable pins during writes into the data and tag control RAMs. It consists of 15 bits, where each bit determines the value placed on the write enable and chip enable pins during a given CPU cycle of the RAM write access. When a given bit of the BC_WE_CTL is set, the write enable and chip enable pins are asserted during the corresponding CPU cycle of the RAM access. The BC_WE_CTL bit [0] (bit [13] in BIU_CTL) corresponds to the second cycle of the write access, BC_WE_CTL [1] (bit [14] in BIU_CTL) to the third CPU cycle, and so on. The write enable pins will never be asserted in the first CPU cycle of a RAM write access. Unused bits in the BC_WE_CTL field must be written with zeros.

[2]BC_BURST_ALL, BC_BURST_SPD, SYS_WRAP, and DELAY_WDATA were not implemented in versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2. PALcode which did not set these bits may be used without change.

(continued on next page)

Table 5–12 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
BC_SIZE	WO,0	This field is used to indicate the size of the external cache. See Table 5–13 for the encodings.
BAD_TCP	WO,0	When set, this bit causes the 21064/21064A to write bad parity into the tag control RAM whenever it does a fast external RAM write. (Diagnostic use only.)
BC_PA_DIS	WO,0	This 4-bit field may be used to prevent the CPU chip from using the external cache to service reads and writes based upon the quadrant of physical address space that they reference. The correspondence between this bit field and the physical address space is shown in Table 5–14. When a read or write reference is presented to the BIU the values of BC_PA_DIS, BC_ENA, and the physical address bits [33:32] determine whether to attempt to use the external cache to satisfy the reference. If the external cache is not to be used for a given reference the BIU does not probe the tag store and makes the appropriate system request immediately. The value of BC_PA_DIS has NO impact on which portions of the physical address space can be cached in the primary caches. System components control this by way of the **dRAck_h** field of the pin bus.
BAD_DP	WO,0	When this bit is set, the BAD_DP causes the 21064/21064A to invert the value placed on bits [0], [7], [14] and [21] of the **check_h [27:0]** field during off-chip writes. This produces bad parity when the 21064/21064A is in parity mode, and bad check bit codes when in ECC mode. (Diagnostic use only.)
SYS_WRAP[2]	WO,0	When this bit is set, it indicates that the system returns read response data wrapped around the requested chunk. This bit is cleared by chip reset. See Section 6.5.5.5.
BC_BURST_SPD[2]	WO,0	When these bits are cleared, the timing of all Bcache reads is controlled by the value of BC_RD_SPD. When these bits are set in 128-bit mode, the second read takes BC_BURST_SPD+1 cycles. When these bits are set in 64-bit mode, the second and fourth reads take BC_BURST_SPD+1 cycles. If BC_BURST_ALL is set, the third read takes BC_BURST_ SPD+1 cycles also. See Section 6.5.4.6.

[2]BC_BURST_ALL, BC_BURST_SPD, SYS_WRAP, and DELAY_WDATA were not implemented in versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2. PALcode which did not set these bits may be used without change.

(continued on next page)

Table 5–12 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
BC_BURST_ALL[2]	WO,0	In 64-bit mode this bit is set if BC_BURST_SPD should be used to time the third (of four) RAM read cycle.
BYTE_PARITY[3]	WO,0	**21064A only.** If set when BIU_CTL ECC is cleared, external byte parity is selected. If set when BIU_CTL ECC is set, this bit is ignored.
IMAP_EN[3]	WO,0	**21064A only.** Set to allow **dMapWE_h [1:0]** to assert for I-stream backup cache reads.
FAST_LOCK[3]	WO,0	**21064A only.** When set, FAST_LOCK mode operation is selected. FAST_LOCK mode can only be used when BIU_CTL [2] OE is also set indicating that OE mode Bcache RAMs are used.

[2]BC_BURST_ALL, BC_BURST_SPD, SYS_WRAP, and DELAY_WDATA were not implemented in versions of the **21064** where the CHIP_ID field of DC_STAT was 000_2. PALcode which did not set these bits may be used without change.

[3]BIU_CTL [44,39,37] FAST_LOCK, IMAP_EN and BYTE_PARITY exist on **21064A** only.

Table 5–13 lists the encoding for BC_SIZE. Table 5–14 lists the BIU_CTL physical addresses.

Table 5–13 BC_SIZE

BC_SIZE	Cache Size	BC_SIZE	Cache Size
0 0 0	128 KB	1 0 0	2 MB
0 0 1	256 KB	1 0 1	4 MB
0 1 0	512 KB	1 1 0	8 MB
0 1 1	1 MB	1 1 1	16 MB

Table 5–14 BC_PA_DIS

BIU_CTL Bits	Physical Address	BIU_CTL Bits	Physical Address
32	PA [33:32] = 0	34	PA [33:32] = 2
33	PA [33:32] = 1	35	PA [33:32] = 3

5.3.16 Data Cache Status Register (DC_STAT—21064 Only)

The DC_STAT is a read-only register and is only used by the diagnostics. It has the same address as the C_STAT register used by the **21064A**.

Figure 5–27 shows the **21064** Dcache status register format. Table 5–15 lists the register fields and gives a description of each.

Figure 5–27 Data Cache Status Register

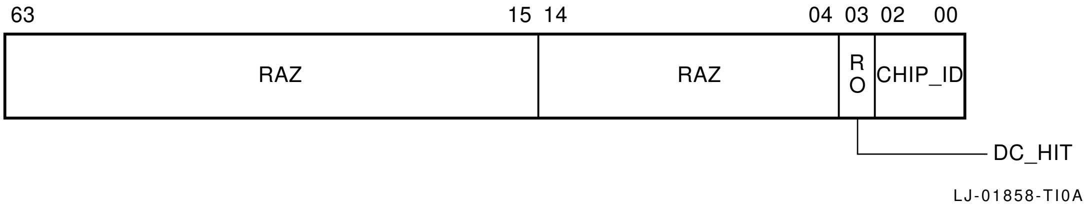


Table 5–15 Dcache Status Register Fields

Field	Type	Description
CHIP_ID	RO	These bits identify the devices as listed here: • 000_2—Early versions of **21064** • 111_2—Production version of **21064**
DC_HIT	RO	This bit indicates whether the last load or store instruction processed by the Abox hit (DC_HIT set) or missed (DC_HIT clear) the Dcache. Loads that miss the Dcache can be completed without requiring external reads. (Diagnostic use only.)

5.3.17 Cache Status Register (C_STAT, 21064A Only)

The C_STAT is a read-only register and is only used by the diagnostics. It has the same address as the DC_STAT register used by the **21064**.

Figure 5–28 shows the **21064A** Dcache status register format. Table 5–16 lists the register fields and gives a description of each.

Figure 5–28 Cache Status Register

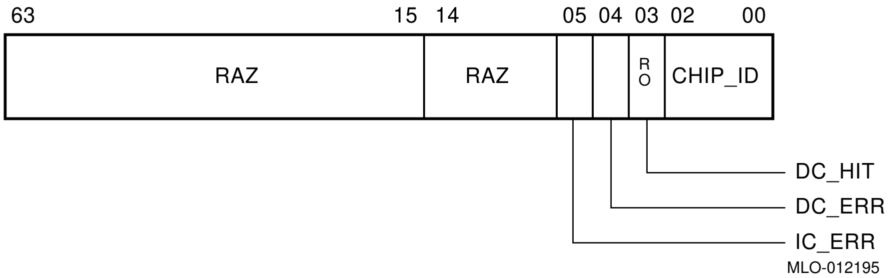


Table 5–16 Cache Status Register Fields

Field	Type	Description
CHIP_ID	RO	These bits identify the devices as listed here: • 001_2—Early version of **21064A** • 011_2—Production version of **21064A**
DC_HIT	RO	This bit indicates whether the last load or store instruction processed by the Abox hit (DC_HIT set) or missed (DC_HIT clear) the Dcache. Loads that miss the Dcache can be completed without requiring external reads. (Diagnostic use only.)
DC_ERR	RC	Set by Dcache parity error.
IC_ERR	RC	Set by Icache parity error.

5.3.18 Bus Interface Unit Status Register (BIU_STAT)

The BIU_STAT is a read-only register.

Bits [6:0] of the BIU_STAT register are locked against further updates when one of the following bits is set:

- BIU_HERR
- BIU_SERR
- BC_TPERR
- BC_TCPERR

The address associated with the error is latched and locked in the BIU_ADDR register. Bits [6:0] of the BIU_STAT register and BIU_ADDR are also spuriously locked when a parity error or an uncorrectable ECC error occurs during a primary cache fill operation. The BIU_STAT bits [7:0] and BIU_ADDR are unlocked when the BIU_ADDR register is read.

When FILL_ECC or FILL_DPERR is set, BIU_STAT bits [13:8] are locked against further updates. The address associated with the error is latched and locked in the FILL_ADDR register. The BIU_STAT bits [14:8] and FILL_ADDR are unlocked when the FILL_ADDR register is read.

This register is not unlocked or cleared by reset and needs to be explicitly cleared by PALcode.

Figure 5–29 shows the bus interface unit status register format. Table 5–17 lists the register fields and gives a description of each.

Figure 5–29 Bus Interface Unit Status Register

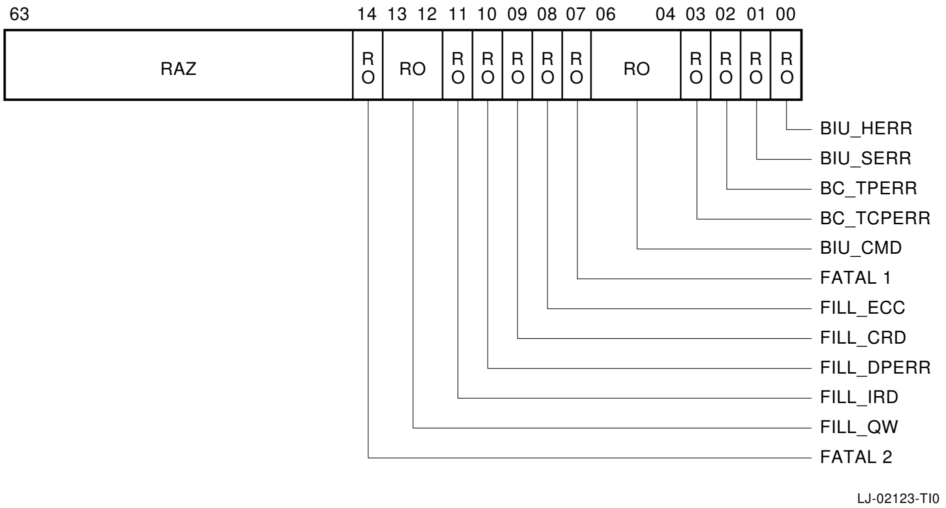


Table 5–17 Bus Interface Unit Status Register Fields

Field	Type	Description
BIU_HERR	RO	When this bit is set, it indicates that an external cycle was terminated with the **cAck_h** pins indicating HARD_ ERROR.
BIU_SERR	RO	When this bit is set, it indicates that an external cycle was terminated with the **cAck_h** pins indicating SOFT_ ERROR.
BC_TPERR	RO	When this bit is set, it indicates that an external cache tag probe encountered bad parity in the tag address RAM.
BC_TCPERR	RO	When this bit is set, it indicates that an external cache tag probe encountered bad parity in the tag control RAM.
BIU_CMD	RO	This field latches the cycle type on the cReq_h pins when a BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR error occurs.
FATAL1	RO	When this bit is set, it indicates that an external cycle was terminated with the **cAck_h** pins indicating HARD_ ERROR or that an external cache tag probe encountered bad parity in the tag address RAM or the tag control RAM while one of BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR was already set.
FILL_ECC	RO	ECC error. When this bit is set, it indicates that primary cache fill data received from outside the CPU chip contained an ECC error.
FILL_CRD	RO	Correctable read. This bit only has meaning when FILL_ ECC is set. When this bit is set, it indicates that the information latched in BIU_STAT [13:8], FILL_ADDR, and FILL_SYNDROME relates to an error quadword which does not contain multi-bit errors in either of its component longwords.
FILL_DPERR	RO	Fill Parity Error. When this bit is set, it indicates that the BIU received data with a parity error from outside the CPU chip while performing either a Dcache or Icache fill. FILL_DPERR is only meaningful when the CPU chip is in parity mode, as opposed to ECC mode.

(continued on next page)

Table 5–17 (Cont.) Bus Interface Unit Status Register Fields

Field	Type	Description
FILL_IRD	RO	This bit is only meaningful when either FILL_ECC or FILL_DPERR is set. The FILL_IRD bit is set to indicate that the error that caused FILL_ECC or FILL_DPERR to set occurred during an Icache fill and clear to indicate that the error occurred during a Dcache fill.
FILL_QW	RO	This field is only meaningful when either FILL_ECC or FILL_DPERR is set. The FILL_QW bit identifies the quadword within the hexaword primary cache fill block which caused the error. It can be used together with FILL_ADDR [33:5] to get the complete physical address of the bad quadword.
FATAL2	RO	When this bit is set, it indicates that a primary cache fill operation resulted in either a multi-bit ECC error or in a parity error while FILL_ECC or FILL_DPERR was already set.

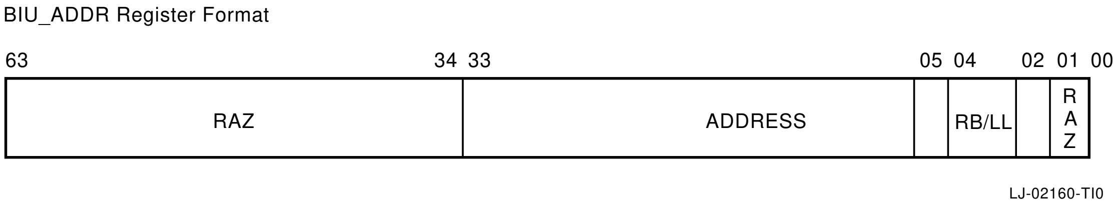
5.3.19 Bus Interface Unit Address Register (BIU_ADDR)

The BIU_ADDR is a read-only register that contains the physical address associated with errors reported by BIU_STAT [7:0]. Its contents are meaningful only when one of BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR are set. Reads of the BIU_ADDR register unlock both BIU_ADDR and BIU_STAT [7:0].

The BIU_ADDR bits [33:5] contain the values of **adr_h** bits [33:5] associated with the pin bus transaction that resulted in the error indicated in BIU_STAT [7:0].

If the BIU_CMD field of the BIU_STAT register indicates that the transaction that received the error was READ_BLOCK or load_locked, then BIU_ADDR [4:2] are UNPREDICTABLE. If the BIU_CMD field of the BIU_STAT register encodes any pin bus command other than READ_BLOCK or load_locked, then BIU_ADDR bits [4:2] will contain zeros. The BIU_ADDR bits [63:34] and BIU_ADDR bits [1:0] always read as zero. Figure 5–30 shows the bus interface unit address register (BIU_ADDR) format.

Figure 5–30 Bus Interface Unit Address Register



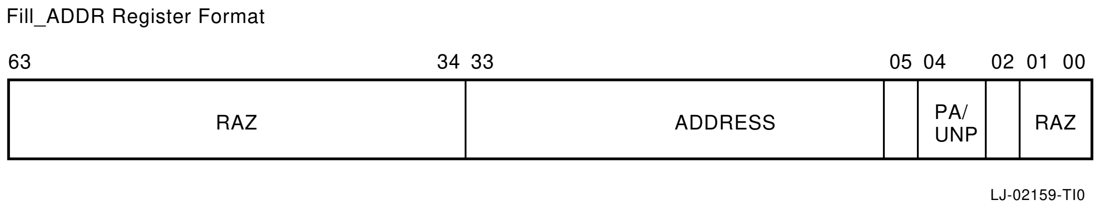
5.3.20 Fill Address Register (FILL_ADDR)

The FILL_ADDR is a read-only register that contains the physical address associated with errors reported by BIU_STAT bits [14:8]. Its contents are meaningful only when FILL_ECC or FILL_DPERR is set. Reads of the FILL_ADDR unlock FILL_ADDR, BIU_STAT bits [14:8] and FILL_SYNDROME.

The FILL_ADDR bits [33:5] identify the 32-byte cache block that the CPU was attempting to read when the error occurred.

If the FILL_IRD bit of the BIU_STAT register is clear, it indicates that the error occurred during a D-stream cache fill. At such times, FILL_ADDR bits [4:2] contain bits [4:2] of the physical address generated by the load instruction that triggered the cache fill. If FILL_IRD is set, then FILL_ADDR bits [4:2] are UNPREDICTABLE. The FILL_ADDR bits [63:34] and FILL_ADDR bits [1:0] will read as zero. Figure 5–31 shows the fill address register (FILL_ADDR) format.

Figure 5–31 Fill Address Register



5.3.21 Fill Syndrome Register (FILL_SYNDROME)

The FILL_SYNDROME register is a 14-bit read-only register.

If the chip is in ECC mode and an ECC error is recognized during a primary cache fill operation, the syndrome bits associated with the bad quadword are locked in the FILL_SYNDROME register. The FILL_SYNDROME bits [6:0] contain the syndrome associated with the lower longword of the quadword, and FILL_SYNDROME bits [13:7] contain the syndrome associated with the upper longword of the quadword. A syndrome value of zero means that no errors were found in the associated longword. See Table 5–18 for a list of syndromes associated with correctable single-bit errors. The FILL_SYNDROME register is unlocked when the FILL_ADDR register is read.

If the chip is in parity mode and a parity error is recognized during a primary cache fill operation, the FILL_SYNDROME register indicates which of the longwords in the quadword got bad parity. The FILL_SYNDROME bit [0] is set to indicate that the lower longword was corrupted, and FILL_SYNDROME bit [7] is set to indicate that the upper longword was corrupted. The FILL_SYNDROME bits [13:8] and [6:1] are RAZ in parity mode. Figure 5–32 shows the fill syndrome register format.

Figure 5–32 FILL_SYNDROME Register

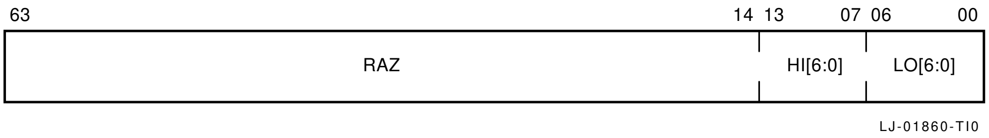


Table 5–18 Syndromes for Single-Bit Errors

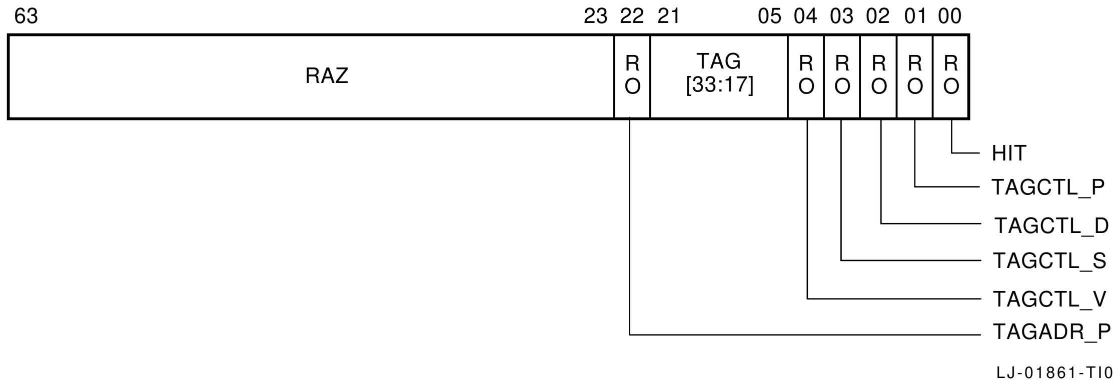
Data Bit	Syndrome (Hex)	Data Bit	Syndrome (Hex)	Check Bit	Syndrome (Hex)
00	4F	16	0E	00	01
01	4A	17	0B	01	02
02	52	18	13	02	04
03	54	19	15	03	08
04	57	20	16	04	10
05	58	21	19	05	20
06	5B	22	1A	06	40
07	5D	23	1C		
08	23	24	62		
09	25	25	64		
10	26	26	67		
11	29	27	68		
12	2A	28	6B		
13	2C	29	6D		
14	31	30	70		
15	34	31	75		

5.3.22 Backup Cache Tag Register (BC_TAG)

The BC_TAG is a read-only register. Unless locked, the BC_TAG register is loaded with the results of every backup cache tag probe. When a tag or tag control parity error or primary fill data error (parity or ECC) occurs, this register is locked against further updates. The software may read the LSB of this register by using the HW_MFPR instruction. Each time an HW_MFPR from BC_TAG completes, the contents of BC_TAG are shifted one bit position to the right, so that the entire register can be read using a sequence of HW_MFPRs. The software may unlock the BC_TAG register using a HW_MTPR to BC_TAG.

Successive HW_MFPRs from the BC_TAG register must be separated by at least one null cycle. Figure 5–33 shows the backup cache tag register format. Table 5–19 lists the register fields and gives a description of each.

Figure 5–33 Backup Cache Tag Register



Note

Unused tag bits in the TAG field of this register are always clear, based on the size of the external cache as determined by the BC_SIZE field of the BIU_CTL register.

Table 5–19 Backup Cache Tag Register Fields

Field	Type	Description
TAGADR_P	RO	Reflects the state of the **tagAdrP_h** signal of the 21064 /21064A when a tag, tag control, or data parity error occurs.
TAG	RO	Contains the tag that is being currently probed.
TAGCTL_V	RO	Reflects the state of the **tagCtlV_h** signal of the 21064 /21064A when a tag, tag control, or data parity error occurs.
TAGCTL_S	RO	Reflects the state of the **tagCtlS_h** signal of the 21064 /21064A when a tag, tag control, or parity error occurs.
TAGCTL_D	RO	Reflects the state of the **tagCtlD_h** signal of the 21064 /21064A when a tag, tag control, or data parity error occurs.
TAGCTL_P	RO	Reflects the state of the **tagCtlP_h** signal of the 21064 /21064A when a tag, tag control, or data parity error occurs.
HIT		When set, indicates that there was a tag match when a tag, tag control, or data parity error occurred.

5.4 PAL_TEMP Registers

The CPU chip contains 32 (64-bit) registers that are accessible by way of the HW_MxPR instructions. These registers provide temporary storage for PALcode.

5.5 Lock Registers

There are two registers per processor that are associated with the LDQ_L/LDL_L and STQ_C/STL_C instructions: the lock_flag register and the locked_physical_address register. The use of these registers is described in the *Alpha Architecture Reference Manual*. These registers are required by the architecture but are *not* implemented on the 21064/21064A. They must be implemented in the application. See Section 6.4.10 for 21064/21064A lock operation.

5.6 Internal Processor Registers Reset State

Table 5–20 lists the state of all the internal processor registers (IPRs) immediately following reset. The table also specifies which registers need to be initialized by power-up PALcode.

Table 5–20 Internal Process Register Reset State

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

(continued on next page)

Table 5–20 (Cont.) Internal Process Register Reset State

IPR	Reset State	Comments
ASTER	UNDEFINED	PALcode must initialize.
SL_XMIT	UNDEFINED	PALcode must initialize. Appears on external pin.
TB_CTL	UNDEFINED	PALcode must select between SP /LP DTB prior to any TB fill.
DTB_PTE	UNDEFINED	
DTB_PTE_TEMP	UNDEFINED	
MM_CSR	UNDEFINED	Unlocked on reset.
VA	UNDEFINED	Unlocked on reset.
DTBZAP	n/a	PALcode must do a DTBZAP on reset before writing the DTB (must do HW_MTPR to DTBZAP register).
DTBASM	n/a	
DTBIS	n/a	
BIU_ADDR	UNDEFINED	Potentially locked.
BIU_STAT	UNDEFINED	Potentially locked.
SL_CLR	UNDEFINED	PALcode must initialize.
DC_STAT	UNDEFINED	Potentially locked. **21064 only**
C_STAT	UNDEFINED	Potentially locked. **21064A only**
FILL_ADDR	UNDEFINED	Potentially locked.
ABOX_CTL	cleared	Write buffer enabled, machine checks disabled, correctable read interrupts disabled, Icache stream buffer disabled, super pages 1 and 2 disabled, endian mode disabled, Dcache disabled, forced hit mode off. (STC_NORESULT disabled, NCACHE_NDISTURB disabled)
ALT_MODE	UNDEFINED	
CC	UNDEFINED	Cycle counter is disabled on reset.

(continued on next page)

Table 5–20 (Cont.) Internal Process Register Reset State

IPR	Reset State	Comments
CC_CTL	UNDEFINED	
BIU_CTL	cleared	Bcache disabled, parity mode enabled, chip enable asserts during RAM write cycles, Bcache forced-hit mode disabled. BC_PA_DIS field cleared. BAD_TCP cleared. BAD_DP cleared. DELAY_WDATA cleared. SYS_WRAP cleared.
FILL_SYNDROME	UNDEFINED	Potentially locked.
BC_TAG	UNDEFINED	Potentially locked.
PAL_TEMP [31:0]	UNDEFINED	

Note

The Bcache parameters listed here are all undetermined on reset and must be initialized in the BIU_CTL register before enabling the Bcache.

- Bcache RAM read speed (BC_RD_SPD)
- Bcache RAM write speed (BC_WR_SPD)
- Bcache delay write data (DELAY_WDATA)
- Bcache write enable control (BC_WE_CTL)
- Bcache size (BC_SIZE)

6 External Interface

6.1 Introduction

This chapter is organized as follows:

- Introduction
- **21064** and **21064A** Logic Symbols
- Signal Names and Functions
- Bus Transactions
- Interface Operation
- Hardware Error Handling

Note

Although the 21064/21064A is configured during reset to use either a 64-bit or 128-bit wide external data bus, most of this chapter describes the chip's operation in 128-bit mode. Section 6.5.6 describes details specific to 64-bit mode operation.

6.2 Logic Symbol

Figure 6–1 shows the logic symbol of the **21064** while Figure 6–2 shows the logic symbol for the **21064A**.

Figure 6–1 21064 Logic Symbol

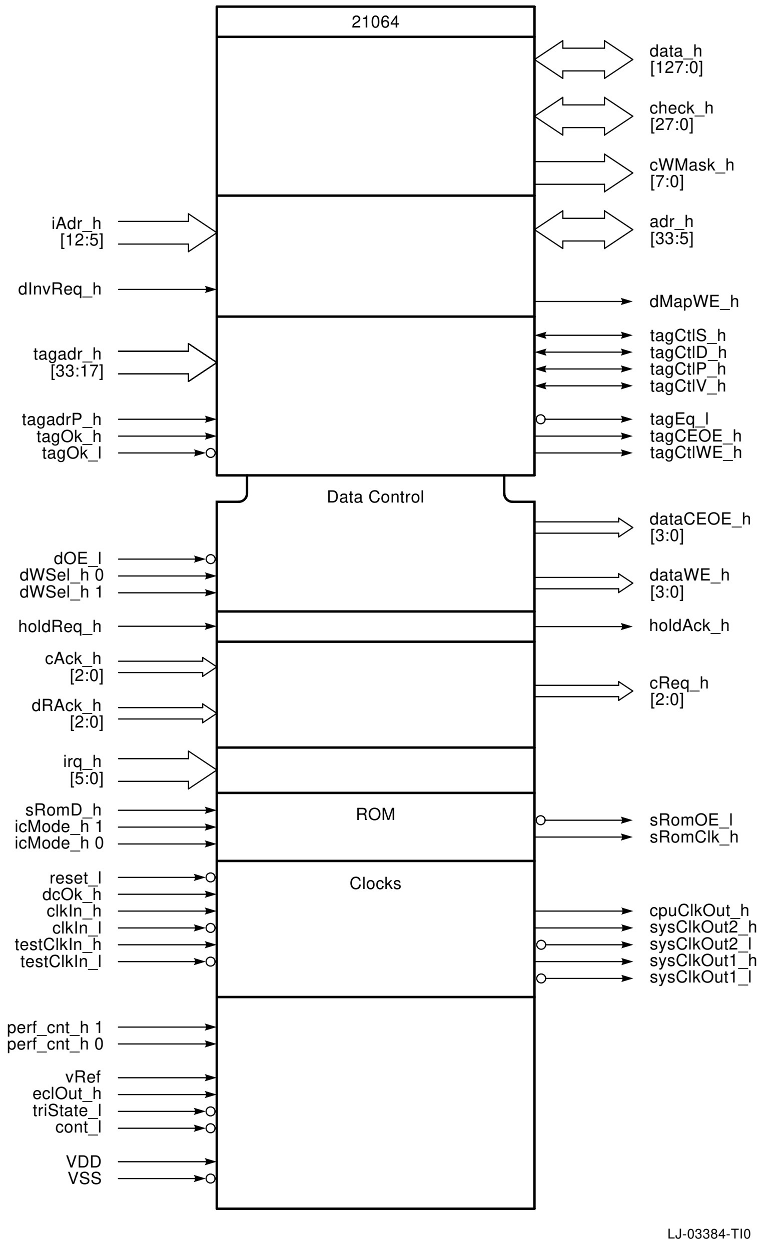
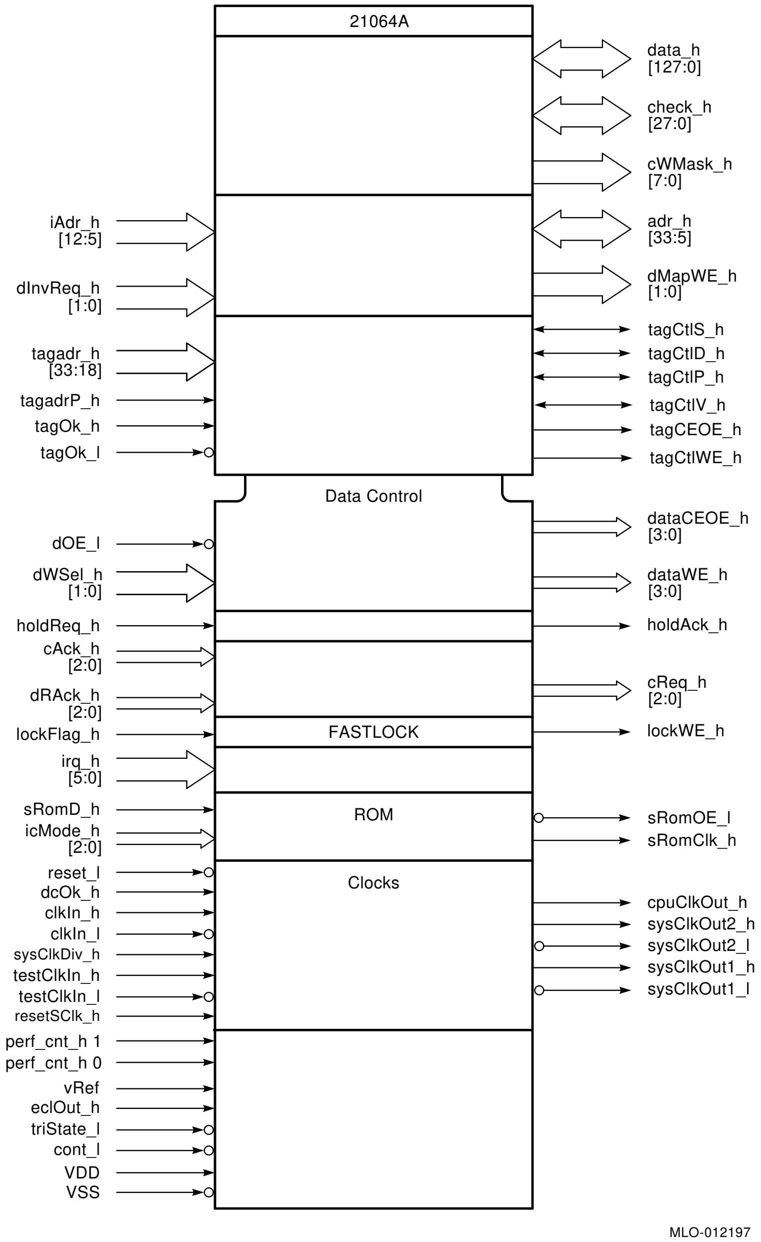


Figure 6–2 21064A Logic Symbol



6.3 Signal Names and Functions

Table 6–1 through Table 6–11 list the various signals grouped by function. The "Type" column identifies a signal as input (I), output (O), or bidirectional (B).

Signals with an **_h** suffix are active (asserted) when high. Those with an **_l** suffix are active (asserted) when low.

Signals which are unique to either the **21064** or **21064A** are identified and the differences stated.

Table 6–1 Data, Address, and Parity/ECC Buses

Signal	Type	Count	Function
data_h [127:0]	B	128	Bidirectional signals providing the data path between the 21064/21064A and the system.
adr_h [33:5]	B	29	Bidirectional signals providing the address path between the 21064/21064A and the system. These address bits provide granularity down to 32-byte internal cache blocks.
check_h [27:0]	B	28	Bidirectional signals providing a path for parity or ECC bits between the 21064/21064A and the rest of the system.

For data, address, and parity/ECC bus operation information, see Section 6.5.9.

Table 6–2 Primary Cache Invalidate

Signal	Type	Count	Function
iAdr_h [12:5]	I	8	Used to index blocks in the Dcache for Dcache invalidates.
dInvReq_h	I	1	**21064 only**—Used by external logic to invalidate the Dcache entry indexed by **iAdr_h**.
dInvReq_h [1:0]	I	2	**21064A only**—Used by external logic to invalidate the Dcache indexed by **iAdr_h**. Each signal line selects one half of the 16K byte Dcache.

For primary cache invalidate operation information, see Section 6.5.3.

Table 6–3 External Cache Control

Signal	Type	Count	Function
tagCEOE_h	O	1	Controls tag and tag control RAM chip enable or output enable during the 21064/21064A controlled external cache accesses.
tagCtlWE_h	O	1	Controls tag control RAM write enable during the 21064/21064A controlled transactions.
tagCtlV_h, tagCtlS_h, tagCtlD_h	B	3	Read/write path for external cache valid, shared, and dirty bits. The following combinations of the **tagCtl** RAM bits are allowed. The **tagCtlS_h** bit can be viewed as a write protect bit.

tagCtlV_h	tagCtlS_h	tagCtlD_h	Status
L	X	X	Invalid
H	L	L	Valid, private
H	L	H	Valid, private, dirty
H	H	L	Valid, shared
H	H	H	Valid, shared, dirty

Signal	Type	Count	Function
tagCtlP_h	B	1	Carries parity across **tagCtlV_h**, **tagCtlD_h**, and **tagCtlS_h**.
tagAdr_h [33:17]	I	17	**21064 only** – Transfers the contents of the **tagAdr** RAM to the **21064**'s address comparator and parity checker.
tagAdr_h [33:18]	I	16	**21064A only**— Transfers the contents of the **tagAdr** RAM to the **21064A**'s address comparator and parity checker.

(continued on next page)

Table 6–3 (Cont.) External Cache Control

Signal	Type	Count	Function
tagAdrP_h	I	1	Transfers the contents of the **tagAdr** RAM to the 21064/21064A's address comparator and parity checker.
tagOk_h, tagOk_l	I	2	Bus interface control signals that allow external logic to stall a CPU-controlled access to the external cache RAMs at the last possible moment. Synchronization of these signals with the CPU clock differs between the **21064** and **21064A**. See Section 7.4.7 and Section 7.4.8.
tagEq_l	O	1	**21064 only** — Asserted by the **21064** during external cache hold if the result of tag equality comparison is true.
dataCEOE_h [3:0]	O	4	Controls data RAMs' output enable or chip enable during the 21064/21064A controlled cache accesses.
dataWE_h [3:0]	O	4	Controls data RAMs' write enable during the 21064 /21064A controlled cache accesses.
dataA_h [4:3]	O	2	Controls data RAMs' address bits [4] and [3] during the 21064/21064A controlled cache accesses.
holdReq_h	I	1	Asserted by external logic to gain access to the external cache.
holdAck_h	O	1	Asserted by the 21064/21064A to indicate that external logic has access to the external cache.
dMapWE_h	O	1	**21064 only** — Controls the write enable input of the (optional) backmap RAM during the **21064** controlled external cache reads.
dMapWE_h [1:0]	O	2	**21064A only** — Controls the write enable input of the (optional) backmap RAM during the **21064A** controlled external cache reads. The signal lines indicate which half of the 16K byte Dcache is being allocated.

For external cache control operation information, see Section 6.5.4.

Table 6–4 External Cycle Control

Signal	Type	Count	Function
dOE_l	I	1	Used by external logic to tell the 21064/21064A to drive the data bus during external write transactions.
dWSel_h [1:0]	I	2	Used by external logic to tell the 21064/21064A which part of the 32-byte block of write data should be driven onto the data bus. The relationship between **dWSel_h [1:0]** and the selected bytes of the 32-byte block is shown below:

dWSel_h [1:0]	Selected Bytes (128-bit data bus)	Selected Bytes (64-bit data bus)
0 0	[15:00]	[07:00]
0 1	N/A	[15:08]
1 0	[31:16]	[23:16]
1 1	N/A	[31:24]

Signal	Type	Count	Function
dRAck_h [2:0]	I	3	Inform the 21064/21064A that read data is valid on the data bus, whether data should be cached in the 21064/21064A internal caches, and whether ECC or parity checking should be attempted. Read data acknowledge types are:

dRAck_h 2	dRAck_h 1	dRAck_h 0	Type
L	L	L	IDLE
H	L	L	OK_ NCACHE_ NCHK
H	L	H	OK_ NCACHE
H	H	L	OK_NCHK
H	H	H	OK

(continued on next page)

Table 6–4 (Cont.) External Cycle Control

Signal	Type	Count	Function
cReq_h [2:0]	O	3	Used by the 21064/21064A to specify a cycle type at the start of an external cycle. The cycle types are:

cReq_h 2	cReq_h 1	cReq_h 0	Type
L	L	L	IDLE
L	L	H	BARRIER
L	H	L	FETCH
L	H	H	FETCH_M
H	L	L	READ_BLOCK
H	L	H	WRITE_BLOCK
H	H	L	LDL_L/LDQ_L
H	H	H	STL_C/STQ_C

Signal	Type	Count	Function
cWMask_h [7:0]	O	8	Supply longword write masks to external logic during write cycle and contains cache miss information during other cycles (see Section 6.5.5.2).
cAck_h [2:0]	I	3	Used by external logic to acknowledge an external cycle. Acknowledgment types are:

cAck_h 2	cAck_h 1	cAck_h 0	Type
L	L	L	IDLE
L	L	H	HARD_ERROR
L	H	L	SOFT_ERROR
L	H	H	STL_C_FAIL /STQ_C_FAIL
H	L	L	OK

For operation external cycle control operation information, see Section 6.5.5.

Table 6–5 Interrupts

Signal	Type	Count	Function
irq_h [5:0]	I	6	Compose the interrupt bus, which provides six types of external interrupts to the 21064/21064A during normal operation and provide initialization information at reset.
			When **reset_l** is asserted, the **irq_h 5** bit is used to select 128-bit or 64-bit mode. If **irq_h 5** is asserted then 128-bit mode is selected.
			When **reset_l** is asserted, the **irq_h [4:3]** bits encode the delay, in CPU clock cycles, from **sysClkOut1** to **sysClkOut2**, as follows:

irq_h 4	irq_h 3	Delay
L	L	0
L	H	1
H	L	2
H	H	3

21064 only — When reset_l is asserted, the **irq_h [2:0]** bits encode the value of the divisor used to generate the system clock from the CPU clock, as follows:

irq_h 2	irq_h 1	irq_h 0	Ratio
L	L	L	2
L	L	H	3
L	H	L	4
L	H	H	5
H	L	L	6
H	L	H	7
H	H	L	8
H	H	H	8

(continued on next page)

Table 6–5 (Cont.) Interrupts

<table>
<tr><th>Signal</th><th>Type</th><th>Count</th><th>Function</th></tr>
<tr><td>sysClkDiv_h[1]</td><td>I</td><td>1</td><td>21064A only — At reset this line provides initialization information to the 21064A.

21064A only — When reset_l is asserted, sysClkDiv_h and the irq_h [2:0] bits encode the value of the divisor used to generate the system clock from the CPU clock, as follows:</td></tr>
</table>

sysClkDiv_h[1]	irq_h [2:0]	Ratio
L	L L L	2
L	L L H	3
L	L H L	4
L	L H H	5
L	H L L	6
L	H L H	7
L	H H L	8
L	H H H	9
H	L L L	10
H	L L H	11
H	L H L	12
H	L H H	13
H	H L L	14
H	H L H	15
H	H H L	16
H	H H H	17

[1]sysClkDiv_h at PGA location AA16 was a spare pin on the **21064**.

For interrupts operation information, see Section 6.5.8. For information on power-up of the **21064**, see Appendix A.

Table 6–6 Instruction Cache Initialization/Serial ROM Interface

<table>
<tr><th>Signal</th><th>Type</th><th>Count</th><th>Function</th></tr>
<tr><td>icMode_h [1:0]</td><td>I</td><td>2</td><td>21064 only — Determines which of three Icache initialization modes is used after reset. The 21064 implements three Icache modes to support chip and printed circuit board level testing.
<table>
<tr><th>icMode_h 1</th><th>icMode_h 0</th><th>Mode</th></tr>
<tr><td>L</td><td>L</td><td>Serial ROM</td></tr>
<tr><td>L</td><td>H</td><td>Disabled</td></tr>
<tr><td>H</td><td>L</td><td>Digital reserved</td></tr>
<tr><td>H</td><td>H</td><td>Digital reserved</td></tr>
</table></td></tr>
<tr><td>icMode_h [2:0]</td><td>I</td><td>3</td><td>21064A only — Determines which Icache initialization mode is used after reset. The 21064A implements several Icache modes used by Digital to support chip and module level testing.
<table>
<tr><th colspan="3">icMode_h [2:0]</th><th>Mode</th></tr>
<tr><td>L</td><td>L</td><td>L</td><td>Serial ROM</td></tr>
<tr><td>L</td><td>L</td><td>H</td><td>Disabled</td></tr>
<tr><td colspan="3">Other six combinations</td><td>Digital reserved</td></tr>
</table></td></tr>
<tr><td>sRomOE_l</td><td>O</td><td>1</td><td>In serial ROM mode, supplies the output enable to the external serial ROM, serving both as an output enable and as a reset.</td></tr>
<tr><td>sRomD_h</td><td>I</td><td>1</td><td>In serial ROM mode, inputs external serial ROM data to the 21064/21064A.</td></tr>
<tr><td>sRomClk_h</td><td>O</td><td>1</td><td>In serial ROM mode, supplies the clock to the external serial ROM that causes it to advance to the next bit.

The signals sRomOE_l, sRomD_h, and sRomClk_h also serve as simple parallel I/O pins to drive a diagnostic terminal.</td></tr>
</table>

For Icache initialization/serial ROM interface operation information, see Section 6.5.7.

Table 6–7 Initialization

Signal	Type	Count	Function
dcOk_h	I	1	Switches clock sources between an on-chip ring oscillator and the external clock oscillator to provide the chip clock.
reset_l	I	1	Forces the CPU into a known state.
resetSClk_h	I	1	**21064A only** — A test signal. It forces the system clock divider into a known state.

For initialization operation information, see Section 6.5.2.

Table 6–8 Fast Lock Mode Signals (21064A only)

Signal	Type	Count	Function
lockWE_h[1]	O	1	The **21064A** is able to probe Bcache for a LDxL transaction. If there is a Bcache hit the **21064A** will assert **lockWE_h** allowing external logic to set a lock flag bit and load a lock address register.
lockFlag_h[2]	I	1	This signal line allows external logic to indicate the state of the lock flag bit (set or clear). When the **21064A** performs a STxC transaction it may probe the Bcache and test this signal. If the signal is asserted the **21064A** will perform the write to Bcache while asserting **lockWE_h** allowing the external logic to clear the lock flag bit.

[1]**lockWE_h** at PGA location P24 is used for the signal **tagEq_l** by the **21064**.

[2]**lockFlag_h** at PGA location R23 is used for the signal **tagAdr_h 17** by the **21064**.

Table 6–9 Performance Monitoring

Signal	Type	Count	Function
perf_cnt_h [1:0]	I	2	Provides 21064/21064A internal performance monitoring hardware access to off-chip events.

For performance monitoring operation information, see Section 5.2.3, Section 5.3.15 and Section 6.5.10.

Table 6–10 Clocks

<table>
<tr><th>Signal</th><th>Type</th><th>Count</th><th>Function</th></tr>
<tr><td>clkIn_h, clkIn_l</td><td>I</td><td>2</td><td>Supply the 21064/21064A with a differential clock from external logic.</td></tr>
<tr><td>testClkIn_h, testClkIn_l</td><td>I</td><td>2</td><td>These two input signals tell the 21064/21064A which clocks will be applied to the input clock signal lines clkIn_h and clkIn_l.
<table>
<tr><th>testClkIn_h</th><th>testClkIn_l</th><th>Function</th></tr>
<tr><td>L</td><td>L</td><td>Digital Reserved</td></tr>
<tr><td>L</td><td>H</td><td>Standard 2x input clock</td></tr>
<tr><td>H</td><td>L</td><td>Standard 2x input clock</td></tr>
<tr><td>H</td><td>H</td><td>1x input clock</td></tr>
</table></td></tr>
<tr><td>cpuClkOut_h</td><td>O</td><td>1</td><td>Supplies the internal chip clock for use by the external interface; the low-to-high transition of cpuClkOut_h is the ‘‘CPU clock’’ used in the timing specification for the tagOk_h and tagOk_l signals.</td></tr>
<tr><td>sysClkOut1_h, sysClkOut1_l</td><td>O</td><td>2</td><td>Provide the system clock for use by the external interface. The low-to-high transition of sysClkOut1_h provides the system clock used as a timing reference throughout this document.</td></tr>
<tr><td>sysClkOut2_h, sysClkOut2_l</td><td>O</td><td>2</td><td>Provide delayed system clock to the external interface. The delay is between zero and three CPU clock cycles. The delay is dependent upon the state of irq_h [4:3] when reset_l is asserted.</td></tr>
</table>

For clocks operation information, see Section 6.5.1.

Table 6–11 Other Signals

Signal	Type	Count	Function
tristate_l	I	1	The assertion of this signal forces all the 21064 /21064A signals, with the exception of **cpuClkOut_ h**, to the high-impedance state.
cont_l	I	1	The assertion of this signal causes the 21064 /21064A to connect all signals to Vss, with the exception of certain clock signals and **vRef**.
vRef	I	1	Supplies a reference voltage of 1.4 V to the input signal sense circuits.
eclOut_h	I	1	Digital reserved; should be tied to Vss.

For miscellaneous signals operation information, see Section 6.5.11.

6.4 Bus Transactions

This section describes bus transactions in detail. These transactions are described for 128-bit data bus mode; see Section 6.5.6 for more information on 64-bit bus mode.

6.4.1 Reset

External logic resets the 21064/21064A by asserting **reset_l**. When the 21064 /21064A detects the assertion of **reset_l**, it terminates all external activity, and places the output signals on the external interface into the states shown in Table 6–12.

Note

All of the control signals have been placed in the state that allows external devices access to the external cache. Under normal operation, this can only be done using the **holdReq** cycle.

Table 6–12 State of Pins at Reset

Pin	State	Pin	State
clkIn_h, clkIn_l	I	tagAdr_h	I
testClkIn_h,testClkIn_l	I	tagAdrP_h	I
cpuClkOut_h	C	tagOk_h, tagOk_l	I
sysClkOut1_h,sysClkOut1_l	C	tagEq_l (**21064 only**)	U
sysClkOut2_h, sysClkOut2_l	C	dataCEOE_h	L
dcOk_h	I	dataWE_h	L
reset_l	I	dataA_h [4:3]	L
icMode_h	I	holdReq_h	I
sRomOE_l	H	holdAck_h	L
sRomD_h	I	cReq_h	L
sRomClk_h	H	cWMask_h	U
adr_h	Z	cAck_h	I
data_h	Z	iAdr_h	I
check_h	Z	dInvReq_h	I
dOE_l	I	dMapWE_h	L
dWSel_h	I	irq_h	I
dRAck_h	I	vRef	I
tagCEOE_h	L	eclOut_h	I
tagCtlWE_h	L	perf_cnt_h [1:0]	I
tagCtlV_h	Z	tristate_l	I
tagCtlS_h	Z	cont_l	I
tagCtlD_h	Z	tagCtlP_h	Z
lockWE_h (**21064A only**)	?	lockFlag_h (**21064A only**)	I
sysClkDiv_h (**21064A only**)	I	resetSClk_h (**21064A only**)	I

H = High
L = Low
U = Unpredictable
I = Chip inputs
C = Continuously cycling (clocks)
Z = Tristate

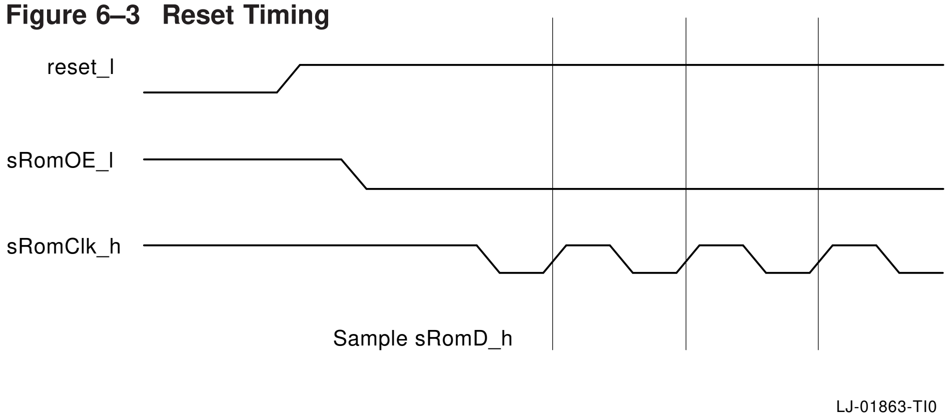
External logic can asynchronously deassert **reset_l**. The 21064/21064A contains internal logic to keep its internal reset signal asserted at least 20 CPU cycles beyond the deassertion of **reset_l**.

When the 21064/21064A detects reset_l going high, it can load bits from an external serial ROM into its internal Icache, based on the value placed on

icMode_h [1:0] for the **21064**
icMode_h [2:0] for the **21064A**

Figure 6–3 shows the SROM timing for the first three bit samples.

Figure 6–3 Reset Timing

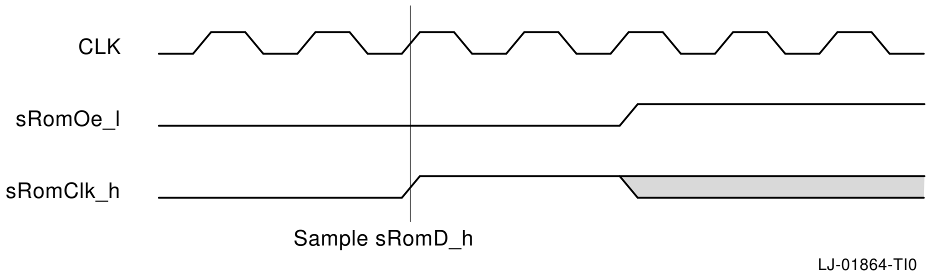


- When **reset_l** is asserted, **sRomOE_l** is deasserted and **sRomClk_h** is asserted.
- The 21064/21064A's internal reset signal remains asserted at least 20 CPU cycles after **reset_l** deasserts, when **sRomOe_l** asserts.
- The first rising edge of **sRomClk_h** occurs
  - For the **21064**, 128 CPU cycles after **sRomOe_l** asserts, and every 126 CPU cycles thereafter.
  - For the **21064A**, 255 CPU cycles after **sRomOe_l** asserts, and every 254 CPU cycles thereafter.
- The 21064/21064A samples **sRomD_h** in the last half of each CPU cycle before the rising edge of **sRomClk_h**.

This sequence continues until the Icache is loaded. There are 256 blocks in the Icache that can be loaded from the SROM. Each block contains 293 bits, 75,008 bits in all, resulting in 75,008 rising edges of **sRomClk_h**.

Figure 6–4 shows the end of the Icache preload sequence. The shaded area indicates unpredictable behavior.

Figure 6–4 Reset Timing — End of Preload Sequence



CLK refers to the 21064/21064A's internal CPU clock and is shown as a cycle reference.

1. The 21064/21064A samples the final serial ROM bit when **sRomClk_h** rises, as shown.
2. Two CPU cycles later, the 21064/21064A deasserts **sRomOe_l** and drives **sRomClk_h** with the value from the TMT bit of the SL_XMIT IPR. Since this bit is not initialized by chip reset, the value driven onto **sRomClk_h** is UNPREDICTABLE.

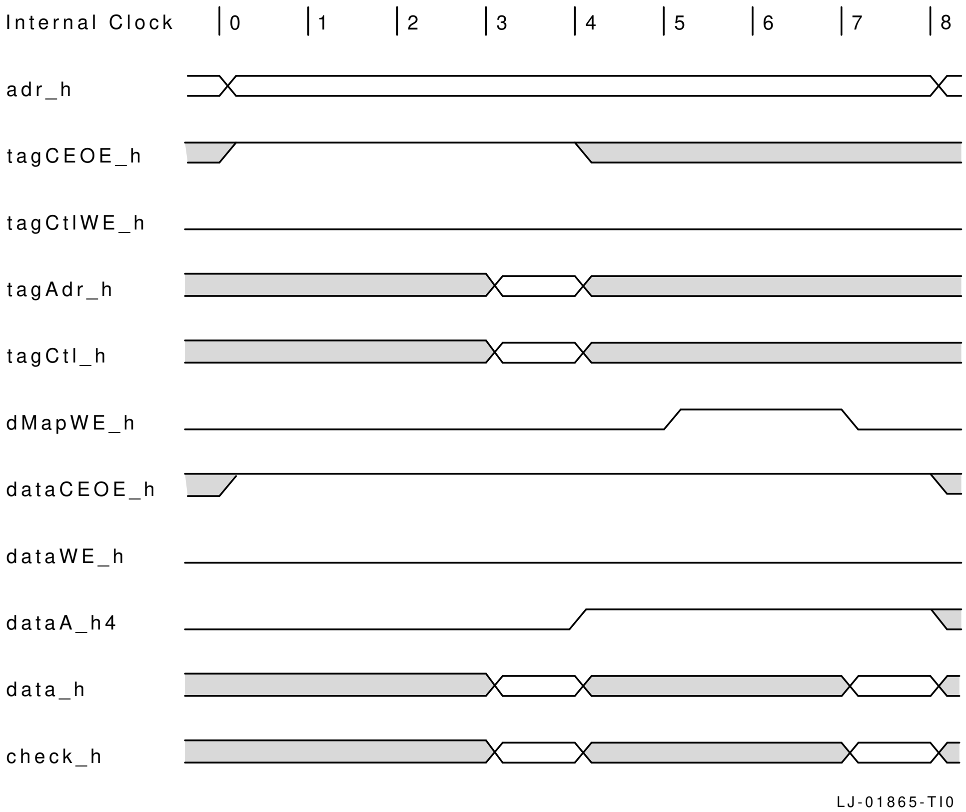
It is possible to disable the serial ROM mechanism altogether (see Section 6.5.7). In this case, since the Icache valid bits are cleared by reset, the first I-stream reference the 21064/21064A makes will miss the Icache and the 21064/21064A will generate an external request to address zero.

6.4.2 Fast External Cache Read Hit

A fast external cache read consists of a probe read (overlapped with the first data read), followed by the second data read if the probe hits.

In Figure 6–5, the external cache is using 4-cycle reads (BC_RD_SPD = 3), 4-cycle writes (BC_WR_SPD = 3), output enable control BIU_CTL [OE] = H), and a 2-cycle write pulse centered in the 4-cycle write (BC_WE_CTL [15:1] = LLLLLLLLLLLLLHH). The shaded areas indicate unpredictable levels.

Figure 6–5 Fast External Read Hit



If the probe misses, then the cycle aborts at the end of clock 3.

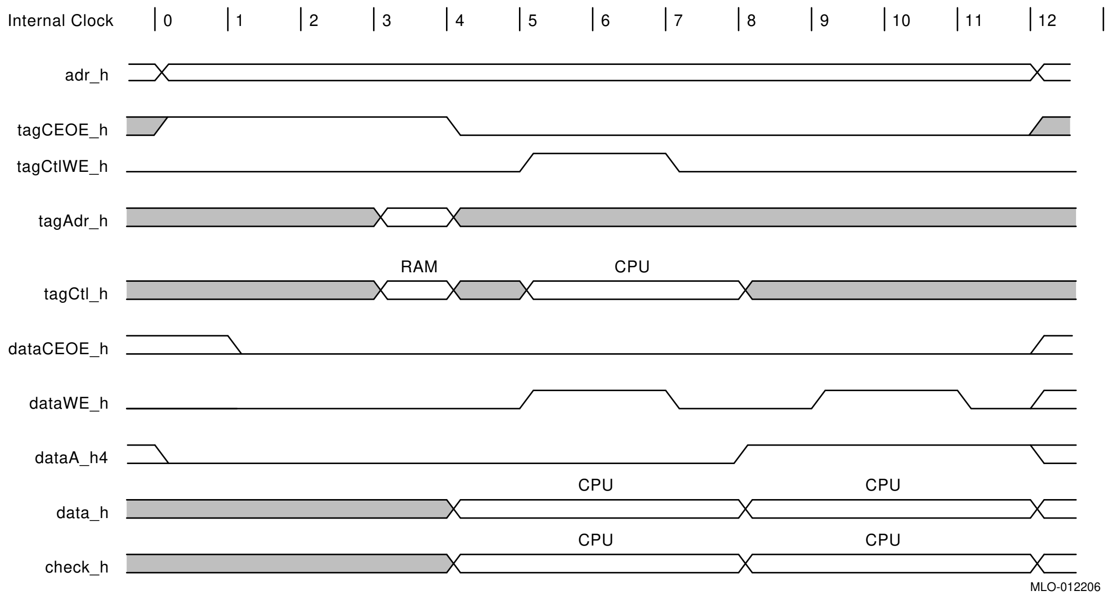
If the probe hits and the miss address had bit 4 set, then the two data reads would have been swapped , **dataA_h 4** would have been true in cycles 0, 1, 2, 3, and would have been false in cycles 4, 5, 6, 7.

6.4.3 Fast External Cache Write Hit

A fast external cache write consists of a probe read, followed by one or two data writes.

Figure 6–6 assumes that the external cache is using 4-cycle reads (BC_RD_SPD = 3), 4-cycle writes (BC_WR_SPD = 3), output enable control (BIU_CTL [OE] = H), and a 2-cycle write pulse centered in the 4-cycle write (BC_WE_CTL [15:1]) = LLLLLLLLLLLLLHH. The shaded areas indicate unpredictable levels.

Figure 6–6 Fast External Cache Write Hit



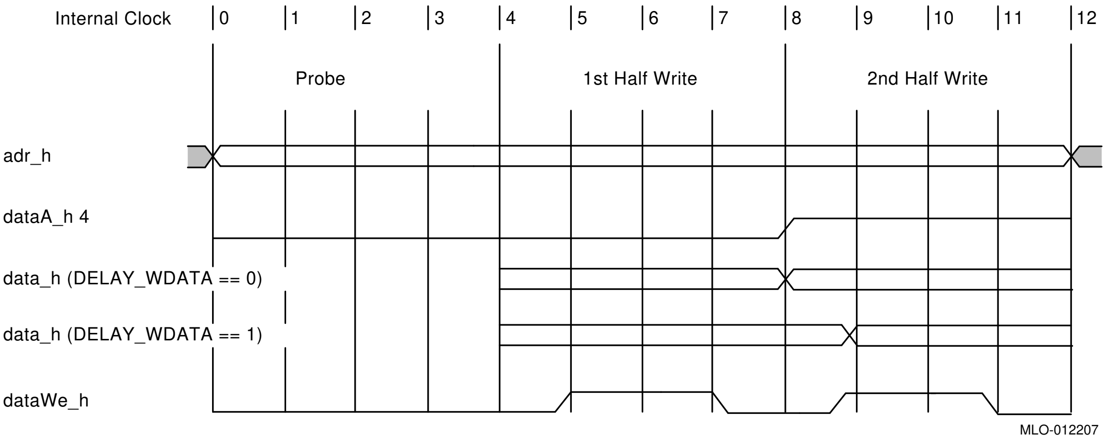
The 21064/21064A drives the **tagCtl_h** signals one CPU cycle later than it drives the **data_h** and **check_h** signals relative to the start of the write cycle. Unlike **data_h** and **check_h**, the **tagCtl_h** field must be read during the tag probe that precedes the write cycle. Because the 21064/21064A can switch its signals to a low impedance state much more quickly than most RAMs can switch their signals to a high impedance state, the 21064/21064A waits one CPU cycle before driving the **tagCtl_h** signals in order to minimize tristate driver overlap.

If the probe misses, then the cycle aborts at the end of clock 3.

6.4.4 External Cache Write Timing (Delayed Data)

The DELAY_WDATA bit of BIU_CTL controls the external write timing mode. When set, DELAY_WDATA changes the timing of the data bus during external cache writes as shown in Figure 6–7. Only the data bus timing associated with the first RAM write sequence is affected. The 21064/21064A puts the data bus in the high impedance state at its usual time, at the end of the second RAM write sequence. The diagram assumes a 4-cycle cache RAM read and write.

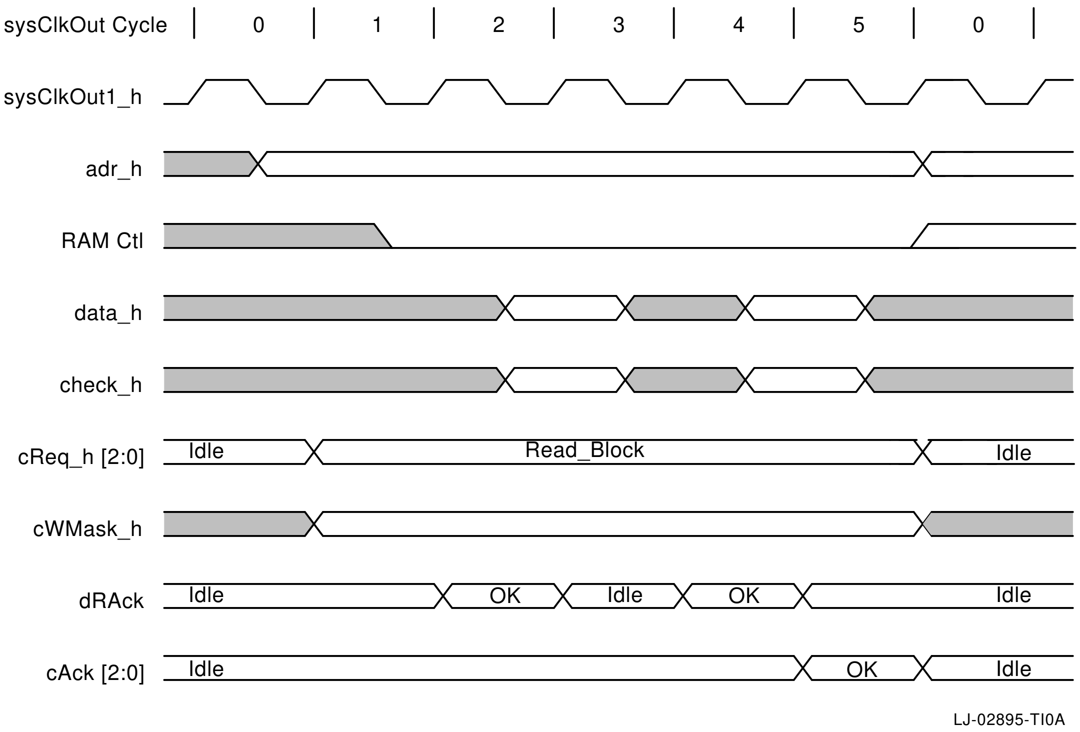
Figure 6–7 External Cache Write Timing



6.4.5 READ_BLOCK

A READ_BLOCK transaction, as shown in Figure 6–8, appears at the external interface on external cache read misses, either because it really was a miss, or because the external cache has not been enabled. The shaded areas indicate unpredictable levels.

Figure 6–8 READ_BLOCK Transaction



0. The **cReq_h** signals are always idle in the system clock cycle immediately before the beginning of an external transaction. The **adr_h** signals always change to their final value (with respect to a particular READ_BLOCK transaction) at least one CPU cycle before the start of the transaction.

1. The READ_BLOCK transaction begins. The 21064/21064A has already placed the address of the block containing the miss on **adr_h**.
   - The 21064/21064A places the quadword-within-block and the instruction/data (I/D) indication on **cWMask_h**.
   - The 21064/21064A places a READ_BLOCK command code on cReq_h.
   - The 21064/21064A clears the RAM control signals (**dataA_h [4:3]**, **dataCEOE_h [3:0]** and **tagCEOE_h**) no later than one CPU cycle after the system clock edge at which the transaction begins.
2. The external logic obtains the first 16 bytes of data. Although a single stall cycle has been shown here, there may be no stall cycles, or many stall cycles. Once the external logic has the first 16 bytes of data:
   - External logic places the data on the **data_h** and **check_h** buses.
   - External logic asserts **dRAck_h** to tell the 21064/21064A that the data and check bit buses are valid.
   - The 21064/21064A detects **dRAck_h** at the end of this cycle, and reads in the first 16 bytes of data at the same time.
3. The external logic obtains the second 16 bytes of data. Although a single stall cycle has been shown here, there could be no stall cycles, or many stall cycles.
4. The external logic has the second 16 bytes of data.
   - External logic places the data on the **data_h** and **check_h** buses.
   - External logic asserts **dRAck_h** to tell the 21064/21064A that the data and check bit buses are valid.
   - The 21064/21064A detects **dRAck_h** at the end of this cycle, and reads in the second 16 bytes of data at the same time.
5. External logic places an acknowledge code on **cAck_h** to tell the 21064 /21064A that the READ_BLOCK cycle is completed.

   The 21064/21064A detects the acknowledge at the end of this cycle, and can change the address.
6. Everything is idle. The 21064/21064A can start a new external cache cycle at this time. This is the same as cycle 0.

Because external logic owns the RAMs (as the chip has deasserted its RAM control signals at the start of the transaction), external logic can cache the data by asserting its write pulses on the external cache during cycles 2 and 4.

The 21064/21064A performs ECC checking (or parity checking) on the data supplied to it by the data and check buses if so requested by the acknowledge code. It is not necessary to place data into the external cache to get checking and correction.

Note

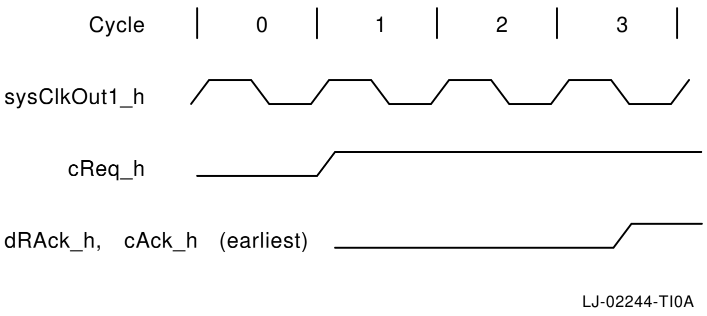
The following restriction applies to **21064** systems using a **sysClkOut** divisor equal to two and an external cache and **21064A** systems using a **sysClkOut** divisor equal to two with or without an external cache.

These systems must never respond to external reads by asserting **dRAck_h** or **cAck_h** earlier than the third system clock cycle of the transaction (see Figure 6–9).

If **cReq_h [2:0]** asserts in cycle 1, then system components must never assert **dRAck_h [2:0]** or **cAck_h [2:0]** before cycle 3.

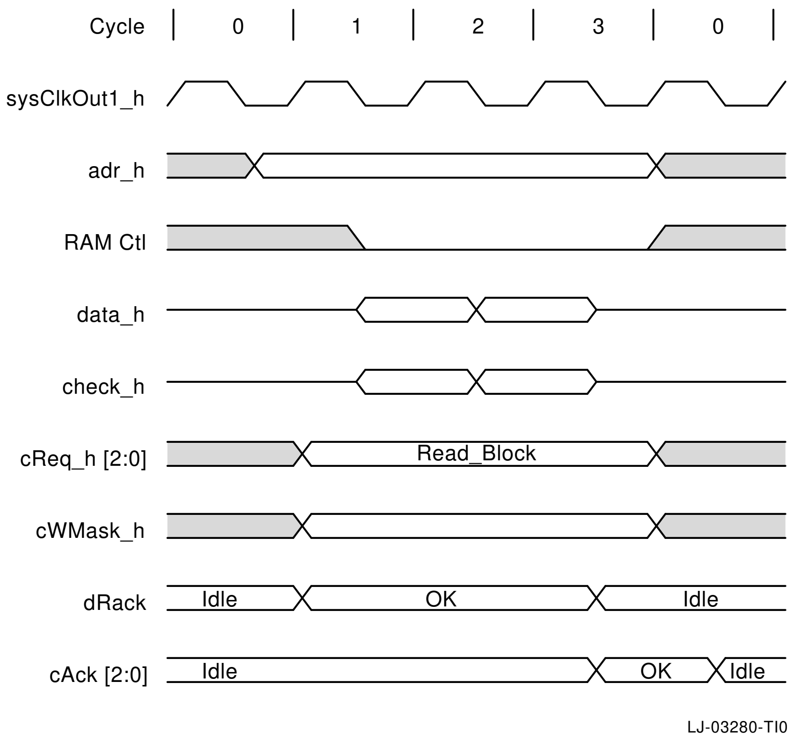
The behavior of the 21064/21064A is UNDEFINED if this restriction is violated.

Figure 6–9 Asserting dRack_h and cAck_h



Minimum cycle time for an external READ_BLOCK transaction is shown in Figure 6–10. The shaded area indicates unpredictable levels.

Figure 6–10 READ_BLOCK Transaction — Minimum Cycle Time



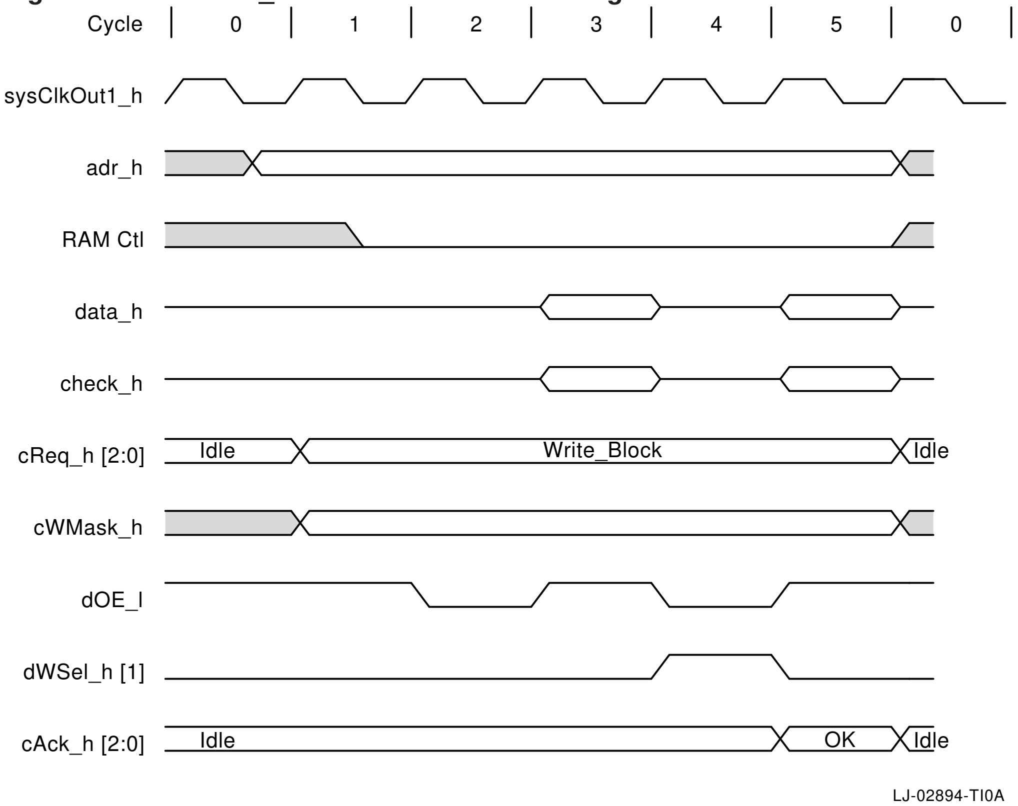
6.4.6 Shortened READ_BLOCK Transactions

For I/O operations, it may be desirable to transfer only the first 16 bytes of a READ_BLOCK transaction. This can be achieved by generating **cAck** after the first **dRAck**, and never generating a second **dRAck**.

6.4.7 WRITE_BLOCK

A WRITE_BLOCK transaction appears at the external interface on external cache write misses (either because it really was a miss, or because the external cache has not been enabled), or on external cache write hits to shared blocks. Figure 6–11 shows WRITE_BLOCK transaction timing. The shaded area indicates unpredictable levels.

Figure 6–11 WRITE_BLOCK Transaction Timing



0. The **cReq_h** signals are always idle in the system clock cycle immediately before the beginning of an external transaction. The **adr_h** pins always change to their final value (with respect to a particular WRITE_BLOCK transaction) at least three CPU cycles before the start of the transaction.

1. The WRITE_BLOCK cycle begins. The 21064/21064A has already placed the address of the block on **adr_h**. The 21064/21064A places a WRITE_BLOCK command code on **cReq_h** and the longword valid masks on **cWMask_h**.

   The 21064/21064A clears **dataCEOE_h [3:0]** at least one CPU cycle before the start of the transaction, and clears the other RAM control signals (**dataA_h [4:3]** and **tagCEOE_h**) at least one CPU cycle after the start of the transaction.

2. The external logic detects the command and asserts **dOE_l** to tell the 21064/21064A to drive the first 16 bytes of the block onto the data bus.

The timing shown for **dOE_l** is for discussion purposes—external logic can assert **dOE_l** by default and only deassert it when it needs to read the data RAMs, such as when writing back a victim block. If **dOE_l** were asserted before the start of the transaction, the 21064/21064A would begin to drive the data bus at the same time as it placed the WRITE_BLOCK command code on **cReq_h**.

3. The 21064/21064A drives the first 16 bytes of write data onto the **data_h** and **check_h** buses, and the external logic writes it into the destination. Although a single stall cycle has been shown here, there may be no stall cycles, or many stall cycles.
4. The external logic asserts **dOE_l** and **dWSel_h** to tell the 21064/21064A to drive the second 16 bytes of data onto the data bus.
5. The 21064/21064A drives the second 16 bytes of write data onto the **data_h** and **check_h** buses, and the external logic writes it into the destination. Although a single stall cycle has been shown here, there may be no stall cycles, or many stall cycles. In addition, the external logic places an acknowledge code on **cAck_h** to tell the 21064/21064A that the WRITE_BLOCK cycle is completed. The 21064/21064A detects the acknowledge at the end of this cycle, and changes the address and command to their next values. **dWSel_h** must be deasserted in this cycle.
6. Everything is idle. The 21064/21064A can start a new external cache access now. This is the same as cycle 0.

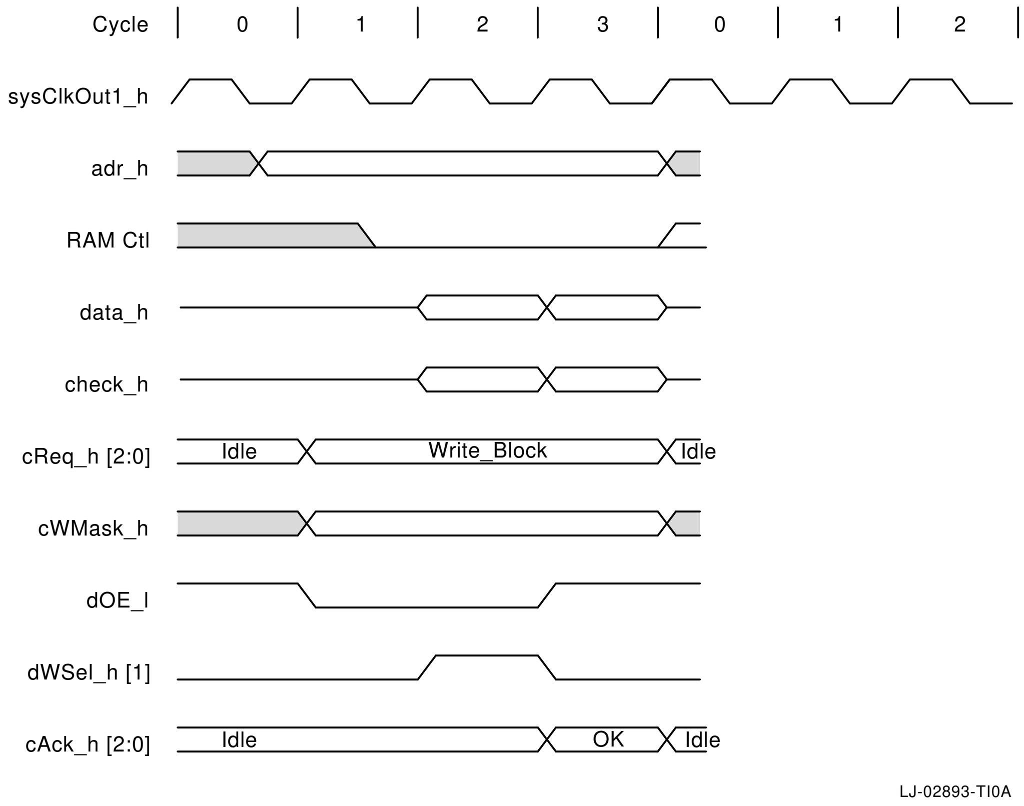
Because external logic owns the RAMs (because the chip has deasserted its RAM control signals at the beginning of the transaction), external logic can cache the data by asserting its write pulses on the external cache during cycles 3 and 5.

The 21064/21064A performs ECC generation (or parity generation) on data it drives onto the data bus.

Figure 6–11 shows external logic cycling through both 128-bit chunks of potential write data; however, this need not always be the case. External logic must pull from the 21064/21064A chip only those 128-bit chunks of data that contain valid longwords as specified by the **cWMask_h** signals. The only requirement is that if both halves are pulled from the 21064/21064A then the lower half must be pulled before the upper half.

Minimum cycle time for an external WRITE_BLOCK transaction is shown in Figure 6–12. (Figure 6–11 illustrates the WRITE_BLOCK transaction timing so that the functions of the signals involved are made clear.) The shaded area indicates unpredictable levels.

Figure 6–12 WRITE_BLOCK Transaction—Minimum Cycle Time

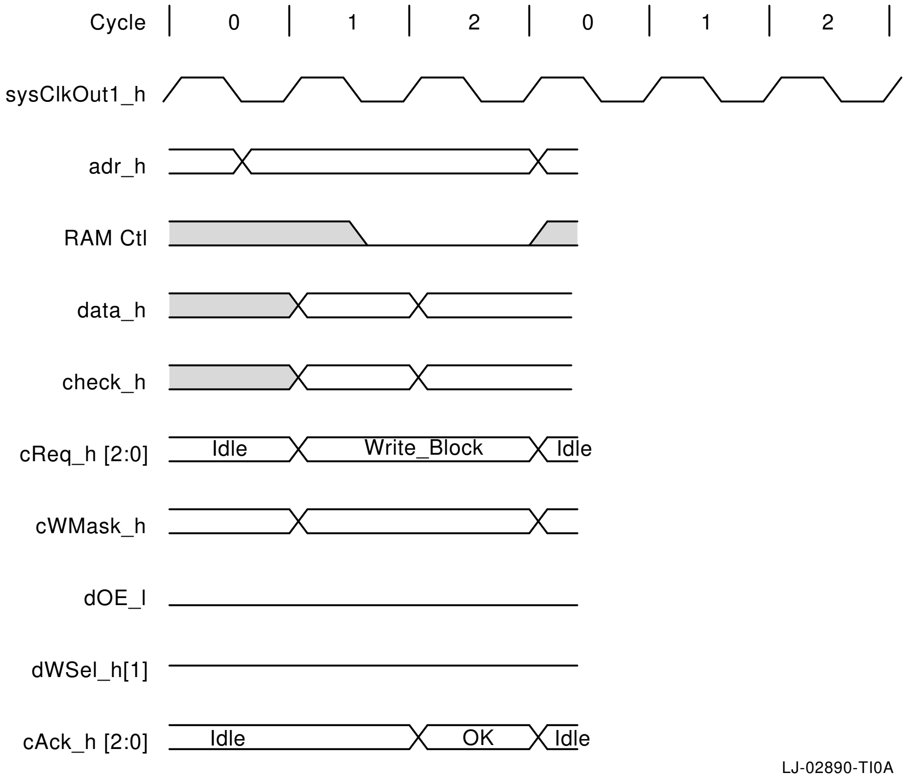


As shown, external logic asserts **dOE_l** by default, so that the 21064/21064A drives the first half of the write buffer entry coincident with its assertion of **cReq_h** in cycle 1. External logic must not assert **dWsel_h [1]** until after the WRITE_BLOCK transaction begins. It asserts **dWsel_h** in cycle 2, samples the second half of the write buffer entry in cycle 3, and terminates the transaction by asserting **cAck_h**.

6.4.8 Write Bandwidth in Systems Without an External Cache

To allow full 32-byte external WRITE_BLOCK transactions to complete in two sysClk cycles in 128-bit mode, **dWSel_h** can be held true even when **cReq_h** is idle. The 21064/21064A will only react to **dWSel_h [1]** when **cReq_h** is not idle. This will allow the external WRITE_BLOCK transaction timing as shown in Figure 6–13.

Figure 6–13 WRITE_BLOCK Transaction Timing Without an External Cache



Because external logic has already asserted **dOE_l**, the 21064/21064A will drive the first half of the write buffer line onto the data bus coincident with its assertion of **cReq_h** in cycle 1. The 21064/21064A will ignore **dWSel_h [1]** while **cReq_h** is idle, so its assertion will take effect in cycle 2. External logic will latch the second half of the write buffer line in cycle 2 and terminate the transaction in that cycle.

6.4.8.1 Write Buffer Unload Timing

The write bandwidth at the pins is reduced when the sysClk divider is two as the 21064/21064A produces two null sysClk cycles between each WRITE_BLOCK transaction.

The 21064/21064A will produce only one null sysClk cycle between WRITE_BLOCK transactions when the sysClk divider is three. Systems with a sysClk divider of three with no external cache will benefit from this feature.

6.4.9 Shortened WRITE_BLOCK Transactions

It may be desirable to transfer only the first 16 bytes of an I/O WRITE_BLOCK transaction. If so, terminate the transaction normally using **cAck_h [2:0]**, but without toggling **dWSel_h [1:0]**.

6.4.10 LDL_L/LDQ_L and STL_C/STQ_C Transactions

The 21064/21064A support LDL_L/LDQ_L and STL_C/STQ_C transactions which do not probe the external cache. The **21064A** also supports a fast lock mode where it does probe the external cache.

6.4.10.1 Transactions Without External Cache Probe

LDL_L/LDQ_L transactions appears at the external interface when an interlocked load instruction is executed. The external cache is not probed. With the exception of the command code output on the **cReq** signals, the LDL_L/LDQ_L transaction is exactly the same as a READ_BLOCK transaction. See Section 6.4.5.

An STL_C/STQ_C transaction appears at the external interface when a conditional store instruction is executed. The external cache is not probed. The STL_C/STQ_C transaction is the same as the WRITE_BLOCK transaction, with the following exceptions:

0. The code placed on the **cReq** signal is different.
1. The cWMask field will never validate more than a single longword or quadword of data.
2. External logic has the option of making the transaction fail by using the cAck code of STL_C_FAIL/STQ_C_FAIL. It can do so without asserting either **dOE_l** or **dWSel_h**.

See Section 6.4.7.

6.4.10.2 Fast Lock Mode (21064A only)

The **21064A** will probe external cache when executing both LDL_C/LDQ_C and STL_C/STQ_C transactions. Use of this mode is only possible when the external cache contains OE-mode RAMs and BIU_CTL [OE] is set. Setting BIU_CTL [FAST_LOCK] causes the **21064A** to enter the fast lock operating mode.

The **21064A** services LDL_L/LDQ_L instructions by performing an external cache 32-byte read if the external cache probe hits a valid external cache block. While accessing the data the **21064A** asserts **lockWE_h**. External logic should use the assertion of **lockWE_h** and **dataCEOE_h** to set the lock flag and load the address into the lock address register. If the probe does not hit a valid external cache block the **21064A** will start a LDL_L/LDQ_L transaction on the pin bus using **cReq_h[2:0]**.

Note

Timing of **lockWE_h** is the same as **dMapWE_h**.

In fast lock mode the **21064A** services STL_C/STQ_C instructions by performing an external cache probe while sampling **lockFlag_h**. If the probe hits a valid non-shared external cache block and **lockFlag_h** is asserted the **21064A** will perform the external cache write. While performing the write the **21064A** will assert **lockWE_h**. The external logic uses the assertion of **tagWE_h** and the deassertion of **dataCEOE_h** to clear the lock flag. If the probe does not hit a valid non-shared external cache block the **21064A** will start a STL_C/STQ_C transaction on the pin bus using **cReq_h[2:0]**.

Note

Timing of **lockWE_h** is the same as **tagCtlWE_h**. The timing requirement for **lockFlag_h** are the same as those of **tagAdr_h [33:18]**.

6.4.10.3 Noncached Loads

When ABOX_CTL [8] (NCACHE_NDISTURB) is clear external D-stream read transactions where external logic responds on **dRAck_h [2:0]** indicating *do not cache* cause the 21064/21064A to invalidate the Dcache line associated with the read address.

When ABOX_CTL [8] (NCACHE_NDISTURB) is set external D-stream read transactions where external logic responds on **dRAck_h [2:0]** indicating *do not cache* causes the 21064/21064A to leave the Dcache line associated with the read address undisturbed.

Also, when ABOX_CTL [8] (NCACHE_NDISTURB) is set external logic must respond with **dRAck_h [2:0]** indicating *do not cache* on only the external reads for which the 21064/21064A does not probe the external cache. The 21064 /21064A does not probe the external cache when:

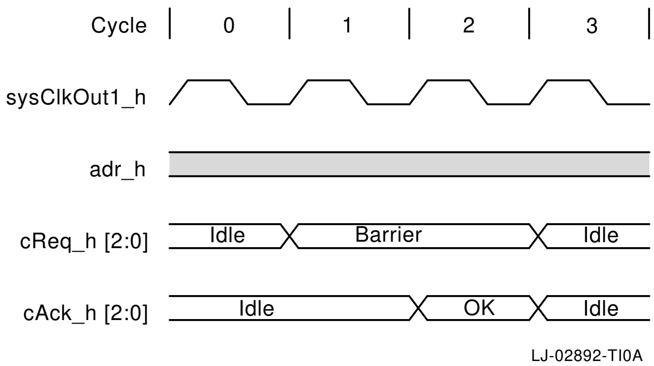
- Servicing LDL_L/LDQ_L instructions
- Accessing a quadrant of physical address space for which the external cache is disabled by setting BIU_CTL [BC_PA_DIS]
- The external cache is disabled completely by setting BIU_CTL [BC_ENA]

A response indicating *do not cache* to other types of external reads will cause the 21064/21064A's behavior to be UNDEFINED.

6.4.11 BARRIER

A BARRIER transaction appears on the external interface as a result of an MB instruction. The acknowledgment of the BARRIER transaction tells the 21064/21064A that all invalidates have been supplied to it, and that any external write buffers have been pushed out to the coherence point. Any errors detected during these operations can be reported to the 21064/21064A when the BARRIER transaction is acknowledged. Figure 6–14 shows the timing of the transaction.

Figure 6–14 BARRIER Transaction

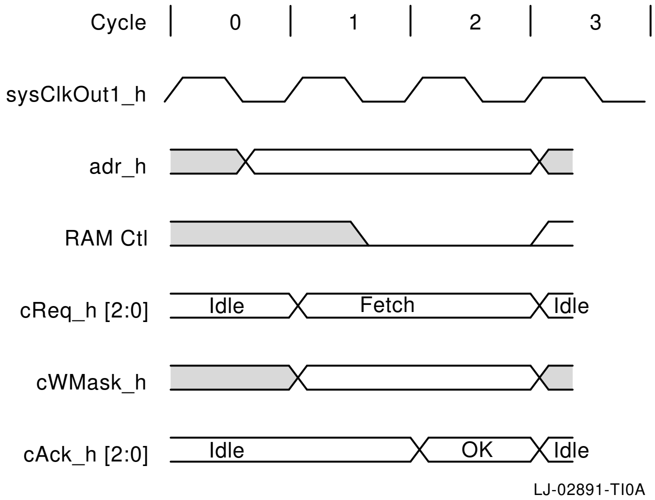


0. The **cReq_h** signals are always idle in the system clock immediately before the beginning of an external transaction.
1. The BARRIER transaction begins. The 21064/21064A places the command code for BARRIER onto the **cReq_h** outputs. The value placed on the address bus during BARRIER transactions is UNPREDICTABLE.
2. The external logic notices the BARRIER command, and because it has completed processing the command, it places an acknowledge code on the **cAck_h** inputs.
3. The 21064/21064A detects the acknowledge on **cAck_h**, and removes the command. The external logic removes the acknowledge code from **cAck_h**. The cycle is finished. This is the same as cycle 0.

6.4.12 FETCH

A FETCH transaction appears on the external interface as a result of a FETCH instruction. The transaction supplies an address to the external logic, which can choose to ignore it, or use it as a memory-to-cache prefetching hint. Figure 6–15 shows the timing of the transaction. The shaded areas indicate unpredictable levels.

Figure 6–15 FETCH Transaction



0. The **cReq_h** signals are always idle in the system clock cycle immediately before the beginning of an external transaction. The **adr_h** signals always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.

1. The FETCH transaction begins. The 21064/21064A has already placed the effective address of the FETCH on the address outputs. The 21064 /21064A places the command code for FETCH on the **cReq_h** outputs, and encodes the quadword granularity address bits (bits [4:3]) in the **cWMask_ h** field. The 21064/21064A clears the RAM control signals (**dataA_h [4:3]**, **dataCEOE_h [3:0]** and **tagCEOE_h**) no later than one CPU cycle after the system clock edge at which the transaction begins.

2. The external logic notices the FETCH command, and because it has completed processing the command, it places an acknowledge code on the **cAck_h** inputs.

3. The 21064/21064A detects the acknowledge on **cAck_h**, and removes the address and the command. The external logic removes the acknowledge code from **cAck_h**. The cycle is finished. This is the same as cycle 0.

6.4.13 FETCH_M

A FETCH_M transaction appears on the external interface as a result of a FETCH_M instruction. With the exception of the command code placed on **cReq_h**, the FETCH_M transaction is the same as the FETCH transaction. See Section 6.4.12.

6.5 Interface Operation

The 21064/21064A uses an external clock source to generate its internal CPU clock. The 21064/21064A will either use the input clock frequency or divide it by two. The clock frequency divisor, 1 or 2, is determined during reset.

Module level hardware that interfaces with the 21064/21064A need not run at CPU clock speed. The 21064/21064A divides its CPU clock frequency to generate systems clocks available for use by the external interface logic. The divisor value, 2 to 8 for the **21064** and 2 to 17 for the **21064A**, is determined during reset. System designers may choose to implement an off-chip secondary cache. The 21064/21064A hardware interface eases this task by allowing the use of commodity static RAMs. Because building high-speed logic is very difficult in low-end systems, the 21064/21064A controls the RAMs directly. The chip contains a programmable external cache interface, so that system designers can make external cache speed and configuration tradeoffs. Because no external cache policy decisions are made by the 21064/21064A, systems designers can choose their own cache coherence protocol.

6.5.1 Clocks

The 21064/21064A requires a differential input clock on **clkIn_h** and **clkIn_l**. During reset **testClkIn_h** and **testClkIn_l** indicate that the input clock will be 1x or 2x as listed here.

testClkIn_h	testClkIn_l	Function
L	L	Digital Reserved
L	H	Standard 2x input clock
H	L	Standard 2x input clock
H	H	1x input clock

The preferred (normal) input clock, 2x, is twice the internal clock frequency. The 21064/21064A divides this clock by two to generate the internal chip clock, called the CPU clock. The CPU clock is made available to the external interface on **cpuClkOut_h**.

The 21064/21064A will also accept a 1x input clock using that input to generate an internal clock with the same frequency as the input clock. This is usually a slower frequency clock used by test purposes.

Note

There is a significant cycle time penalty associated with using 1x clocks that the module designer should understand before choosing this option.

The CPU clock is divided by a programmable value to generate a system clock. The system clock is supplied to the external interface on **sysClkOut1_h** and **sysClkOut1_l**. The programmable divisor is:

- From 2 to 8 for the **21064**
- From 2 to 17 for the **21064A**

The system clock divisor, chosen by the system designer, is selected at chip reset for the:

- **21064** by **irq_h [4:3]**
- **21064A** by **irq_h [4:3]** and **sysClkDiv_h**

The system clock is delayed by a programmable number of CPU clock cycles between 0 and 3 to generate a delayed system clock, **sysClkOut2_h** and **sysClkOut2_l**. The system clock delay, again chosen by the system designer, is selected by **irq_h [4:3]** at chip reset.

The clock generator runs while the chip is held in reset, generating **cpuClkOut_h** and correctly timed and positioned **sysClkOut1** and **sysClkOut2**.

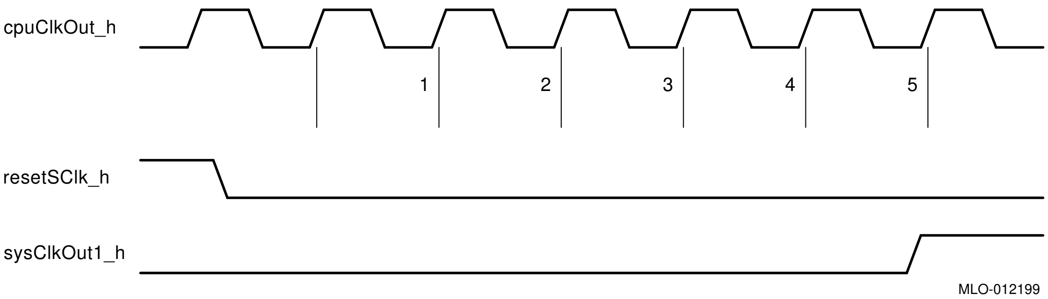
The output of the programmable divider is symmetric if the divisor is even, and asymmetric with **sysClkOut1_h** high (true) for one extra CPU cycle if the divisor is odd.

Almost all transactions on the external interface run synchronously to the CPU clock and phase aligned to the system clock, so the external interface appears to be running synchronously to the system clock (most setup and hold times are referenced to the system clock). The exceptions to this are the fast 21064/21064A controlled transactions on the external caches and the sampling of the **tagOk_h** and **tagOk_l** inputs, which are synchronous to the CPU clock, but independent of the system clock.

The **21064A** has an input, **resetSClk_h**, which is provided for test purposes and is used to force the system clock divider into a known state. At power-up **resetSClk_h** must be asserted for a minimum of 10 CPU cycles. While **resetSClk_h** is asserted the system clock signals, **sysClkOut1_h** and **sysClkOut1_l**, are deasserted as shown in Figure 6–16.

resetSClk_h should be deasserted synchronously to the internal CPU clock. The **21064A** samples **resetSClk_h** at the rising edge of **cpuClkOut_h**. The **21064A** will assert sysClkOut1_h on the fifth CPU clock cycle after detecting the deassertion of **resetSClk_h**.

Figure 6–16 21064A Delay of sysClkOut1_h



6.5.2 21064/21064A Initialization

The 21064/21064A contains a ring oscillator that is switched into service during power-up to provide an internal chip clock.

The dcOk_h Signal

The **dcOk_h** signal switches clock sources between the on-chip ring oscillator and the external clock oscillator. If **dcOk_h** is deasserted, then the on-chip ring oscillator feeds the clock generator, and the 21064/21064A is held in reset independent of the state of the **reset_l** signal. If **dcOk_h** is asserted, then the external clock oscillator feeds the clock generator. When **dcOk_h** is asserted the **vRef** input must be valid so that inputs can be sensed.

The **dcOk_h** signal is special because it does not require that **vRef** be stable to be sensed. It is important to emphasize the importance of driving **dcOk_h** low until the voltage on **vRef** has stabilized. Because chip testers can apply clocks and power to the chip at the same time, the chip tester can always drive

dcOk_h high, but the tester must drive **reset_l** low for a period longer than the minimum hold time of **vRef**.

The clock outputs follow the internal ring oscillator when the 21064/21064A is running off the oscillator (as they would when real clocks are applied). The frequency of the ring oscillator varies from chip to chip within a range of 10 MHz to 100 MHz. This corresponds to an internal CPU clock frequency range of 5 MHz to 50 MHz. When the **dcOk_h** signal is deasserted, the system clock divisor is forced to eight, and the **sysClkOut2_h**, **sysClkOut2_l** delay is forced to three.

CAUTION

When the **dcOk_h** signal is generated by an RC delay, there is no check to determine that the input clocks are really running. If power is applied to a board in manufacturing with a missing, defective, or mis-soldered clock oscillator, then the 21064/21064A will enter a possibly destructive high-current state. Furthermore, if a clock oscillator fails, then the 21064/21064A can also enter this state. Module designers must understand the frequency and duration of such events to decide if this is really a problem.

The reset_l Signal

The **reset_l** signal forces the CPU into a known state (see Section 5.6). The signal can be asynchronous, but must be asserted at least until the assertion of **dcOk_h** to guarantee that the 21064/21064A chip is properly reset.

In order to bring the chip out of internal reset at a deterministic time, the **reset_l** signal can be deasserted synchronously with respect to the system clock. See Chapter 7 for the setup and hold requirements of the **reset_l** signal when used in this way.

While in reset, the 21064/21064A reads sysClkOut and external bus configuration information off the **irq_h** signals; external logic should drive the configuration information onto the **irq_h** signals any time **reset_l** is asserted. In addition the **21064A** reads sysClkOut configuration information off the **sysClkDiv_h** signal; external logic should assert the sysClkOut information onto **sysClkDiv_h** at all times.

Power and Other Considerations

The 21064/21064A uses a 3.3 V power supply. This voltage supply must be stable before any input goes above 4 V.

The **irq_h [5]** bit is used to select 128-bit or 64-bit mode. If **irq_h [5]** is asserted then 128-bit mode is selected.

When the **tristate_l** signal is asserted, the chip is internally forced into the reset state.

See Chapter 7.

6.5.3 Internal Cache/Primary Cache Invalidate

External logic must be able to invalidate primary data cache blocks to maintain coherence. The 21064/21064A provides a mechanism to perform the necessary invalidates, but enforces no policy as to when invalidates are needed. Simple systems may choose to invalidate more or less blindly, and complex systems may choose to implement elaborate invalidate filters.

There are at least three situations where entries in the on-chip Dcache may need to be invalidated.

- When an external agent updates a block in memory (for example, an I/O device does a DMA transfer into memory), and that block has previously been loaded into the external cache, then the external cache block must be either invalidated or updated. If that external cache block has previously been loaded into the Dcache then that Dcache block must be invalidated.
- In the situation where a system is maintaining the Dcache as a subset of the external cache, and a Dcache miss results in an external cache block being replaced, and that external cache block has previously been loaded into Dcache, then an invalidate is needed.
- A third case can occur if the system is maintaining the Dcache as a subset of the external cache, and external system logic allocates blocks in the external cache during WRITE_BLOCK transactions. In this case, the Dcache must be invalidated when the WRITE_BLOCK command is issued.

6.5.3.1 21064 Primary Cache Invalidate

External logic invalidates an entry in the Dcache by asserting the **dInvReq_h** signal. The **21064** samples **dInvReq_h** at every system clock. When the **21064** detects **dInvReq_h** asserted, it invalidates the block in the Dcache whose index is on the **iAdr_h** signals.

The **21064** can accept an invalidate at every system clock.

The **dInvReq_h** input is synchronous, and external logic must guarantee setup and hold with respect to the system clock. The **iAdr_h** inputs are also synchronous, and external logic must guarantee setup and hold with respect to the system clock in any cycle in which **dInvReq_h** is asserted.

6.5.3.2 21064A Primary Cache Invalidate

External logic invalidates an entry in the Dcache by asserting the **dInvReq_h 0** and/or **dInvReq_h 1** signals. The **21064A** samples **dInvReq_h [1:0]** at every system clock. When either or both of **dInvReq_h [1:0]** are asserted, the **21064A** invalidates the blocks in the Dcache pointed to by the asserted **dInvReq_h** signals and the index on **iAdr_h [12:5]**.

Note

The IPR register bit ABOX_CTL [DOUBLE_INVAL] may be set which has the effect of asserting **dInvReq_h 1** whenever **dInvReq_h 0** is asserted.

The **21064A** can accept an invalidate at every system clock.

The **dInvReq_h [1:0]** inputs are synchronous, and external logic must guarantee setup and hold with respect to the system clock. **iAdr_h [12:5]** are also synchronous, and external logic must guarantee setup and hold with respect to the system clock in any cycle in which one of **dInvReq_h [1:0]** is asserted.

The **21064A** manages the 16K byte Dcache so that it never contains two different blocks that have equal values for PA [17:13]. This ensures that the Dcache never contains two blocks which map to the same Bcache block, for all supported Bcache sizes.

6.5.3.3 Backmap

Systems can maintain a backmap of the contents of the primary Dcache to improve the quality of their invalidate filtering. The 21064/21064A must maintain the backmap for external cache read hits, because external cache read hits are controlled totally by the 21064/21064A. External logic maintains the backmaps for external cycles (read misses, invalidates, and so on).

The backmap is only consulted by external logic, so that its format, and even its existence, is irrelevant to the 21064/21064A. Simple systems need not maintain a backmap, and need not connect the backmap write pulse to anything, and should generate extra invalidates.

21064 Support of Backmap

The **21064** drives a write pulse onto **dMapWE_h** whenever it fills the on-chip Dcache from the external cache. In 128-bit mode **dMapWE_h** asserts one CPU cycle into the second (last) data read cycle, and negates one CPU cycle from the end of that cycle. If read cycles are three CPU cycles long, then **dMapWE_h** is one CPU cycle long. See Section 6.5.6 for 64-bit mode operations.

Note

> This anomaly is caused by the backmap write overlapping a cycle whose length is specified by BC_RD_SPD. If the **21064** used the standard write pulse timing mechanism, and BC_WR_SPD were longer than BC_RD_SPD, the address would go away in the middle of the write cycle.

The backmap may be implemented by external logic that has the write enable input of the Dcache backmap RAM controlled by a two-input NOR gate. One side of the two-input NOR gate is driven by **dMapWE_h**, and the other input is driven by external logic.

21064A Support of Backmap

The **21064A** Dcache is viewed by external logic as a two-way set associative cache, therefore, the **21064A** external logic needs more information when implementing a backmap.

VA 13 is used as the MSB when Dcache is addressed and external logic may view this bit as selecting a cache set.

When the **21064A** fills the on-chip Dcache from the external cache, it asserts one of **dMapWE_h [1:0]**: **dMapWE_h 0** if VA 13 of the load instruction was zero and **dMapWE_h 1** if VA 13 was one. During external read transactions the **21064A** will place the value of VA 13 on both **cWMask_h 3** and **cWMask_h 4**.

In 128-bit mode **dMapWE_h 1** or **dMapWE_h 0** asserts one CPU cycle into the second (last) data read cycle, and negates one CPU cycle from the end of that cycle. If read cycles are three CPU cycles long, then **dMapWE_h [1:0]** is one CPU cycle long. See Section 6.5.6 for 64-bit mode operations.

Note

> This anomaly is caused by the backmap write overlapping a cycle whose length is specified by BC_RD_SPD. If the **21064A** used the standard write pulse timing mechanism, and BC_WR_SPD were longer

than BC_RD_SPD, the address would go away in the middle of the write cycle.

6.5.4 External Cache Control

The 21064/21064A's hardware interface allows system designers to build a second level external cache. There are few restrictions regarding the size, speed or coherence policy of the external cache. One restriction is that the external cache must be direct mapped. The 21064/21064A always views the external cache as having a tag for each 32-byte block (the same as the on-chip Icache and Dcache), although this need not be so. The external cache block size can be 32 bytes or larger.

The external cache size is selected by the BC_SIZE field in the BIU_CTL register.

- The **21064** supports an external cache of 128 KB to 16 MB. The cache size can increase by a factor of two starting at 128 KB.
- The **21064A** supports an external cache of 256 KB to 16 MB. The cache size can increase by a factor of two starting at 256 KB.

The external cache tag RAMs are located between the 21064/21064A's local address bus and its tag inputs. The external cache data RAMs are located between the CPU's local address bus and the CPU's local data bus. The 21064 /21064A reads the external cache tag RAMs to determine if it can complete a cycle without any interaction with external logic, and the 21064/21064A reads or writes the external cache data RAMs if this is the case.

A cycle requires no interaction with external logic if:

- It is a non-LDL_L/LDQ_L read hit to a valid block.
- It is an LDL_L/LDQ_L read on a **21064A** with fast lock mode enabled. See Section 6.4.10.2.
- A non-STL_C/STQ_C write hit to a valid block for which the tag control's S bit is clear.

All other cycles require interaction with external logic.

All cycles require interaction with external logic if:

- The external cache is disabled (the BC_ENA bit in the BIU_CTL IPR is cleared).

- The physical address of the reference is in a quadrant in memory that is not cached, that is, the appropriate bit in the BC_PA_DIS field in the BIU_CTL IPR is set for the quadrant of the reference.

All the 21064/21064A controlled cycles on the external cache have fixed timing, described in terms of the 21064/21064A's internal clock. The actual timing of the cycle is programmable by the BC_RD_SPD, BC_WR_SPD, and BC_WE_CTL fields in the BIU_CTL IPR, allowing for much flexibility in the choice of CPU clock frequencies and cache RAM speeds.

The external cache RAMs can be logically partitioned into three sections.

- tagAdr RAM
- tagCtl RAM
- Data RAM

Sections must not straddle physical RAM chips.

6.5.4.1 tagAdr RAM

The tagAdr RAM contains the high-order address bits associated with the external cache block, along with a parity bit. The contents of the tagAdr RAM are fed to the on-chip address comparator and parity checker then compared with **tagAdr_h [33:17]** and **tagAdrP_h**.

The 21064/21064A verifies that **tagAdrP_h** is an EVEN parity bit over **tagAdr_h** when it reads the tagAdr RAM. If the parity is wrong, the tag probe is forced to miss, and an external transaction is initiated. If machine checks are enabled (the MCHK_EN bit in the Abox_CTL IPR is set), the 21064/21064A traps to PALcode.

The number of bits of tagAdr_h that participate in the address compare and the parity check is controlled by the BC_SIZE field in the BIU_CTL IPR. The **tagAdr_h** signals go down to address bit 17, allowing an external cache as small as 128 KB.

The chip enable or output enable for the tagAdr RAM can be driven by a two-input NOR gate. One input of the gate is driven by **tagCEOE_h**, and the other input is driven by external logic. The 21064/21064A deasserts **tagCEOE_h** during reset, during external cache hold, and during any external cycle. This gives external logic control over these RAM input signals during these times. The OE bit in the BIU_CTL IPR determines if **tagCEOE_h** has chip enable timing or output enable timing.

6.5.4.2 tagCtl RAM

The tagCtl RAM contains control bits associated with the external cache block, along with a parity bit. The 21064/21064A reads the tagCtl RAM by way of the three tagCtl signals to determine the state of the block. The 21064/21064A writes the tagCtl RAM by the three tagCtl signals to make blocks dirty.

The 21064/21064A verifies that **tagCtlP_h** is an even parity bit over **tagCtlV_h**, **tagCtlS_h**, and **tagCtlD_h** when it reads the tagCtl RAM. If the parity is wrong, the tag probe results in a miss, and an external transaction is initiated. If machine checks are enabled (the MCHK_EN bit in the Abox_CTL IPR is set) the 21064/21064A traps to PALcode. The 21064/21064A computes even parity across the **tagCtlV_h**, **tagCtlS_h**, and **tagCtlD_h** bits, and drives the result onto the tagCtlP_h signal, when it writes the tagCtl RAM.

Table 6–13 shows the allowed combinations of the tagCtl RAM bits.

Note

The bias toward conditional write-through coherence is really only in name; the **tagCtlS_h** bit can be viewed simply as a write protect bit for a given external Bcache block. If the 21064/21064A gets a hit on a write probe and the **tagCtlS_h** bit is set, it will initiate a WRITE_BLOCK transaction. It is up to external hardware to re-probe the cache to determine whether **tagCtlS_h** is set and then impose whatever cache coherency policy is appropriate for the system. The **tagCtlS_h** bit is ignored during read cycles.

Table 6–13 Tag Control Encodings

tagCtlV_h	tagCtlS_h	tagCtlD_h	Meaning
L	X	X	Invalid
H	L	L	Valid, private
H	L	H	Valid, private, dirty
H	H	L	Valid, shared
H	H	H	Valid, shared, dirty

The 21064/21064A can satisfy a read probe if the tagCtl bits indicate the entry is valid (**tagCtlV_h** is asserted). The 21064/21064A can satisfy a write probe if the tagCtl bits indicate the entry is valid and not shared (**tagCtlV_h** is asserted, **tagCtlS_h** is deasserted).

The chip enable or output enable for the tagCtl RAM can be driven by a two-input NOR gate. One input of the gate is driven by **tagCEOE_h**, and the other input is driven by external logic. The 21064/21064A deasserts **tagCEOE_h** during reset, during external cache hold, and during any external cycle. The OE bit in the BIU_CTL IPR determines if **tagCEOE_h** has chip enable timing or output enable timing.

The write enable for the tagCtl RAM is normally driven by a two-input NOR gate. One input of the gate is driven by **tagCtlWE_h**, and the other input is driven by external logic. The 21064/21064A deasserts **tagCtlWE_h** during reset, during external cache hold, and during any external cycle. The BC_WE_CTL field in the BIU_CTL IPR determines the width of the write enable, and its position within the write cycle.

6.5.4.3 Data RAM

The data RAM contains the actual cache data, along with any ECC or parity bits.

The most significant bits of the data RAM address are driven by buffers from **adr_h [33:5]**. The least significant bit of the data RAM address can be driven by a two-input NOR gate. One of the inputs of the gate is driven by **dataA_h [4]**, and the other input is driven by external logic. The 21064/21064A deasserts **dataA_h [4]** during reset, during external cache hold, and during any external cycle.

The chip enables or output enables for the data RAM can be driven by a two-input NOR gate. One input of the gate is driven by **dataCEOE_h [3:0]**, and the other input is driven by external logic. The 21064/21064A deasserts **dataCEOE_h [3:0]** during reset, during external cache hold, and during external cycles. The OE bit in the BIU_CTL IPR determines whether **dataCEOE_h [3:0]** has chip enable timing or output enable timing.

The write enables for the data RAM can be driven by a two-input NOR gate. One input of the gate is driven by **dataWE_h [3:0]**, and the other input is driven by external logic. The 21064/21064A deasserts **dataWE_h [3:0]** during reset, during external cache hold, and during any external cycle. The BC_WE_CTL field in the BIU_CTL IPR determines the width of the write enable, and its position within the write cycle.

6.5.4.4 holdReq_h and holdAck_h External Cache Access

The external caches are normally controlled by the 21064/21064A. External logic may gain access to external cache RAMs by two methods: one uses **holdReq_h** and **holdAck_h** and the other uses **tagOk_l** and **tagOk_h**.

The simple method for external logic to access the external caches asserting the **holdReq_h** signal is described here.

When **holdReq_h** is asserted, the 21064/21064A does the following:

1. Finishes any external cache cycle that may be in progress
2. Tristates **adr_h**, **data_h**, **check_h**, **tagCtlV_h**, **tagCtlD_h**, **tagCtlS_h** and **tagCtlP_h**
3. Deasserts **tagCEOE_h**, **tagCtlWE_h**, **dataCEOE_h**, **dataWE_h** and **dataA_h [4:3]**
4. Asserts **holdAck_h**

The **cReq_h** and **cWMask_h** signals are not modified in any way. When external logic is finished with the external caches it deasserts **holdReq_h**. When the 21064/21064A detects the deassertion of **holdReq_h** it deasserts **holdAck_h** and re-enables its outputs.

The **holdReq_h** signal is synchronous, and external logic must guarantee setup and hold requirements with respect to the system clock. The **holdAck_h** signal is synchronous to the CPU clock but phase aligned to the system clock.

The 21064/21064A generates the **holdAck_h** signal in a way that allows it to be tied directly to the enable-inputs of external tristate drivers connecting to the bidirectional pin bus signals. The 21064/21064A turns off its tristate drivers on or before the system clock edge at which it asserts **holdAck_h**. The **21064** and **21064A** respectively turn on their tristate drivers two and four CPU cycles after the system clock edge at which they deassert **holdAck_h**.

The delay from **holdReq_h** assertion to **holdAck_h** assertion depends on the programming of the external interface, and on exactly how the system clock is aligned with a pending external cache cycle.

- In the best case, the external cache is idle or is just about to start a cycle, in which case **holdAck_h** asserts one system clock cycle after the system clock edge at which the 21064/21064A samples the **holdReq_h** assertion.
- In the worst case, the system clock edge at which the 21064/21064A samples the **holdReq_h** assertion occurs one CPU clock cycle into an external cache write probe that hits on a non shared line and requires two RAM data cycles to complete. In this case, **holdAck_h** asserts at the first

system clock edge that is at least
((BC_RD_SPD + 1) - 1) + 2*(BC_WR_SPD + 1) + 1 CPU cycles after the system clock edge at which the 21064/21064A sampled the **holdReq_h** assertion.

holdAck_h deasserts in the system clock cycle immediately following the system clock edge at which the 21064/21064A samples the deassertion of **holdReq_h**.

A **holdReq_h/holdAck_h** sequence can happen at any time, even in the middle of an external transaction. The assertion of **holdReq_h** prevents the 21064/21064A's BIU sequencer from starting new CPU requests. However, the BIU sequencer initiates the external transaction by driving the **cReq_h** signals to the appropriate value (despite **holdReq_h's** assertion) if two things are true:

- The BIU sequencer has already started an external cache tag probe when **holdReq_h** is asserted.
- The result of the tag probe requires an external transaction to complete the CPU's request.

holdAck_h asserts at the next system clock edge after the tag probe completes.

Note

The **21064** waits two CPU cycles and the **21064A** waits four CPU cycles before turning on their tristate drivers after they deassert **holdAck_h**. External logic must be careful about when external logic continues with an interrupted external transaction at the end of a **holdReq_h /holdAck_h** sequence.

6.5.4.5 tagOk_h and tagOk_l External Cache Access

Although using the **holdReq_h** and **holdAck_h** lines is the simplest method for external logic to gain access to the Bcache, the fastest way for external logic to gain access is to use the **tagOk_h** and **tagOk_l** signals. These signals allow external logic to stall a 21064/21064A cycle on the external cache RAMs at the last possible instant.

All tradeoffs surrounding these signals have been made in favor of high-performance systems, making them next to impossible to use in low-end systems.

The **tagOk_h** and **tagOk_l** signals are synchronous and **21064** external logic must guarantee setup and hold requirements with respect to the CPU clock. This implies very fast logic, since the **21064** CPU clock can run at 150, 166 or 200 MHz. See Section 7.4.7 for **21064** synchronization information and Section 7.4.8 for similar information about the **21064A**.

The **tagOk** signals are normally asserted (that is, **tagOk_h** is high and **tagOk_l** is low). When deasserted the **tagOk** signals stall a sequencer in the **21064** bus interface unit. The **21064** does not tristate the buses that run between the **21064** and the external cache RAMs. External logic must supply the necessary multiplexing functions in the address and data path.

If a **tagOk** signal is asserted at a CPU clock edge, the external logic is making a guarantee that:

- The tagCtl and tagAdr RAMs were owned by the 21064/21064A in the previous BC_RD_SPD+1 CPU cycles.
- The tagCtl RAMs will be owned by the 21064/21064A in the next BC_WR_SPD+1 cycles.
- The data RAMs were owned by the 21064/21064A in the previous BC_RD_SPD+1 cycles.
- The data RAMs will be owned by the 21064/21064A in the next BC_RD_SPD+1 CPU cycles or in the next 2*(BC_WR_SPD+1) CPU cycles, whichever is longer.

The bus interface unit samples tagOk signals in the last two cycles of each tag probe, and only proceeds if tagOk has been asserted in both of these cycles. If the 21064/21064A samples tagOk as deasserted in either of the last two CPU cycles of a tag probe, then it stalls until it samples tagOk true in consecutive cycles. At that time, all of these assertions are true, which means, in particular, that any address the 21064/21064A has been holding on the address bus throughout this time has made it through the external cache RAMs. The 21064/21064A then proceeds normally.

6.5.4.6 External RAM Timing

Many external static RAMs support two access times—a "long" access time from address transition to data out, and a "short" access time from a particular address pin transition to data out. In order to fill a primary Icache block the 21064/21064A performs two (128-bit data bus mode) or four (64-bit data bus mode) external RAM cycles. When using RAMs which support dual access speeds, the BIU_CTL register BC_RD_SPD field controls the "long" access and the BC_BURST_SPD and BC_BURST_ALL fields control the "short" access time.

6.5.5 Bus Cycle Control

The 21064/21064A requests an external cycle when it determines that the cycle it wants to run requires interaction with external logic.

6.5.5.1 Cycle Request

An external cycle begins when the 21064/21064A puts a cycle type onto the **cReq_h** outputs. These outputs change simultaneously with the rising edge of **sysClkOut1_h**. Some cycles put an address on the **adr_h** outputs, and additional information (low-order address bits, I/D stream indication, write masks) on the **cWMask_h** outputs.

The cycle types are shown in Table 6–14.

Table 6–14 Cycle Types

cReq_h [2]	cReq_h [1]	cReq_h [0]	Type
L	L	L	IDLE
L	L	H	BARRIER
L	H	L	FETCH
L	H	H	FETCH_M
H	L	L	READ_BLOCK
H	L	H	WRITE_BLOCK
H	H	L	LDL_L/LDQ_L
H	H	H	STL_C/STQ_C

The MB instruction generates the BARRIER cycle. Normally, the module acknowledges it. Modules that have write buffers between the 21064 /21064A and the memory system must drain these buffers before the cycle is acknowledged to guarantee that machine checks caused by transport and/or memory system errors get posted on the correct side of the MB instruction.

The FETCH and FETCH_M instructions respectively generate FETCH and FETCH_M cycles. The address bus contains the effective address generated by the FETCH or FETCH_M instruction. These addresses can be used by module level prefetching logic to preload one or more cache blocks into the external cache. Simpler systems can acknowledge the cycles without prefetching data.

The READ_BLOCK cycle is generated on read misses. External logic reads the addressed block from memory and supplies it, 128 bits at a time, to the 21064/21064A on the data bus. External logic can also write the data into the external cache, after perhaps writing a victim.

The WRITE_BLOCK cycle is generated on write misses, and on writes to shared blocks. External logic pulls the write data, 128 bits at a time, from the 21064/21064A with the data bus, and writes the valid longwords to memory. External logic can also write the data into the external cache, after perhaps writing a victim.

The interlocked load instructions generate the LDL_L/LDQ_L cycle. The cycle works in the same way as a READ_BLOCK, although the external cache has not been probed (so the external logic needs to check the external cache for hits), and the address must be latched into a locked-address register.

The conditional store instructions generate the STL_C/STQ_C cycle. The cycle works in the same way as a WRITE_BLOCK, although the external cache has not been probed (so that external logic needs to check for hits), and the cycle can be acknowledged with a failure status.

6.5.5.2 Cycle Write Masks

On WRITE_BLOCK and STL_C/STQ_C cycles the cWMask_h signals supply longword write masks to the external logic, indicating which longwords in the 32-byte block are valid. A **cWMask_h** bit is true if the longword is valid. **cWMask_h** bit [0] is associated with longword 0 in the 32-byte block, **cWMask_h** bit [1] is associated with longword 1 in the 32-byte block, and so on.

WRITE_BLOCK commands can have any combination of mask bits set. STL_C /STQ_C cycles can only have combinations that correspond to a single longword or quadword.

See Table 6–15 for correspondence between **cWMask_h [7:0]** and **adr_h [4:3]**.

Table 6–15 FETCH/FETCH_M Cycle Write Mask Addresses

cWMask_h [7:0]$_2$	adr_h [4:3]$_2$
00000011	00
00001100	01
00110000	10
11000000	11

On READ_BLOCK and LDL_L/LDQ_L cycles the **cWMask_h** signals have additional information about the transaction on them.

- **cWMask_h [1:0]** signals contain transaction address bits [4:3] (points to quadword).

- **cWMask_h 5** contains address bit 2 (points to the LW).
- **cWMask_h 2** is asserted for a D-stream reference, and deasserted for an I-stream reference.
- **21064A only—cWMask_h [4:3]** both indicate VA 13: one or zero.
- **21064A only—cWMask_h 6** indicates the data size (1 for LW, 0 for QW) during LDL_L/LDQ_L and D-stream READ_BLOCK transactions.

6.5.5.3 Cycle Acknowledgment

A cycle remains on the external interface until external logic acknowledges it by placing an acknowledgment type on the **cAck_h** signals. The **cAck_h** inputs are synchronous, and external logic must guarantee setup and hold requirements with respect to the system clock.

Table 6–16 shows acknowledgment types.

Table 6–16 Acknowledgment Types

cAck_h 2	cAck_h 1	cAck_h 0	Type
L	L	L	IDLE
L	L	H	HARD_ERROR
L	H	L	SOFT_ERROR
L	H	H	STL_C_FAIL/STQ_C_FAIL
H	L	L	OK

The 21064/21064A behavior in response to **cAck_h** encodings, other than those listed, is UNDEFINED.

The HARD_ERROR type indicates that the cycle has failed in some catastrophic manner. The 21064/21064A latches sufficient state to determine the cause of the error, and initiates a machine check.

The SOFT_ERROR type indicates that a failure occurred during the cycle, but the failure was corrected. The 21064/21064A latches sufficient state to determine the cause of the error, and initiates a corrected error interrupt.

The STL_C_FAIL/STQ_C_FAIL type indicates that a STL_C/STQ_C cycle has failed. The result is UNDEFINED if this type is used on anything but an STL_C/STQ_C cycle. Only STL_C/STQ_C transactions that are terminated with STL_C_FAIL/STQ_C_FAIL result in a zero being written to the destination register of the associated STL_C/STQ_C instruction.

The OK type indicates success.

6.5.5.4 Read Data Acknowledgment

The **dRAck_h** signals inform the 21064/21064A if:

- Read data is valid on the data bus.
- Data should be cached.
- ECC or parity checking should be attempted.

The **dRAck_h** inputs are synchronous, and external logic must guarantee setup and hold requirements with respect to the system clock. If **dRAck_h** is sampled IDLE at a system clock, then the data bus is ignored. If **dRAck_h** is sampled non-IDLE at a system clock, then the data bus is latched at that system clock, and external logic must guarantee that the data meet setup and hold with respect to the system clock.

Table 6–17 shows acknowledgment types.

Table 6–17 Read Data Acknowledgment Types

dRAck_h 2	dRAck_h 1	dRAck_h 0	Type
L	L	L	IDLE
H	L	L	OK_NCACHE_NCHK
H	L	H	OK_NCACHE
H	H	L	OK_NCHK
H	H	H	OK

The 21064/21064A behavior in response to **dRAck_h** encoding, other than those listed in Table 6–17 is UNDEFINED.

READ_BLOCK and LDL_L/LDQ_L transactions can be terminated with HARD_ERROR status before any expected **dRAck_h** cycles are received. In this event the contents of the entire internal cache block, including its tag and valid bit, are UNPREDICTABLE. A machine check is posted if so enabled.

The 21064/21064A can use D-stream primary cache fill data as soon as it is received, including data received in the first half of a READ_BLOCK transaction that is later terminated with HARD_ERROR. The 21064/21064A does not use any I-stream primary cache fill data until it successfully receives the entire cache block.

The 21064/21064A does not change its interpretation of **dRAck_h [1:0]** based on **cAck_h** if all expected dRAck signals are received. Therefore, external logic must avoid caching and/or ECC/parity checking data which is known to be invalid.

The 21064/21064A behavior is UNDEFINED if **dRAck_h** is asserted in a non-read cycle.

The 21064/21064A checks **dRAck_h 0** (the bit that determines whether the block is ECC/parity checked) when sampling each half of the 32-byte block. It is legal, but probably not useful, to check only one half of the block.

External logic should supply the same value on **dRack_h [1]** (the bit that determines whether the block should be internally cached) during both halves of the fill sequence. External logic should never signal I-stream reads to be noncached.

The first non-IDLE sample of **dRAck_h** tells the 21064/21064A to sample data bytes 15:0, and the second non-IDLE sample of **dRAck_h** tells the 21064 /21064A to sample data bytes 31:16.

Note

External logic can assert the second **dRAck_h** and **cAck_h** during the same system clock cycle, but systems will suffer from bus contention. The 21064/21064A can launch an external cache access on the same clock edge as it samples cAck. System logic will therefore be driving data to the CPU when the CPU asserts **dataCEOE_h**.

6.5.5.5 Support for Wrapped Read Transactions

The 21064/21064A supports two modes for returning read data, depending on the state of the SYS_WRAP bit in BIU_CTL IPR. If SYS_WRAP is clear, read data must be returned in order from lowest address to highest address. The first non-IDLE sample of **dRAck_h** tells the 21064/21064A to sample data bytes [15:0]. The second non-IDLE sample of **dRAck_h** tells the 21064 /21064A to sample data bytes [31:16]. If SYS_WRAP is set, external logic must return the 128-bit data chunk containing the requested quadword first. If **cWMask_h 1** is set, meaning address bit 4 was set in the original request, the first non-IDLE sample of **dRAck_h** tells the 21064/21064A to sample data bytes [31:16] and the second non-IDLE sample of **dRAck_h** tells the 21064 /21064A to sample data bytes [15:0].

The read data for 128-bit mode would be returned as shown here.

Requested Address (HEX)	Return Order	
	SYS_WRAP=0	SYS_WRAP=1
0	0, 10	0, 10
1 0	0, 10	10, 0

The read data for 64-bit mode would be returned as shown here.

Requested Address (HEX)	Return Order	
	SYS_WRAP=0	SYS_WRAP=1
0	0, 8, 10, 18	0, 8, 10, 18
8	0, 8, 10, 18	8, 0, 18, 10
10	0, 8, 10, 18	10, 18, 0, 8
18	0, 8, 10, 18	18, 10, 8, 0

When the SYS_WRAP bit of the BIU_CTL IPR is clear, external logic can terminate external noncached D-stream reads that request data from bytes [15:0] by asserting **cAck_h** after or during the first **dRack_h** assertion. If the noncached read requests data from bytes [31:16], two **dRAck_h** assertions are always required.

When SYS_WRAP is set, external logic can always terminate external noncached reads by asserting cAck_h after or during the first assertion of **dRAck_h**.

6.5.5.6 Enabling the Data Bus

The **dOE_l** input tells the 21064/21064A if it should drive the data bus. Because it is a synchronous input, external logic must guarantee setup and hold with respect to the system clock.

- If **dOE_l** is sampled true at the end of a system clock cycle, then the 21064/21064A drives the data bus at the beginning of the next system clock cycle, as long as it has a WRITE_BLOCK or STL_C/STQ_C request pending. (The request can already be on the cReq signals, or it can appear on the cReq signals at the same system clock edge as the data appears.)
- If **dOE_l** is sampled false at the end of a system clock cycle, then the 21064/21064A tristates the data bus at the beginning of the next system clock cycle. For example, if **dOE_l** was sampled false at the end of cycle 1, then it would be tristated during cycle 2.

The transaction type is factored into the enable so that systems can leave **dOE_l** asserted unless external logic needs to drive the data bus within the context of a WRITE_BLOCK or STL_C/STQ_C transaction.

6.5.5.7 Selecting Write Data

The **dWSel_h [1]** input tells the 21064/21064A which half of the 32-byte block of write data should be driven onto the data bus (**dOE_l** permitting). This is a synchronous input, so external logic must guarantee setup and hold with respect to the system clock.

- If **dWSel_h [1]** is sampled false at the end of a system clock cycle, then bytes [15:0] are driven onto the data bus in the next system clock cycle.
- If **dWSel_h [1]** is sampled true at the end of a system clock cycle, then bytes [31:16] are driven onto the data bus in the next system clock cycle. Once **dWSel_h [1]** has been sampled true bytes [15:0] are lost; there is no backing up.

In the 21064/21064A, **dWSel_h [1]** should only be asserted after external logic has sampled bytes [15:0] within the WRITE_BLOCK or STL_C/STQ_C transaction, which means that this signal should never be asserted while **cReq_h** is idle.

6.5.6 64-Bit Mode

The 21064/21064A can be configured at reset to use a 64-bit wide external data bus, in which case **data_h [127:64]** and **check_h [27:14]** are not used. These pins are internally pulled to Vss, so no external connections to these signals are required.

The **dataA_h [3]** signal is used as an additional address line for the external cache data RAMs. Like the **dataA_h [4]** signal, it can drive a two-input NOR gate, with the other input being driven by external logic. The 21064/21064A deasserts **dataA_h [3]** during reset, during external cache hold, and during any external cycle.

The **dWSel_h [0]** signal should be used by external logic along with the **dWSel_h [1]** pin to select which quadword of a 32-byte block is driven onto **data_h [63:0]** during each system clock cycle of an external WRITE_BLOCK or STL_C/STQ_C transaction. The relationship between **dWSel_h [1:0]** and the selected bytes of the 32-block block is shown in Table 6–18.

Table 6–18 dWSel_h Byte Selection

dWSel_h [1:0]	Selected Bytes
00	[07:00]
01	[15:08]
10	[23:16]
11	[31:24]

External logic must select quadwords in increasing order within the 32-byte block, but is free to skip over any quadword which does not have corresponding longword mask bits TRUE in **cWMask_h [7:0]**.

In 64-bit mode, **dWsel_h [1:0]** should only be asserted within the context of a WRITE_BLOCK or STL_C/STQ_C transaction.

Systems should ignore **dataCEOE_h [3:2]** and **dataWE_h [3:2]**.

External cache read hit transactions are extended to consist of four cache read cycles in 64-bit mode.

Cache Read Cycle	Type
First	Tag probe and data read
Second through fourth	Data reads

The 21064/21064A bus interface optimizes the external cache read hit transaction by wrapping cache read cycles around the quadword that the 21064/21064A originally requested. The **dMapWE_h** signal asserts one CPU cycle into the second cache read cycle and remains asserted until one CPU cycle before the end of the fourth cache read cycle.

External cache write hit transactions consist of one cache tag probe cycle that is (BC_RD_SPD + 1) CPU cycles long, followed by one, two, three or four external cache write cycles that are each (BC_WR_SPD + 1) cycles long. The 21064/21064A bus interface uses the minimum number of cache write cycles required to write the necessary longwords within the 32-byte block.

The maximum delay from **holdReq_h** assertion to **holdAck_h** assertion in 64-bit mode is longer than in 128-bit mode. In the worst case the system clock at which the 21064/21064A samples the **holdReq_h** assertion occurs one CPU cycle into an external cache probe. In this case, the 21064/21064A may not assert holdAck_h until the first system clock edge that is at least: ((BC_RD_SPD+ 1)−1) +4∗ (BC_WR_SPD+1) +1, or ((BC_RD_SPD+1)−1) + 3∗ (BC_RD_SPD+1) +1, whichever is longer.

The guarantee external logic must make for availability of the external cache data RAMs when asserting tagOk is different for 64-bit mode than for 128-bit mode. In 64-bit mode, if tagOk is true at a CPU clock edge, the external logic is guaranteeing that the:

- tagCtl and tagAdr RAMs were owned by the 21064/21064A in the previous BC_RD_SPD +1 CPU cycles.
- tagCtl RAMs will be owned by the 21064/21064A in the next BC_WR_SPD+1 cycles.
- Data RAMs were owned by the 21064/21064A in the previous BC_RD_SPD+1 cycles.
- Data RAMs will be owned by the 21064/21064A in the next 3 ∗ (BC_RD_SPD+ 1) CPU cycles or in the next 4 ∗ (BC_WR_SPD+ 1) CPU cycles, whichever is longer.

Noncached D-stream read transactions can be terminated early by asserting **cAck_h** during or after the system clock cycle in which the 21064/21064A samples the requested quadword.

Each quadword is parity/ECC checked based on the **dRAck_h** code supplied with that quadword. The **dRAck_h** code returned with the first quadword of data determines whether the block is internally cached.

6.5.7 Instruction Cache Initialization/Serial ROM Interface

The 21064/21064A implements Icache initialization modes to support normal use along with chip and PCB level testing.

The **21064** uses the value on **icMode_h [1:0]** to determine which mode is used after the **21064** is reset, as shown in Table 6–19. Unlike the value placed on **irq_h [5:0]** during reset, the value placed on **icMode_h [1:0]** must be retained after **reset_l** is deasserted.

Table 6–19 21064 Icache Test Modes

icMode_h [1]	icMode_h [0]	Mode
L	L	Serial ROM
L	H	Disabled
H	L	Digital reserved
H	H	Digital reserved

The **21064A** uses the value on **icMode_h [2:0]** to determine which mode is used after the **21064A** is reset, as shown in Table 6–20. The value placed on **icMode_h [2:0]** must also be retained after **reset_l** is deasserted.

Table 6–20 21064A Icache Test Modes

icMode_h [2:0]	Mode
L L L	Serial ROM
L L H	Disabled
Six other combinations	Digital reserved

If the value on **icMode_h** selects Serial ROM Mode, the 21064/21064A loads the contents of its internal Icache from an external serial ROM before executing its first instruction. The serial ROM may contain enough code to complete the configuration of the external interface (for example, setting the timing on the external cache RAMs) and diagnose the path between the CPU chip and the real ROM. The 21064/21064A is in PALmode following the deassertion of **reset_l**—this gives the code loaded into the Icache access to all of the visible state within the chip.

Three signals are used to interface to the serial ROM.

- The **sRomOE_l** output signal supplies the output enable to the ROM, serving both as an output enable and as a reset.
- The **sRomClk_h** output signal supplies the clock to the ROM that causes it to advance to the next bit.
- The **sRomD_h** input signal allows the 21064/21064A to read the ROM data. In this mode the instruction cache is written at a rate of one bit each:
  - 126 CPU cycles for the **21064**
  - 254 CPU cycles for the **21064A**

Using the **icMode_h** signals, the serial ROM interface can be disabled altogether. In this case, since the Icache valid bits are cleared by reset, the first instruction fetch will miss the Icache.

In the 21064/21064A, all Icache bits are loaded from the serial ROM interface. The Icache blocks are loaded in sequential order starting with block zero and ending with block 256.

The order in which bits within each block are serially loaded is shown in Figure 6–17 and listed, with bits per field, in Table 6–21.

Figure 6–17 Icache Load Order

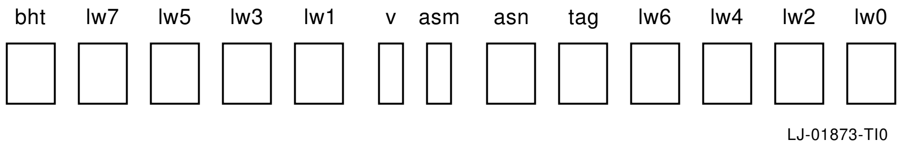


Table 6–21 Icache Field Size

Field	Bits	Field	Bits
bht	8	asn	6
v	1	tag	21
asm	1	lw0—lw7	32 (per field)

Note

In Figure 6–17, high-order bits are on the left within each field. The serial chain starts with bht and shifts to the right.

The valid and asm bits in each cache block must be set. The tag field must also be written with zero. The value written into the branch history table (bht) and address space number (asn) fields are "don't cares."

6.5.7.1 Implementing the Serial Line Interface

Once the data in the serial ROM has been loaded into the Icache, the three special signals become simple parallel I/O pins that can be used to drive a diagnostic terminal. When the serial ROM is not being read, the **sRomOE_l** output signal is false. This means that the **sRomOE_l** pin can be wired to the active high enable of an RS422 receiver driving onto **sRomD_h** and to the active high enable of an RS422 driver driving from **sRomClk_h**. The CPU allows **sRomD_h** to be read and **sRomClk_h** to be written by PALcode; this is sufficient hardware support to implement a software driven serial interface.

6.5.8 Interrupts

The **irq_h [5:0]** inputs generate external interrupts to the 21064/21064A. The six interrupts are identical, they can be asynchronous, they are level sensitive, and they can be individually masked by PALcode.

The use of each of these interrupt requests and the priority and vector assigned to them is controlled by PALcode and is completely controlled by the system designer.

To aid pattern-driven chip testers, the **irq_h** signals can be driven synchronously with respect to the system clock. See Section 7.4.9.1 for the setup and hold requirements of the **irq_h** signals with respect to the system clock for this case.

6.5.9 External Bus Interface

The use and operation of the address, data, and parity/ecc lines is described in this section.

6.5.9.1 Address Bus—adr_h [33:5]

The 21064/21064A implements 34 physical address bits, enough to address 16 Gbytes of storage. The bidirectional, tristate **adr_h [33:5]** signals provide a path for addresses to flow between the 21064/21064A and the rest of the system. These address bits provide granularity down to 32-byte internal cache blocks. In systems which implement an external cache, these signals are generally connected by buffers to the address inputs of the cache RAMs. For the 21064/21064A-controlled reads and writes of the external cache, further address resolution is provided by **dataA_h [4]** and **dataWe_h [3:0]**.

The **adr_h [33:5]** signals are also connected to external logic responding to the 21064/21064A generated requests which are not completed using the external cache. For these transactions, longword address granularity is provided by the **cWMask_h [7:0]** pins.

The address bus is normally driven by the 21064/21064A. The 21064/21064A stops driving the address bus during reset and during external cache hold. In the external cache hold state the address bus acts like an input.

The **21064** output **tagEq_l**[1] is the result of an equality compare between **adr_h** and **tagAdr_h**. Only bits that are part of the cache tag, as specified by the BC_SIZE field of the BIU_CTL IPR, participate in the compare. The **tagEq_l** signal is asserted during external cache hold only if the result of the tag comparison is true, and the parity calculated across the appropriate bits of

[1] The **21064A** does not implement the signal line **tagEq_l**.

tagAdr_h matches the value on **tagAdrP_h**. Even parity is used. **tagEq_l** is deasserted when the address bus is not in the external cache hold state.

6.5.9.2 Data Bus—data_h [127:0]

The bidirectional, tristate **data_h** signals provide a path for data to flow between the 21064/21064A and the rest of the system. In systems with an external cache, these pins also connect directly to the I/O pins of the external cache data RAMs.

The data bus is driven by the 21064/21064A when it is controlling a write cycle on the external caches, or when some type of write cycle has been presented to the external interface and external logic has enabled the data bus drivers (by **dOE_l**).

6.5.9.3 Parity/ECC Bus—check_h [27:0]

The 21064/21064A provides longword ECC and longword parity protection for data transferred on the data bus. The **21064A** provides byte parity protection also.

BIU_CTL [ECC] determines if the **21064** is in ECC mode or in parity mode.

BIU_CTL [BYTE_PARITY] and BIU_CTL [ECC] determine if the **21064A** is in ECC, LW parity, or byte parity mode as shown in Table 6–22.

Table 6–22 21064A Data Protection Mode Selection

BIU_CTL [BYTE_ PARITY]	BIU_CTL [ECC]	Protection Mode
0	0	LW parity
X	1	ECC
1	0	Byte parity

The bidirectional, tristate **check_h** signals provide a path for parity or ECC bits to flow between the 21064/21064A and the rest of the system. In systems with an external cache, these pins also connect directly to the I/O pins of the external cache data RAMs.

ECC Mode

If the 21064/21064A is in ECC mode, then the **check_h** signals carry seven check bits for each longword on the data bus.

- Bits **check_h [6:0]** are the check bits for **data_h [31:0]**.
- Bits **check_h [13:7]** are the check bits for **data_h [63:32]**.
- Bits **check_h [20:14]** are the check bits for **data_h [95:64]**.
- Bits **check_h [27:21]** are the check bits for **data_h [127:96]**.

Figure 6–18 shows the ECC code used. Data bits [31:0] are shown across the top of the table.

Figure 6–18 ECC Code

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
c6	XOR	X	X	X	X	X	X	X	X																	X	X	X	X	X	X	X	X
c5	XOR	X	X	X	X	X	X	X	X									X	X	X	X	X	X	X	X								
c4	XOR	X	X							X	X	X	X	X	X			X	X							X	X	X	X	X	X		
c3	XNOR			X	X	X				X	X	X				X	X			X	X	X				X	X	X				X	X
c2	XNOR	X		X			X	X		X			X	X			X	X		X			X	X		X			X	X			X
c1	XOR				X		X		X		X		X		X	X	X				X		X		X		X		X		X	X	X
c0	XOR	X		X	X		X					X		X	X	X			X			X		X	X	X	X		X				X

LJ-01874-TI0

By arranging the data and check bits correctly, it is possible to arrange that any number of errors restricted to a 4-bit group can be detected. One such arrangement is as follows:

d 00,	d 01,	d 03,	d 25
d 02,	d 04,	d 06,	c 06
d 05,	d 07,	d 12,	c 03
d 08,	d 09,	d 11,	d 14
d 10,	d 13,	d 15,	d 19
d 16,	d 17,	d 22,	d 28
d 18,	d 23,	d 30,	c 05
d 20,	d 27,	c 04,	c 00
d 21,	d 26,	c 02,	c 01
d 24,	d 29,	d 31	

LW Parity Mode

If the 21064/21064A is in longword parity mode, then four of the **check_h** signals carry even parity for each longword on the data bus as indicated in Table 6–23. The remaining **check_h** signals are unused.

Table 6–23 LW Parity Check Bits

Parity Bit	Parity Field	Parity Bit	Parity Field
check_h 0	**data_h [31:0]**	**check_h 7**	**data_h [63:32]**
check_h 14	**data_h [95:64]**	**check_h 21**	**data_h [127:96]**

Byte Parity Mode—21064A only

When the **21064A** is in byte parity mode, **check_h** pins carry even parity across their associated **data_h** pins as shown in Table 6–24.

Table 6–24 21064A Byte Parity check_h Bits

check_h	data_h	check_h	data_h
check_h 0	**data_h [7:0]**	**check_h 1**	**data_h [15:8]**
check_h 2	**data_h [23:16]**	**check_h 3**	**data_h [31:24]**
check_h 7	**data_h [39:32]**	**check_h 8**	**data_h [47:40]**
check_h 9	**data_h [55:48]**	**check_h 10**	**data_h [63:56]**
check_h 14	**data_h [71:64]**	**check_h 15**	**data_h [79:72]**
check_h 16	**data_h [87:80]**	**check_h 17**	**data_h [95:88]**
check_h 21	**data_h [103:96]**	**check_h 22**	**data_h [111:104]**
check_h 23	**data_h [119:112]**	**check_h 24**	**data_h [127:120]**

6.5.10 Performance Monitoring

The **perf_cnt_h [1:0]** signals provide a means of giving the 21064/21064A's internal performance monitoring hardware access to off-chip events. These signals are system clock synchronous inputs which can be selected by the ICCSR IPR to be inputs to the performance counters inside the 21064/21064A chip. If in a given system clock cycle a **perf_cnt_h** signal is sampled high (true), and the signal is selected as the source of its respective performance counter, then the counter will increment.

6.5.11 Various Other Signals

Tristate (tristate_l)

The **tristate_l** signal, if asserted, causes the 21064/21064A to float all of its output and bidirectional signals with the exception of **cpuClkOut_h**. When **tristate_l** is asserted, the 21064/21064A is forced into the reset state.

Continuity (cont_l)

The **cont_l** signal, if asserted, causes the 21064/21064A to connect all of its signals to Vss, with the exception of **clkIn_h**, **clkIn_l**, **testClkIn_h**, **testClkIn_l**, **cpuClkOut_h**, **vRef** and **cont_l**.

vRef

The **vRef** input supplies a reference voltage to the input sense circuits. If external logic ties this to Vss + 1.4V then all inputs sense TTL levels.

eclOut_h

Output mode selection; this pin should be tied to Vss.

6.6 Hardware Error Handling

For the following discussion the term "single-bit error" refers to a single corrupted bit in a single longword or its associated 7-bit check field, and the term "double-bit error" refers to two or more corrupted bits in a single longword or its associated 7-bit check field.

When in ECC mode, the 21064/21064A generates longword ECC on writes, and checks ECC on reads. The 21064/21064A contains hardware which can correct all single-bit errors which are confined to a single quadword with each 32-byte cache fill block.

Because the 21064/21064A requires complete instruction cache blocks from which to execute instructions, Icache fill blocks containing more than one bad quadword are not correctable in hardware, even if no longword within the block contains more than a single bad bit.

For D-stream ECC errors the correction hardware corrects errors in the quadword requested by the load instruction which originally invoked the fill operation. The correction hardware sends the corrected quadword to the CPU to satisfy the original request, invalidates the Dcache, and ensures that no other load instructions except the one which invoked the fill are allowed to use the data from the corrupted Dcache fill transaction. This means that the 21064/21064A can recover in hardware from all true single bit D-stream ECC errors.

The 21064/21064A hardware can recover from the following ECC errors:

- Single-bit errors which are confined to a single aligned quadword within a 32-byte Icache fill block
- Any combination of multiple single bit errors which occur within a 32-byte Dcache fill block

When a correctable ECC error occurs, the 21064/21064A corrects the error and posts a corrected-read interrupt if so enabled by ABOX_CTL [CRD_EN] and HIER [CRE]. The 21064/21064A also latches the physical address, syndrome, and other information in its internal BIU_STAT, FILL_SYNDROME and FILL_ADDR registers.

The 21064/21064A hardware cannot recover from the following ECC errors:

- Double-bit errors within a single longword or its associated 7-bit check field
- Multiple single-bit errors which corrupt more than a single quadword of an Icache fill block

When an uncorrectable ECC error occurs, the 21064/21064A traps to the PALcode machine check handler if enabled by ABOX_CTL [MCHK_EN], and latches information about the error in its internal BIU_STAT, FILL_SYNDROME and FILL_ADDR registers. If the uncorrectable error is due to single-bit errors in more than one quadword of an Icache fill block, a correctable-read interrupt will also be posted.

While hardware cannot recover from multiple single-bit errors which corrupt more than one aligned quadword of an Icache fill block, these errors may often be corrected by PALcode. If the machine check occurred while the processor was executing in native mode (as opposed to PALmode), PALcode may be able to recover by flushing the Icache and its associated stream buffer, scrubbing the corrupted block and returning. In effect, the combination of the 21064/21064A hardware and its associated PALcode can correct and recover from all true single-bit errors except those in which multiple single-bit errors corrupt more than one quadword of an Icache fill block while the processor is in PALmode.

6.6.1 Single-bit Errors

The error-reporting effects of several single-bit ECC errors are listed here.

Single-bit I-stream ECC Error—Single Corrupted Quadword

- Correct corrupted bits
- Post corrected-read interrupt if enabled by ABOX_CTL [CRD_EN]
- BIU_STAT: FILL_ECC, FILL_IRD and FILL_CRD set
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_SYNDROME contains syndrome bits associated with failing quadword
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

Single-bit I-stream ECC Error—Multiple Corrupted Quadwords

If the correction hardware receives a second corrupted quadword within an I-stream read transaction, it posts a machine check, if enabled by ABOX_CTL [MCHK_EN].

Single-bit D-stream ECC Error

- Correct quadword requested by CPU
- Invalidate Dcache
- Post corrected-read interrupt if enabled by ABOX_CTL [CRD_EN]
- BIU_STAT: FILL_ECC set, FILL_IRD clear, FILL_CRD set
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_ADDR [4:2] contain PA bits [4:2] of location that the failing load instruction attempted to read
- FILL_SYNDROME contains syndrome bits associated with failing quadword
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

6.6.2 Double-bit ECC Errors

The error-reporting effects of several double-bit ECC errors are listed here.

Double-bit I-stream ECC Error

- Corrupted data put into Icache, block gets validated
- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: FILL_DPERR set, FILL_IRD set, FILL_CRD clear
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_SYNDROME identifies corrupted longword(s)
- BIU_ADDR, BIU_STAT [6:0] locked—contents are UNPREDICTABLE
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

Double-bit D-stream ECC Error

- Corrupted data put into register file, Dcache invalidated
- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: FILL_DPERR set, FILL_IRD clear, FILL_CRD clear
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_ADDR [4:2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL_SYNDROME identifies corrupted longword(s)
- BIU_ADDR, BIU_STAT [6:0] locked—contents are UNPREDICTABLE
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

6.6.3 BIU Single Errors

The error-reporting effects of several Bus Interface Unit (BIU) single errors are listed here.

BIU: Tag Address Parity Error

- Recognized at end of tag probe sequence
- Lookup uses predicted parity so transaction misses the external cache
- BC_TAG holds results of external cache tag probe
- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: BC_TPERR set
- BIU_ADDR holds address

BIU: Tag Control Parity Error

- Recognized at end of tag probe sequence
- Transaction forced to miss external cache
- BC_TAG holds results of external cache tag probe
- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: BC_TCPERR set
- BIU_ADDR holds address

BIU: System External Transaction Terminated with CACK_SERR

- CRD interrupt posted if enabled by ABOX_CTL [CRD_EN]
- BIU_STAT: BIU_SERR set, BIU_CMD holds cReq_h [2:0]
- BIU_ADDR holds address

BIU: System Transaction Terminated with CACK_HERR

- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: BIU_HERR set, BIU_CMD holds cReq_h [2:0]
- BIU_ADDR holds address

BIU: I-stream Parity Error (parity mode only)

- Data put into Icache unchanged, block gets validated
- Machine check if enabled by ABOX_CTL [MCHK_EN]
- BIU_STAT: FILL_DPERR set, FILL_IRD set
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_SYNDROME identifies failing longword(s)
- BIU_ADDR, BIU_STAT [6:0] locked—contents are UNPREDICTABLE
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

BIU: D-stream Parity Error (parity mode only)

- Data put into Dcache unchanged, block gets validated
- Machine check if enabled by ABOX_CTL[MCHK_EN]
- BIU_STAT: FILL_DPERR set, FILL_IRD clear
- FILL_ADDR [33:5] and BIU_STAT [FILL_QW] give bad QW's address
- FILL_ADDR [4:2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL_SYNDROME identifies failing longword(s)
- BIU_ADDR, BIU_STAT [6:0] locked—contents are UNPREDICTABLE
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

6.6.4 Multiple Errors

This section describes the 21064/21064A's response to multiple hardware errors, that is, to errors which occur after an initial error and before execution of the PALcode exception handler associated with that initial error.

The 21064/21064A error reporting hardware consists of two sets of independent error reporting registers.

- BIU_STAT [7:0] and BIU_ADDR contain information about the following hardware errors:
  - Correctable or uncorrectable errors reported with cAck_h [2:0] by system components
  - Tag probe parity errors in the tag address or tag control fields
- BIU_STAT [14:8], FILL_ADDR and FILL_SYNDROME contain error information about data fill errors.

The BC_TAG register contains information that can relate to any of the error conditions listed above.

Each of the two sets of error registers can contain information about either corrected or uncorrected hardware errors. When a hardware error occurs information about that error is loaded into the appropriate set of error registers and those registers are locked against further updates until PALcode explicitly unlocks them. If a second error occurs between the time that an initial error occurs and the time that software unlocks the associated error reporting registers, information about the second is lost.

When the 21064/21064A recognizes the second error it still posts the required corrected-read interrupt or machine check, however it does not over write information previously locked in an error reporting register. If the second hardware error is not correctable and the error reporting register normally associated with this second error is already locked, the 21064/21064A will set a bit to indicate that information about an uncorrectable hardware error was lost. Each set of error reporting register has a bit to report these fatal errors.

For example, BIU_STAT [FATAL1] is set by hardware to indicate that a tag probe parity error or HARD_ERROR-terminated external transaction occurred while BIU_STAT [6:0], BIU_ADDR and BC_TAG were already locked due to some previous error. If a SOFT_ERROR-terminated transaction occurs while these registers are locked FATAL1 is not set, however. Similarly, BIU_STAT [FATAL2] is set by hardware to indicate that a primary cache fill received either a parity or double bit ECC error while BIU_STAT [13:8], FILL_ADDR, FILL_SYNDROME and BC_TAG were already locked.

BIU_STAT [FATAL2] will not be set by hardware when a primary cache fill receives a single bit ECC error while BIU_STAT [13:8], FILL_ADDR, FILL_SYNDROME and BC_TAG are already locked.

6.6.5 Cache Parity Errors—21064A Only

The **21064A** supports cache parity for data and tag on both Icache and Dcache.

6.6.5.1 Dcache Parity Errors—21064A Only

Dcache parity errors are nonrecoverable. In the event of a Dcache parity error the **21064A** will set C_STAT [DC_ERR] and will initiate a machine check if enabled by ABOX_CTL [MCHK_EN].

6.6.5.2 Icache Parity Errors—21064A Only

Icache parity errors are nonrecoverable. In the event of an Icache parity error the **21064A** will set C_STAT [IC_ERR] and will initiate a machine check if enabled by ABOX_CTL [MCHK_EN].

7
Electrical Data

7.1 Introduction

This chapter lists maximum power and maximum temperature ratings and includes ac and dc electrical data for the 21064/21064A Alpha AXP microprocessors.

7.2 Absolute Maximum Ratings

Table 7–1 lists the maximum ratings for the 21064/21064A microprocessor.

Table 7–1 21064/21064A Maximum Ratings

Characteristics	Ratings
Storage temperature	-55° C to 125° C (-67° F to 257° F)
Supply voltage	Vss -0.5 V, Vdd 3.6 V
Junction operating temperature	90° C (194° F)
Voltage applied to pins	
3 V tolerant pins	-0.5 V to Vdd + 0.5 V
5 V tolerant pins	-0.5 V to 5.5 V
Maximum power	See Section 7.3.4

Caution

Stress beyond the absolute maximum rating can cause permanent damage to the 21064/21064A. Exposure to absolute maximum rating conditions for extended periods of time can affect the 21064/21064A reliability.

7.2.1 Absolute Operating Limits

All of the 21064/21064A inputs can be driven to a 5 V nominal level by external logic except for the following:

- **clkIn_h** and **clkIn_l**
- **testclkIn_h** and **testclkIn_l**
- **tagOk_h** and **tagOk_l** (**21064 only**)[1]
- **dcOk_h**
- **eclOut_h**
- **tristate_l**
- **cont_l**

The 21064/21064A provides no clamping of positive input voltages on 5 V capable pins. In no case can an input transient exceed 6.5 V (above Vss) for reasons of device reliability.

7.3 dc Electrical Data

The 21064/21064A microprocessor uses CMOS/TTL voltages levels.

7.3.1 Power Supply

In CMOS mode the Vss pins are connected to 0.0 V and the Vdd pins are connected to 3.3 V nominal +/- 5%.

7.3.1.1 Power Consideration

Caution

To prevent damage to the 21064/21064A, it is important that the Vdd power supply be stable before any of its input or bidirectional pins be allowed to rise above 4.0 V.

To help meet this requirement, the assertion levels of the 21064/21064A's input pins are arranged so that their default state is electrically low. This makes them active high, with the exception of **tagOk_l** and **dOE_l**, which are true (low) by default.

[1] In the **21064A tagOk_h** and **tagOk_l** are reference to **vRef** and may be driven to a 5 V nominal level by external logic.

Once power has been applied and **vRef** has met its hold time, the majority of input pins can be driven by 5.0 V (nominal) signals without damaging the 21064/21064A.

Once power has been applied, input and bidirectional pins can be driven to a maximum dc voltage of 5.5 V without damaging the 21064/21064A. It is not necessary to use static RAMS with 3.3 V outputs.

7.3.1.2 Reference Supply

The **vRef** analog input should be connected to a 1.4 V +/-10% reference supply. See Section 7.4.1.

The reference supply (**vRef**) is an analog reference voltage used by the 21064 /21064A input buffers of all signals except:

- **clkIn_h** and **clkIn_l**
- **testclkIn_h** and **testclkIn_l**
- **tagOk_h** and **tagOk_l** (**21064 only**)[1]
- **dcOk_h**
- **eclOut_h**
- **tristate_l**
- **cont_l**

7.3.2 Input Clocks

The **clkIn_h** and **clkIn_l** are differential signals generated from an external oscillator circuit. The signals can be ac coupled (if Vcc to the oscillator is greater than Vdd), with nominal dc bias of Vdd/2 set by a high-impedance (that is greater than 1k ohm) resistive network on the chip. The signals need not be ac coupled if Vdd is used as the Vcc supply to the oscillator. Also, see Section 7.4.2.

[1] In the **21064A tagOk_h** and **tagOk_l** are reference to **vRef**.

7.3.3 Signal Pins

The 21064/21064A input pins are CMOS inputs that use standard TTL levels, set by **vRef**. Table 7–2 lists the dc input/output characteristics.

There are some signals that are sampled before **vRef** is stable. They cannot be driven above the power supply (Vdd). The signals are **dcOk_h**, **tristate_l** (3.3 V), **cont_l** (3.3 V), and **eclOut_h** (GND).

The 21064/21064A output pins are 3.3 V CMOS output that can be driven between Vdd and Vss. Timing is specified to standard TTL levels.

Table 7–2 dc Input/Output Characteristics

Symbol	Description	Min	Max	Units	Test Conditions
Vdd	Power supply voltage	3.135	3.465	V	–
Vih	High-level input voltage (except dcOk_h and cont_l)	2.0	–	V	–
Vihs	High-level input voltage (static pins dcOk_h,cont_l)	2.7	–	V	–
Vil	Low-level input voltage	–	0.8	V	–
Voh	High-level output voltage Ioh = 100 μA	2.4	–	V	–
Vol	Low-level output voltage Iol = 3.2 mA	–	0.4	V	–
Vdiffc	Differential clock input swing (duty cycle 45–55%)	600 mV	3.0	V	–
Iil	Input leakage current (except eclOut_h)	–100	100	μA	0<Vin<Vdd V
Iel	Input leakage current (eclOut_h)	–150	150	μA	0<Vin<Vdd V
Ioz	Output leakage current (tristate)	–100	100	μA	–
Icin	Clock input leakage	–4	4	mA	–0.5<Vin<3.465 V

Note

Values in this table are valid only for Vref = 1.4 V.

7.3.4 dc Power Dissipation

The formulas for calculating Idd (Max) and Idd (Peak) at varying frequencies and values of Vdd for the **21064** and **21064A** are listed here:

- **21064**

$$Idd(Max) = (116mA/V + 10.5mA/V * MHz * f) * Vdd$$
$$Idd(Peak) = (116mA/V + 12.9mA/V * MHz * f) * Vdd$$

- **21064A**

$$Idd(Max) = (116mA/V + 9.56mA/V * MHz * f) * Vdd$$
$$Idd(Peak) = (116mA/V + 11.5mA/V * MHz * f) * Vdd$$

f is the CPU frequency in MHz.
Vdd is the power supply voltage in volts.

Using the values calculated for Idd (Max) and Idd (Peak) it is then possible to calculate Power (Max) and Power (Peak) using the formulas formulas listed here.

- Using the listed values for the **21064**:

  f = 200 MHz (period is 5.0 ns) Vdd = 3.465 V (Max)

  The calculations would be:

$$Idd(Max) = 116 * 3.465 + 10.5 * 3.465 * 200 = 7678mA = 7.678A$$
$$Power(Max) = Vdd * Idd(Max) = 3.465 * 7.678 = 26.6W$$

$$Idd(Peak) = 116 * 3.465 + 12.9 * 3.465 * 200 = 9342mA = 9.342A$$
$$Power(Peak) = Vdd * Idd(Peak) = 3.465 * 9.342 = 32.36W$$

- Using the listed values for the **21064A**:

  f = 275 MHz (period is 3.64 ns) Vdd = 3.465 V (Max)

  The calculations would be:

$$Idd(Max) = 116 * 3.465 + 9.56 * 3.465 * 275 = 9511mA = 9.511A$$
$$Power(Max) = Vdd * Idd(Max) = 3.465 * 9.511 = 32.95W$$

$$Idd(Peak) = 116 * 3.465 + 11.5 * 3.465 * 275 = 11360mA = 11.36A$$
$$Power(Peak) = Vdd * Idd(Peak) = 3.465 * 11.36 = 39.36W$$

Note

Idd (Max) is used by thermal engineers.
Idd (Peak) is used by power supply designers to compute peak power.

7.4 ac Electrical Data

This section contains the ac characteristics for the 21064/21064A.

The 21064/21064A does provide silicon diode clamping of negative input transients.

Note

It is recommended that clamping current not exceed 25 mA per pin, nor should clamping charge exceed 50 pC per pin per transition. This is most important when large numbers of inputs participate simultaneously.

7.4.1 Reference Supply

Upon power-on, **reset_l** can not be sampled until **vRef** is stable. There is a large internal capacitance on **vRef**. There is a RC delay between **vRef** pin and the input buffers. Systems must not assert **dcOk_h** until a suitable interval following the stability of the **vRef** source. This interval is specified as the greater of 1 μs and 10 nF * Zout, when Zout is the **vRef** source impedance.

7.4.2 Input Clocks Frequency

The **clkIn_h** and **clkIn_l** input clocks have differential inputs.

Generally, the designer will apply standard 2x input clocks to the pin of **clkIn_h** and **clkIn_l**.

The 21064/21064A input clock circuit also allows for applications that require it to run on 1x input clocks (150 MHz input clocks on a 21064 150 MHz implementation).

To use the 21064/21064A with 1x input clocks, the designer need only to drive the 1x clock inputs into the **clkIn_h,l** pins and tie **testClkIn_h** and **testClkIn_l** to logic 1.

Note

Driving a clock into testClkIn pins will result in unpredictable behavior.

Electrically, the circuitry attached to the testClkIn pins is identical to the circuitry attached to the tristate pin. The same restrictions that are listed in Section 7.3 for the tristate pin apply to the testClkIn pins.

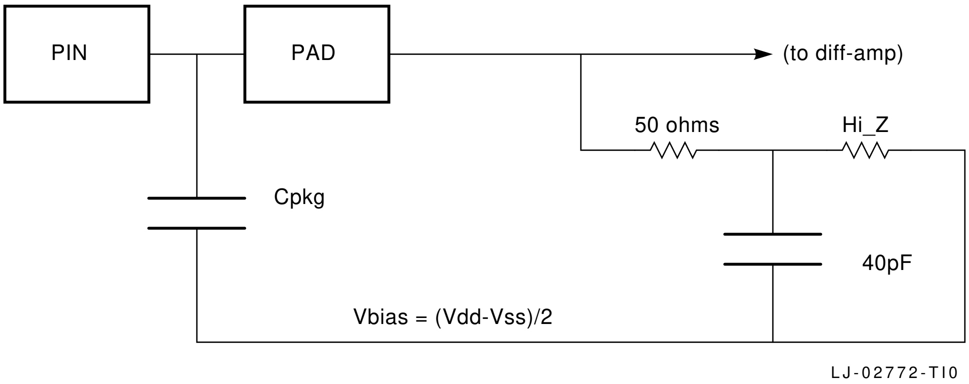
Table 7–3 lists the possible states of the testClkIn pins and the resulting functions.

Table 7–3 testClkIn Pins State

testClkIn_h	testClkIn_l	Functions
0	0	Reserved for Digital
0	1	Standard 2x input clocks applied to ClkIn pins
1	0	Standard 2x input clocks applied to ClkIn pins
1	1	1x input clocks applied to ClkIn pins

The termination on these signals are designed to be compatible with system oscillators of arbitrary dc bias. Figure 7–1 shows clock termination.

Figure 7–1 Clock Termination



The chip provides a 50 ohm termination (approximate) for the purpose of impedance matching for those systems that drive input clocks across long etches. The chip uses a high impedance bias driver that allows a clock source of arbitrary dc bias to be ac coupled to the clock input. The peak-to-peak amplitude of the clock source must be between 0.6 V and 3.0 V, as seen by the 21064/21064A. Either a "square-wave" or a sinusoidal source can be used.

Table 7–4 and Table 7–5 list the input clock cycle times for 21064/21064A speed bins. These periods equal one-half the corresponding CPU cycle times.

Table 7–4 21064 Input Clock Timing

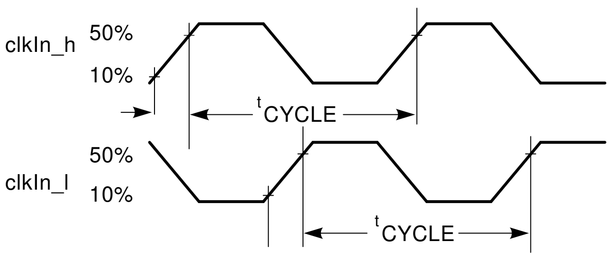
Name	21064-AA (21064-150)	21064-CA (21064-166)	21064-BA (21064-200)
clkIn period min	3.3 ns	3.0 ns	2.5 ns
clkIn symmetry	50%+/-10%	50%+/-10%	50%+/-10%

Table 7–5 21064A Input Clock Timing

Name	21064-BB (21064A-233)	21064-DB (21064A-275)
clkIn period min	2.15ns	1.82 ns
clkIn symmetry	50%+/-10%	50%+/-10%

Figure 7–2 shows the timing diagram for the input clock .

Figure 7–2 Input Clock Timing Diagram



7.4.3 Test Specification

The method of specification of the 21064/21064A timing parameters is constrained by VLSI tester limitations. Timing of the 21064/21064A inputs must be specified with respect to **vRef** crossings (set to midpoint on the tester), because the relatively slow tester-driven edges would otherwise distort the measured results.

The 21064/21064A generates the clocks with which the following times are specified:

- Setup
- Hold
- Delay

The drivers of these clocks are nominally identical to the signal drivers, with identical timing from the 21064/21064A internal timebase. Therefore, output delay is largely a matter of skew, independent of loads provided the loads are identical. Setup and hold times depend on the loads on the outgoing clocks. The test load most closely approximates a linear transmission line with modest end-of-line capacitance. The tester compensates for its nominal transmission delay through software. Therefore, the most appropriate clock load for purposes of specification is a modest lumped capacitance at the clock pin. This load, chosen as 15pF to approximate a worst-case end-of-line load on a system module, is taken as the standard load for all outputs (except **cpuClkOut_h**).

Note

System designers are responsible for adding (to setup times) or subtracting (from hold times) the additional delay appropriate to their systems.

The timing for all output signals, including clocks (except **cpuClkOut_h**) are specified with respect to their crossings of the midpoint from Vss to Vdd into a 15pF lumped capacitive load at the package pin. The "20/80" transition time of each signal into an open load is specified as less than 1.0 ns.

Note

The **cpuClkOut_h** signal and low-going bidirectional signals driven from 5 V are excepted. It is only the open load transition time specification that cannot be tested on a production basis, but margin should be adequate.

As a measure of output impedance, each output (except **cpuClkOut_h**) is specified to drive its pin to a dc value not more than 50% nor less than 35% from its intended rail when loaded by 50 ohms to the opposite rail. Timing for all input signals (except **tagOk_h** and **tagOk_l**) are specified with respect to the point at which they cross vRef at the 21064/21064A pin, assuming a skew rate of at least 1 V/ns at this point.

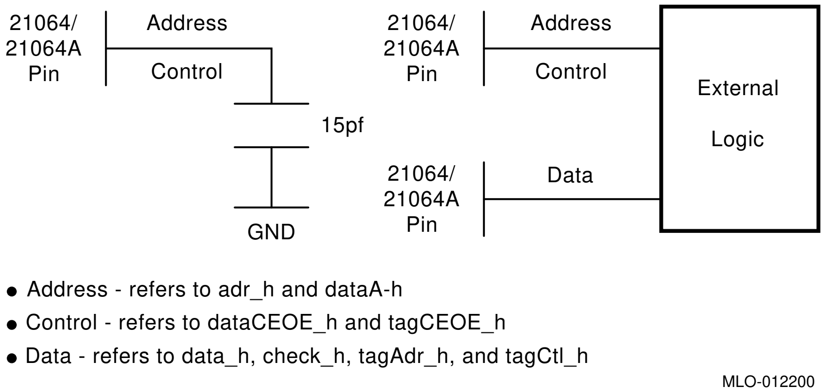
7.4.4 Fast Cycles on External Cache

From a system standpoint, fast cycles on the external cache are completely unclocked.

7.4.4.1 Fast Read Cycles

External logic must meet the maximum flow-through delay, as defined with respect to Figure 7–3.

Figure 7–3 Flow-Through Delay (External Cache)



Assume that address/control is driven from the same internal clock edge in the two cases shown in Figure 7–3. External flow-through delay (propagation delay) is defined as the delay between address/control valid to the 15pF standard load in the case on the left and data valid to the 21064/21064A (using a **vRef** threshold) in the case on the right. It cannot exceed the fast read cycle time:

- (BC_RD_SPD+1 CPU cycle) less 4.5 ns for the **21064**
- (BC_RD_SPD+1 CPU cycle) less 4.0 ns for the **21064A**

The 21064/21064A guarantees that its address drivers are enabled at least one CPU cycle prior to a fast cache access, such that **adr_h** does not need to be pulled down from 5 V during the cycle.

7.4.4.2 Fast Write Cycles

External logic must guarantee that fast writes complete. Data, address, and control (including **dataWE_h** and **tagCtlWE_h**) are driven by the 21064/21064A with identical timing from its internal clock. The timing of **dMapWE_h** during Dcache read hits is specified in the same way.

7.4.5 External Cycles

All external cycle timing is referenced to the rising edge of **sysClkOut1_h**. The minimum values for output delay are negatives, reflecting the fact that data can switch before **sysClkOut1_h**. This is possible because there is no cause-effect relationship between the system clock outputs and data. The system clock outputs are described as data pins that happen to switch in a fixed pattern.

Output enable time is defined as output delay from a high impedance state. It can generally exceed standard output delay because it can entail pulling the signal down from a 5 V level.

Address enable timing is relevant only for systems using the holdReq protocol with two CPU cycles per system cycle. All bidirectional lines can be considered enabled or disabled simultaneously with the rising edge of **sysClkOut1_h**. Table 7–6 lists the referenced times to **sysClkOut1_h**.

Table 7–6 External Cycles

Name	Minimum	Maximum	Units
Output Enable, sysClkOut1_h to			
adr_h	-1.0	2.0	ns
data_h (WRITE_BLOCK)	-1.0	2.0	ns
check_h (WRITE_BLOCK)	-1.0	2.0	ns
Output Delay, sysClkOut1_h to			
adr_h	-1.0	1.0	ns
data_h (WRITE_BLOCK)	-1.0	1.0	ns
check_h (WRITE_BLOCK)	-1.0	1.0	ns
cReq_h	-1.0	1.0	ns
cWMask_h	-1.0	1.0	ns
holdAck_h	-1.0	1.0	ns

Note: This timing is valid by design.

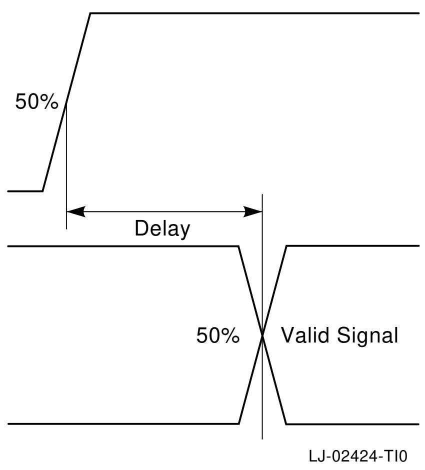
(continued on next page)

Table 7–6 (Cont.) External Cycles

Name	Minimum		Maximum	Units
Input Setup relative to sysClkOut1_h	**21064**	**(21064A)**		
cAck_h	9.3	(7.0)	–	ns
dRAck_h	9.3	(7.0)	–	ns
dWSel_h	9.3	(7.0)	–	ns
dOE_l	9.3	(7.0)	–	ns
holdReq_h	4.8	(3.8)	–	ns
dInvReq_h	4.5	(3.5)	–	ns
iAdr_h	4.5	(3.5)	–	ns
data_h (READ_BLOCK)	3.5	(2.5)	–	ns
check_h (READ_BLOCK)	3.5	(2.5)	–	ns
perf_cnt_h	4.5	(3.5)	–	ns
Input Hold relative to sysClkOut1_h				
cAck_h	0		–	ns
dRAck_h	0		–	ns
dWSel_h	0		–	ns
dOE_l	0		–	ns
holdReq_h	0		–	ns
dInvReq_h	0		–	ns
iAdr_h	0		–	ns
data_h (READ_BLOCK)	0		–	ns
check_h (READ_BLOCK)	0		–	ns
perf_cnt_h	0		–	ns

Figure 7–4 shows the 21064/21064A output delay measurement.

Figure 7–4 Output Delay Measurement



Note

This delay could be positive or negative.

Figure 7–5 shows the 21064/21064A setup and hold time measurement.

Figure 7–5 Setup and Hold Time Measurement

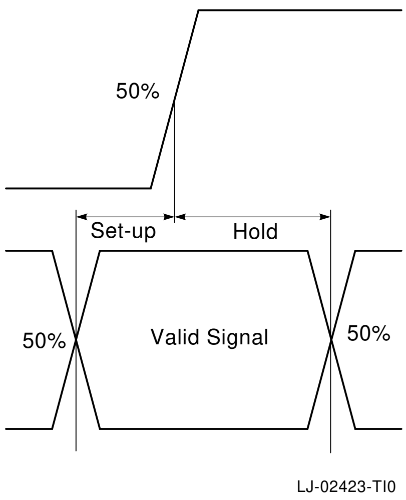
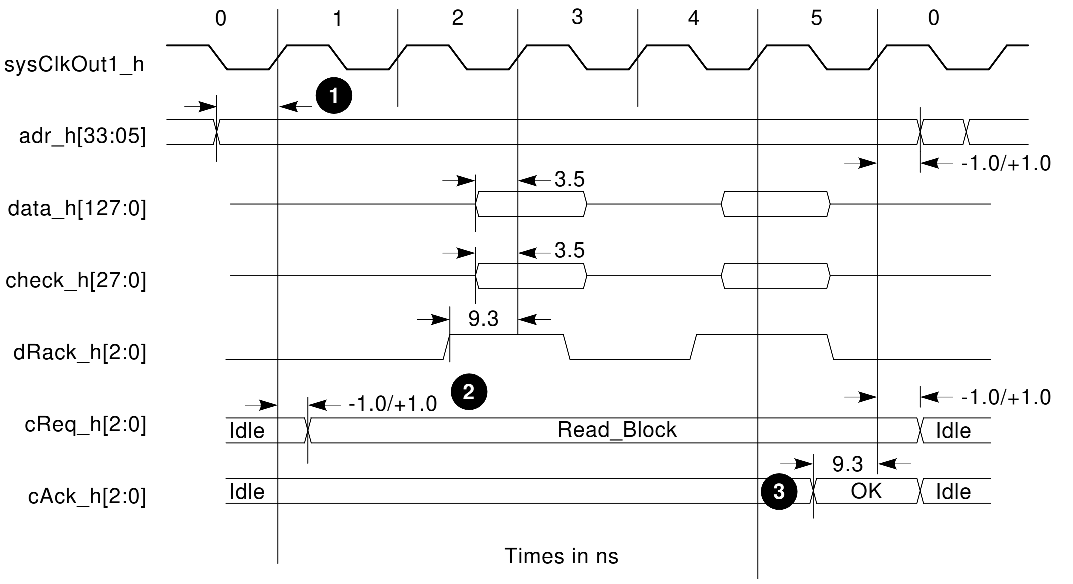


Figure 7–6 shows the **21064** READ_BLOCK timing diagram.

Figure 7–6 21064 READ_BLOCK Timing Diagram



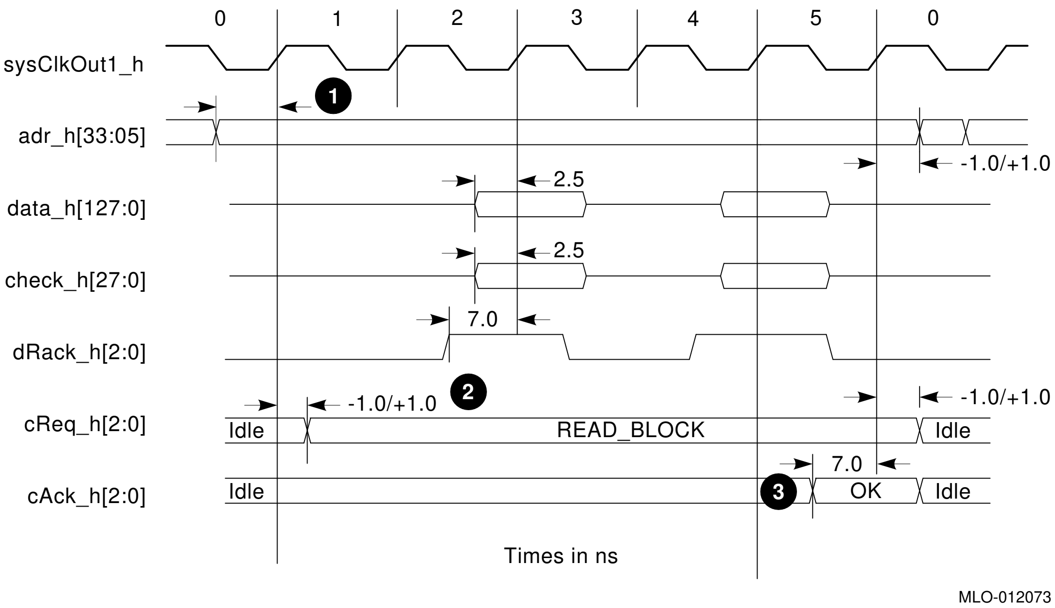
❶ tcycle ± 1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–7 shows the **21064A** READ_BLOCK timing diagram.

Figure 7–7 21064A READ_BLOCK Timing Diagram



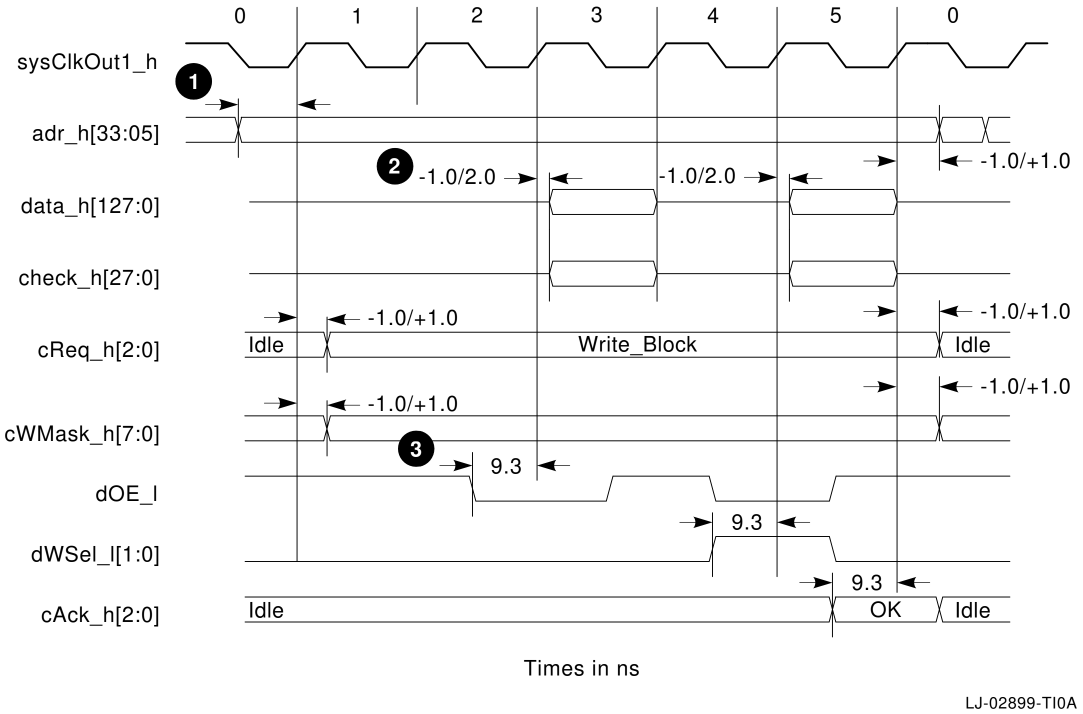
❶ tcycle ± 1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–8 shows the **21064** WRITE_BLOCK timing diagram.

Figure 7–8 21064 WRITE_BLOCK Timing Diagram



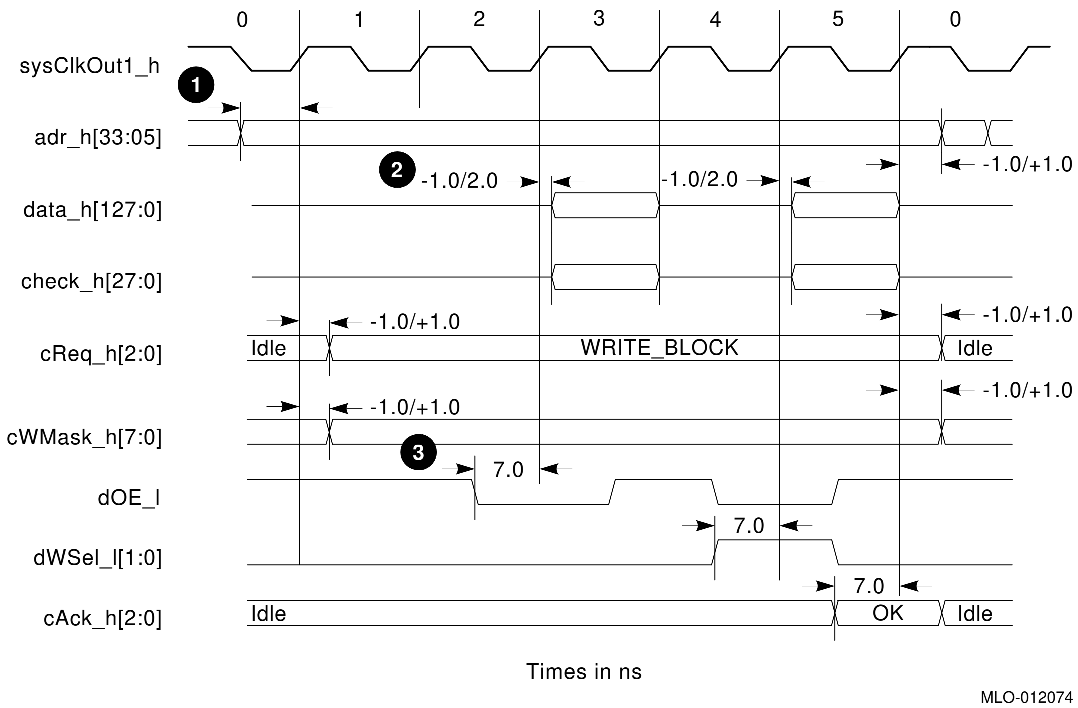
❶ 3x tcycle ± 1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–9 shows the **21064A** WRITE_BLOCK timing diagram.

Figure 7–9 21064A WRITE_BLOCK Timing Diagram



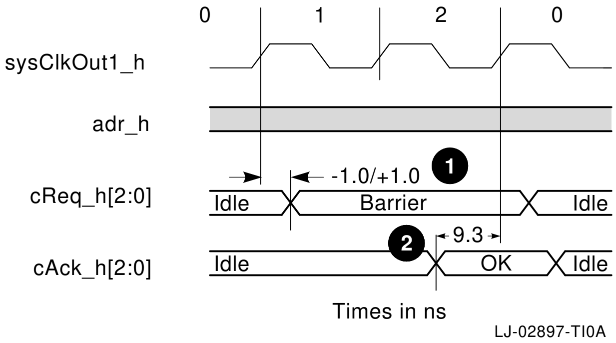
❶ 3x tcycle ± 1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–10 shows the **21064** BARRIER timing diagram.

Figure 7–10 21064 BARRIER Timing Diagram

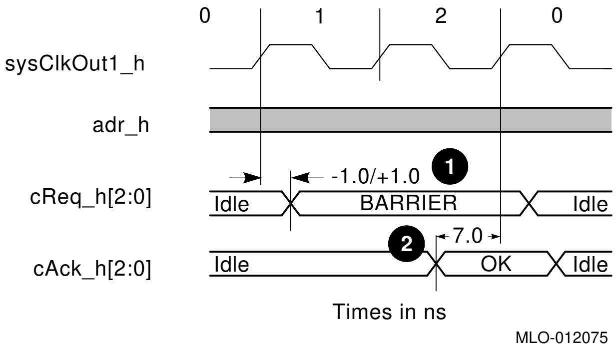


❶ Indicates minimum/maximum

❷ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–11 shows the **21064A** BARRIER timing diagram.

Figure 7–11 21064A BARRIER Timing Diagram

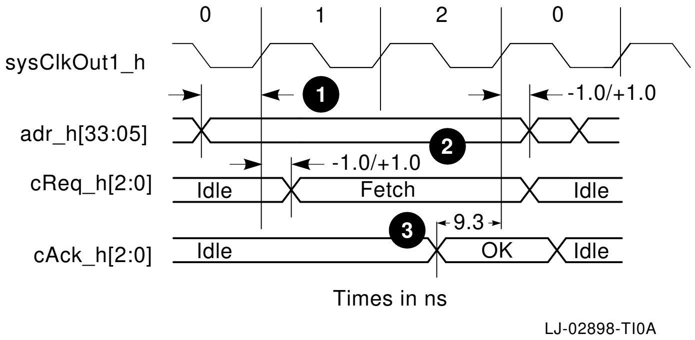


❶ Indicates minimum/maximum

❷ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–12 shows the **21064** FETCH/FETCH_M timing diagram.

Figure 7–12 21064 FETCH/FETCH_M Timing Diagram



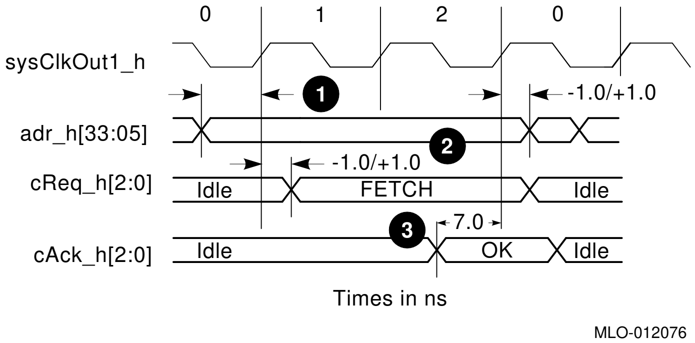
❶ tcycle ±1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

Figure 7–13 shows the **21064A** FETCH/FETCH_M timing diagram.

Figure 7–13 21064A FETCH/FETCH_M Timing Diagram



❶ tcycle ±1.0 ns, where tcycle = period of **cpuClkOut_h**

❷ Indicates minimum/maximum

❸ Minimum setup time shown. All hold times are a minimum of 0.0 ns.

7.4.6 tagEq_l (21064 only)

Active during external cache hold, the timing of **tagEq_l** is specified when its inputs become valid at the **21064** pins.

Table 7–7 lists the **21064** tagEq_l timing.

Table 7–7 tagEq_l Timing

Name	Min	Max	Units
Delay, adr_h -> tagEq_l	—	Tcyc/2 + 17.0	ns
Delay, tagAdr_h -> tagEq_l	—	Tcyc/2 + 17.0	ns

Note

The delay to **tagEq_l** is a function of the chip cycle time (Tcyc). At 6.6 ns cycle time, this delay is 20.3 ns.

The signal line **tageq_l** is not implemented in the **21064A**.

7.4.7 21064 tagOk Synchronization

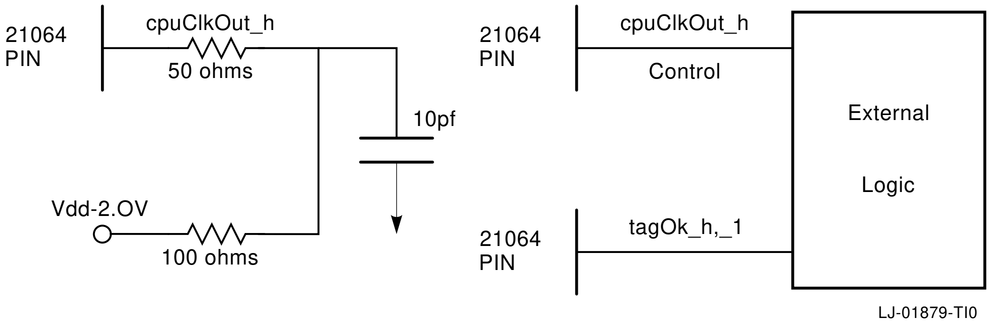
The **cpuClkOut_h** signal is to be used only by a synchronizer in **21064** systems using the tagOk protocol. In order to accommodate ECL levels, the driver consists of only a PMOS pullup device. ECL 100K levels can be constructed with a 50 ohm resistor in series with the driver and a 100 ohm resistor between the load and Vdd minus 2 volts. CMOS Vdd must equal Vcc in this scheme.

Note

The connections to the 21064/21064A must be electrically short to ensure good signal integrity and maintain a stable circuit impedance.

The **21064** receives the **tagOk_h** and **tagOk_l** signals directly from the final stage of a synchronizer, which is clocked by **cpuClkOut_h**. As in the case of fast external cache cycles, the system must meet a maximum flow-through delay. This delay is defined in Figure 7–14.

Figure 7–14 Flow-Through Delay (TagOk)



The **cpuClkOut_h** signal is driven from the same **21064** internal clock edge in the two cases shown in Figure 7–14. External flow-through delay is defined as the delay between **cpuClkOut_h** valid to the 10pF ECL "standard" load in the case on the left and **tagOk_h** and **tagOk_l** valid to the **21064** in the case on the right. It can not exceed the nominal CPU cycle time minus 3.9 ns.

Note

Resistors on the printed circuit board are considered as part of the external logic in the circuit on the right (Figure 7–14).

The **cpuClkOut_h** signal is considered valid when it crosses the ECL threshold Vbb (equal to roughly Vcc minus 1.3V). The **tagOk_h** or **tagOk_l** signal is considered valid when the differential lines cross each other.

7.4.8 21064A tagOk Synchronization

The **21064A** includes an on-chip synchronizer circuit for **tagOk_h** and **tagOk_l** which will add a worst case delay of three CPU clock cycles to the path.

tagOk_h and **tagOk_l** are both single-ended inputs referenced to **Vref**.

Systems which use **tagOk_h** should tie **tagOk_l** to Vss.
Systems which use **tagOk_l** should tie **tagOk_h** to Vdd.
Systems which do not use the **tagOK** signal lines should tie **tagOk_h** to Vdd and **tagOk_l** to Vss.

7.4.9 Tester Considerations

Timing characteristics which should be considered when planning to test the 21064/21064A are presented in this section.

7.4.9.1 Asynchronous Inputs

The following signals are asynchronous:

- **reset_l**
- **irq_h**
- **sRomD_h** (when in software-controlled UART mode)

For test purposes, these signals should be driven synchronously with **sysClkOut1_h** with the timing given in Table 7–8.

Note

These parameters are given with respect to the rising edge of **sysClkOut1_h**.

Table 7–8 Asynchronous Signals During Test

Name	Min	Max	Units
Setup, reset_l -> sysClkOut1_h	5.0	—	ns
Setup, irq_h -> sysClkOut1_h	5.0	—	ns
Hold, irq_h -> sysClkOut1_h	0	—	ns
Setup, sRomD_h -> sysClkOut1_h	5.0	—	ns
Hold, sRomD_h -> sysClkOut1_h	0	—	ns

7.4.9.2 Signals Timed from CPU Clock

It is expected that speed testing will be done with the test clock equal to system clock (**sysClkOut1_h**). Fast external cache operation and serial ROM operation are timed from the internal CPU clock. Therefore, the following transactions can occur at different time points within a tester cycle from one cycle to the next:

- Input sampling
- Output enabling
- Switching

The number of such points is finite, equal to the number of CPU cycles per tester cycle.

For any given transaction, each signal will have its standard external cycle timing with respect to the rising edge of **sysClkOut1_h** or to a phantom edge offset from **sysClkOut1_h** by exactly an integer number of CPU cycles.

Note

The following signals have the same delay timing as **adr_h**:

- **dataA_h**
- **dataCEOE_h**
- **dataWE_h**
- **tagCEOE_h**
- **tagCtlWE_h**
- **dMapWE_h**

Outputs can be sampled deterministically with appropriate placement of the tester strobe, and inputs can be received deterministically with appropriate placement of the edge of the driving signal.

Bidirectional signals present a different problem. The tester can enable or disable a given driver at just one point within its cycle. It must in the worst case drive an input beyond its 21064/21064A sample point by at least (N-1) CPU cycles. (N is the number of CPU cycles per system cycle.) In the worst case, the 21064/21064A will enable its drivers just one CPU cycle after sampling (for example, **tagCtl_h** following probe write).

The serial ROM outputs **sRomOE_l** and **sRomClk_h** can be strobed with the same timing as the **data_h** pins when driven by the 21064/21064A. The serial ROM input **sRomD_h** can be switched with the same timing used in serial port mode.

Note

SPICE simulation models are available for 21064/21064A I/Os.

8
Thermal Management

8.1 Introduction

This chapter describes the thermal issues that should be considered by a designer when using the 21064/21064A. The chapter is organized as follows:

- Introduction
- Thermal Device Characteristics
- Thermal Management Techniques
- Critical Parameters of Thermal Design

Note

The overall enclosure and power supply must be designed to handle the maximum power value, as stated in Section 7.2.

All necessary information to design a printed circuit board (PCB) or system for the adequate cooling of the 21064/21064A can be found in the following sections.

Note

The combination of airflow, heat sink design, and the package thermal characteristics must be considered when calculating the power dissipation to not exceed the maximum junction temperature (Tj).

The 21064/21064A is specified with a maximum power dissipation for a recommended:

- Maximum junction temperature
- Thermal resistance from die junction to case
- Thermal resistance from case to ambient

This chapter provides a method of evaluating the environment, specifically air flow and ambient temperature requirements. It also provides the designer with the information to design a cooling method that meets the thermal performance requirements, depending upon the constraints of the PCB environment.

8.2 Thermal Device Characteristics

The 21064/21064A is a high performance chip that has some stringent thermal characteristics which need to be considered when evaluating a method of cooling the device.

8.2.1 21064/21064A Die and Package

The 21064/21064A is packaged in 431 pin alumina-ceramic (cavity-down) package. This cavity-down design allows the die to be attached to the top surface of the package, which increases the ability of the die to dissipate the heat through the package and attached heat sink surface. A metal slug with two mounting studs is brazed on the ceramic package for the heat sink assembly. The slug is 3.18 cm (1.25 in) in diameter and 0.089 cm (0.035 in) thick. The package has mounting pads for 28 capacitors on the top surface that limits the heat sink contact area to 3.18 cm (1.25 in) in diameter. The specific dimensions of the heat sink should be determined by the designer to meet the thermal requirements, based upon the:

- Available room in the system
- Ambient temperature
- Air flow in the system

8.2.2 Power Consideration

The 21064/21064A has a maximum power rating, which varies depending on the operating frequency. The approximate power dissipation for the 21064 /21064A, which varies directly with frequency, can be calculated as follows:

21064-150 $Power = (21.0/150) * f$
21064-166 $Power = (22.5/1\ 66) * f$
21064-200 $Power = (2\ 7.0/200) * f$
21064A-233 $Power = (2\ 8.0/233) * f$
21064A-275 $Power = (33.0/275) * f$

In the equation f equal the frequency in MHz.

8.2.3 Relationships Between Thermal Impedance and Temperatures

The junction to ambient and junction to case thermal resistance values are used as measures of device thermal performance. These parameters are defined by the following equations:

$\theta ja = (Tj - Ta)/P$
$\theta jc = (Tj - Tc)/P$
$\theta ja = \theta jc + \theta ca$

An alternative equation is: $Tj = Ta + P * \theta ja$

In the equations,

θ ja is the junction to ambient thermal resistance (C/W).

θ jc is the junction to case thermal resistance (C/W). θ jc is defined from the device junction to the center of the heat sink.

θ ca is the case to ambient thermal resistance (C/W).

Tj is the maximum junction temperature.

Tc is the case temperature at a predefined location (° C). Tc is defined as the heatsink temperature assuming a Grafoil pad is used as the interface material with proper heatsink assembly procedure.

Ta is the ambient temperature.

P is the power dissipation in watts (W).

C/W degrees centigrade per watt.

θ jc is a measure of package internal thermal resistance from the silicon die to the package exterior. This value is strongly dependent upon packaging materials, thermal conductivities, and package geometry and is therefore generally fixed. θ ca values include the conductive and convective thermal resistance from package exterior to the ambient air. θ ca values depend on package geometry as well as environmental conditions such as flow rate and coolant physical properties.

The components and locations for temperature measurements are listed as follows:

❶ Heat sink temperature (Ths)

❷ Case temperature (Tc)

❸ Junction temperature (Tj)

❹ Package lid

❺ DECchip 21064 or DECchip 21064A

❻ Package

❼ Grafoil

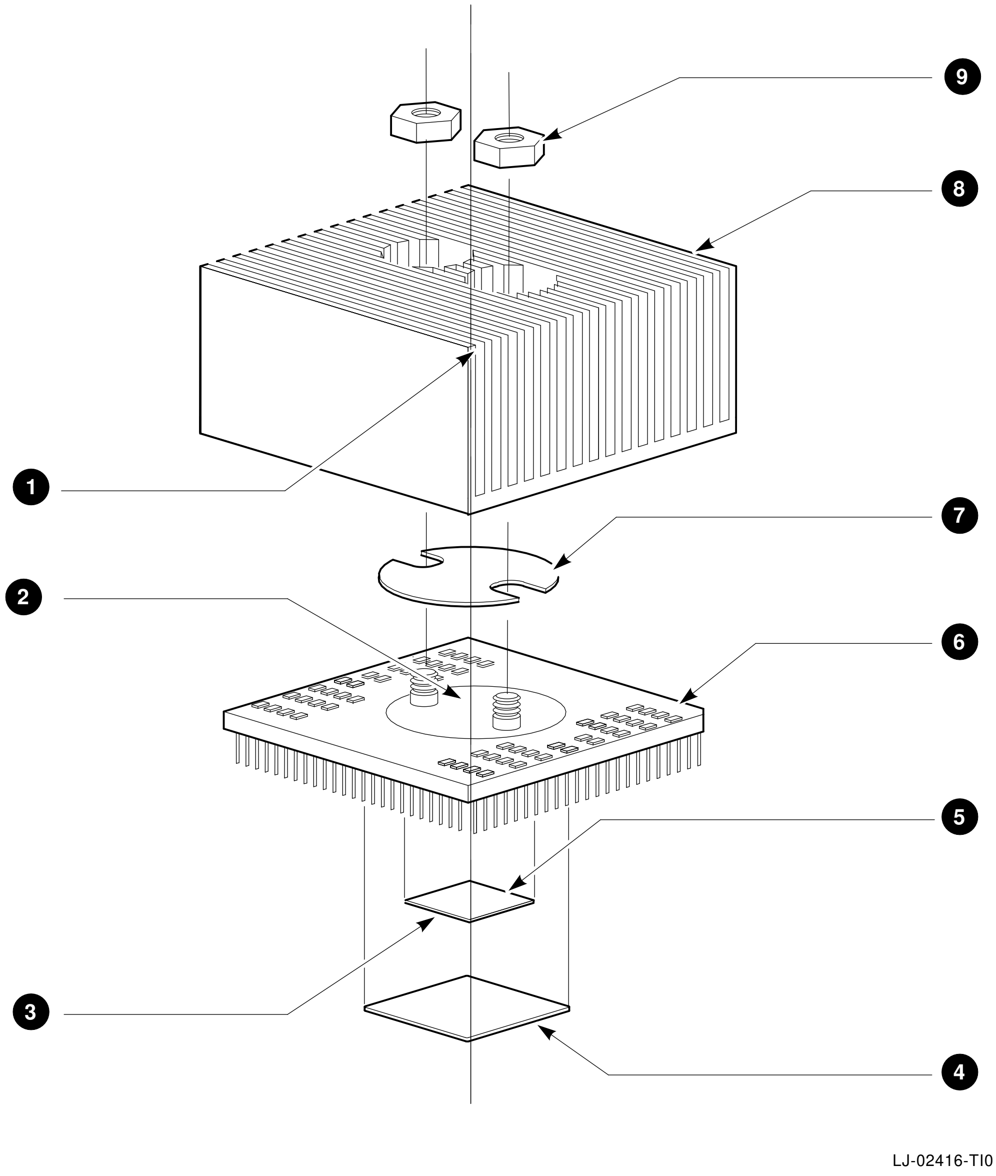
❽ Heat sink

❾ Nut

Figure 8–1 labels all the components of the package and the locations for temperature measurements.

The total thermal resistance of a package, θ ja, is a combination of its two components, θ jc and θ ca. These components represent the barrier to heat flow from the semiconductor junction to the package surface (θ jc) and from the surface to the outside ambient (θ ca). θ jc is device related and it cannot be influenced by the user but θ ca can be controlled by the user. Good thermal management by the user can significantly reduce θ ca achieving either a lower junction temperature or allowing a higher ambient operating temperature for a given air flow condition. θ ca can be reduced by applying thermal management techniques.

Figure 8–1 Package Components and Temperature Measurement Locations



8.3 Thermal Management Techniques

There are a number of thermal management techniques developed to keep θ ca very low. The following thermal management techniques are either being used or being considered in the industry:

- Forced air cooling
- Liquid cooling
- Heatpipe cooling
- Air or liquid impingement cooling
- Immersion cooling
- Peltier cooling
- Refrigeration cooling

Only the forced air cooling method is described in this section because of its wide use in the industry. It is one of the most inexpensive methods and a simple thermal management technique.

8.3.1 Thermal Characteristics with a Heat Sink and Forced Air

In choosing a heat sink, the designer must consider many factors:

- Heat sink size
- Material
- Method of attachment
- Interface material
- Heat sink orientation

Package orientation with respect to the air flow direction is very critical as the designed heat sinks are bidirectional. The package must be oriented so the air flow direction is parallel to the direction of heat sink fins.

The heat sink size is an important parameter in heat sink design. A large heat sink will provide better cooling. The most benefit of a large heat sink (of the bidirectional fin type) would be at lower air flow conditions. In about 100 lfpm air flow, the difference in value of θ ja with and without the heat sink is approximately four times, which decreases to two times at 1000 lfpm.

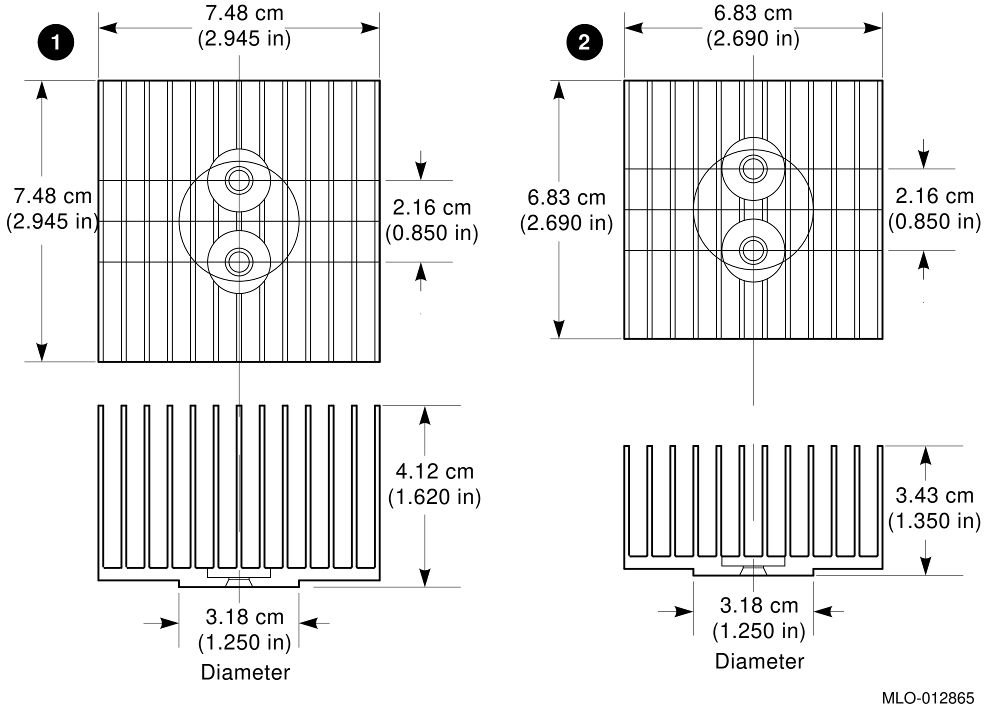
8.3.2 Heat Sink Design Considerations

The surface of the heat sink pedestal that mates to the package must have a high degree of planarity and a specification needs to be determined for the planarity of the surface. Sufficient space between fins will increase the heat sink effectiveness as well as reduce the pressure head requirement. The heat sink base and fin thicknesses must be designed to minimize the spreading resistance in the heat sink material.

8.3.3 Package and Heat Sink Thermal Performance

Figure 8–2 shows two examples of heat sinks which may be used to help cool the 21064/21064A. The primary heat sink (number 2 in Figure 8–2) is available from Digital. The pedestal, which is 3.18 cm (1.25 in) in diameter and 0.178 cm (0.070 in) thick, makes contact with the slug on the package. The heat sink is machined and made of an aluminum alloy.

Figure 8–2 Heat Sinks Dimensions



Although there are several ways to attach a heat sink to the package, the 21064/21064A uses a separable heat sink attached with two aluminum nuts. A grafoil pad is used as the interface material between the package and the heat sink to reduce the contact thermal resistance. A wet contact (thermal grease) has been avoided for the ease of heat sink assembly. Table 8–1 shows the thermal test results of this heatsink assembly. It shows θ jc, θ ca, and θ ja as measured, along with the maximum ambient temperature in order to maintain the maximum specified die temperature of:

- 90°C (194°F) for 21064-200
- 90°C (194°F) for 21064-150 and 21064-166
- 90°C (194°F) for 21064A-233 and 21064A-275

Table 8–1, Table 8–2, and Table 8–3 show the thermal characteristics for the **21064**.

Table 8–1 21064-150 Thermal Characteristics in a Forced-Air Environment

21064 at 150 MHz — Tc=75°C (167°F)					
		Heatsink 1		Heatsink 2	
Air Velocity	Power	TaMax	θca	TaMax	θca
100 lfpm	21.0 W	48.8°C (119.8°F)	1.25 C/W	40.4°C (104.7°F)	1.65 C/W
200 lfpm	21.0 W	57.2°C (135.0°F)	0.85 C/W	49.8°C (121.6°F)	1.20 C/W
400 lfpm	21.0 W	62.4°C (144.3°F)	0.60 C/W	57.2°C (135.0°F)	0.85 C/W
600 lfpm	21.0 W	64.1°C (147.4°F)	0.52 C/W	61.4°C (142.5°F)	0.65 C/W
1000 lfpm	21.0 W	66.6°C (151.9°F)	0.40 C/W	63.2°C (145.8°F)	0.56 C/W

Table constants and abbreviations

Tj is 90°C (194°F).
θjc is 0.7 C/W.
lfpm is linear feet per minute

Table 8–2 21064-166 Thermal Characteristics in a Forced-Air Environment

21064 at 166 MHz — Tc=72°C (161.6°F)					
		Heatsink 1		Heatsink 2	
Air Velocity	**Power**	**TaMax**	**θca**	**TaMax**	**θca**
100 lfpm	22.5 W	43.9°C (111.0°F)	1.25 C/W	34.9°C (94.8°F)	1.65 C/W
200 lfpm	22.5 W	52.9°C (127.2°F)	0.85 C/W	45.0°C (113.0°F)	1.20 C/W
400 lfpm	22.5 W	58.5°C (137.3°F)	0.60 C/W	52.9°C (127.2°F)	0.85 C/W
600 lfpm	22.5 W	60.3°C (140.5°F)	0.52 C/W	57.4°C (135.3°F)	0.65 C/W
1000 lfpm	22.5 W	63.0°C (145.4°F)	0.40 C/W	59.4°C (138.9°F)	0.56 C/W

Table constants and abbreviations

Tj is 90°C (194°F).
θjc is 0.7 C/W.
lfpm is linear feet per minute

Table 8–3 21064-200 Thermal Characteristics in a Forced-Air Environment

21064 at 200 MHz — Tc=70°C (158.0°F)					
		Heatsink 1		Heatsink 2	
Air Velocity	**Power**	**TaMax**	**θca**	**TaMax**	**θca**
100 lfpm	27.0 W	36.3°C (97.3°F)	1.25 C/W	25.5°C (77.9°F)	1.65 C/W
200 lfpm	27.0 W	47.1°C (116.8°F)	0.85 C/W	37.6°C (97.7°F)	1.20 C/W
400 lfpm	27.0 W	53.8°C (128.8°F)	0.60 C/W	47.1°C (116.8°F)	0.85 C/W
600 lfpm	27.0 W	56.0°C (132.8°F)	0.52 C/W	52.5°C (126.5°F)	0.65 C/W
1000 lfpm	27.0 W	59.2°C (138.6°F)	0.40 C/W	54.9°C (130.8°F)	0.56 C/W

Table constants and abbreviations

Tj is 90°C (194°F).
θjc is 0.7 C/W.
lfpm is linear feet per minute

Table 8–4 and Table 8–5 show the thermal characteristics for the **21064A**.

Table 8–4 21064A-233 Thermal Characteristics in a Forced-Air Environment

21064A at 233 MHz — Tc=71.0°C (159.8°F)					
		Heatsink 1		Heatsink 2	
Air Velocity	**Power**	**TaMax**	**θca**	**TaMax**	**θca**
100 lfpm	28.0 W	36.0°C (96.8°F)	1.25 C/W	24.8°C (76.6°F)	1.65 C/W
200 lfpm	28.0 W	47.2°C (117.0°F)	0.85 C/W	37.4°C (99.3°F)	1.20 C/W
400 lfpm	28.0 W	54.2°C (129.6°F)	0.60 C/W	47.2°C (117.0°F)	0.85 C/W
600 lfpm	28.0 W	56.4°C (133.5°F)	0.52 C/W	52.8°C (127.0°F)	0.65 C/W
1000 lfpm	28.0 W	59.8°C (139.6°F)	0.40 C/W	55.3°C (131.5°F)	0.56 C/W

Table constants and abbreviations

Tj is 90°C (194°F).
θjc is 0.7 C/W.
lfpm is linear feet per minute

Table 8–5 21064A-275 Thermal Characteristics in a Forced-Air Environment

21064A at 275 MHz — Tc=67.0°C (152.6°F)					
		Heatsink 1		Heatsink 2	
Air Velocity	**Power**	**TaMax**	**θc**	**TaMax**	**θca**
100 lfpm	33.0 W	25.8°C (78.4°F)	1.25 C/W	—	—
200 lfpm	33.0 W	39.0°C (102.2°F)	0.85 C/W	27.4°C (81.3°F)	1.20 C/W
400 lfpm	33.0 W	47.2°C (117.0°F)	0.60 C/W	39.0°C (102.2°F)	0.85 C/W
600 lfpm	33.0 W	49.8°C (121.6°F)	0.52 C/W	45.6°C (114.0°F)	0.65 C/W
1000 lfpm	33.0 W	53.8°C (128.8°F)	0.40 C/W	48.5°C (119.3°F)	0.56 C/W

Table constants and abbreviations

Tj is 90°C (194°F).
θjc is 0.7 C/W.
lfpm is linear feet per minute

Note

The values in Table 8–4 and Table 8–5 are base upon the assumption that maximum power will be 28.0 W and 33.0 W for the 21064A-233 and 21064A-275 respectively.

8.3.3.1 Comparison of Thermal Performance of Various Heat Sink Designs

One heat sink has been chosen as the primary design (which is available from Digital); others have been characterized. The other two designs are shown with different form factors and are targeted for relatively high air velocity systems with tighter printed circuit board spacing.

Figure 8–3 compares the overall dimensions of the three heat sinks designs. As shown in Figure 8–3:

- Heat sink number ❷ is relatively smaller (in all 3 dimensions) than the primary heat sink (heat sink number ❶). However, it has more fins (17 fins compared to 12 fins).
- Heat sink number ❷ can be used in relatively high air velocity systems where the spacing between the adjacent fins can be reduced. The extra fins provide more surface area.
- Heat sink number ❸ is very short compared to the primary heat sink ❶. It is also significantly wider and longer than the primary heat sink. Heat sink number ❸ can be used in systems where the printed circuit board spacing is very tight. The penalty for the use of this heat sink is that the large footprint required for the heat sink consumes printed circuit board space. This heat sink can be used only in high air velocity systems.

From the design and the performance of the three heat sinks, it can be seen that by sacrificing height, either the large heat sink (XY dimensions) must be used or the air flow must be increased or the ambient temperature must be kept lower or some combination.

Figure 8–3 Comparison of Dimensions for Heat Sink Designs

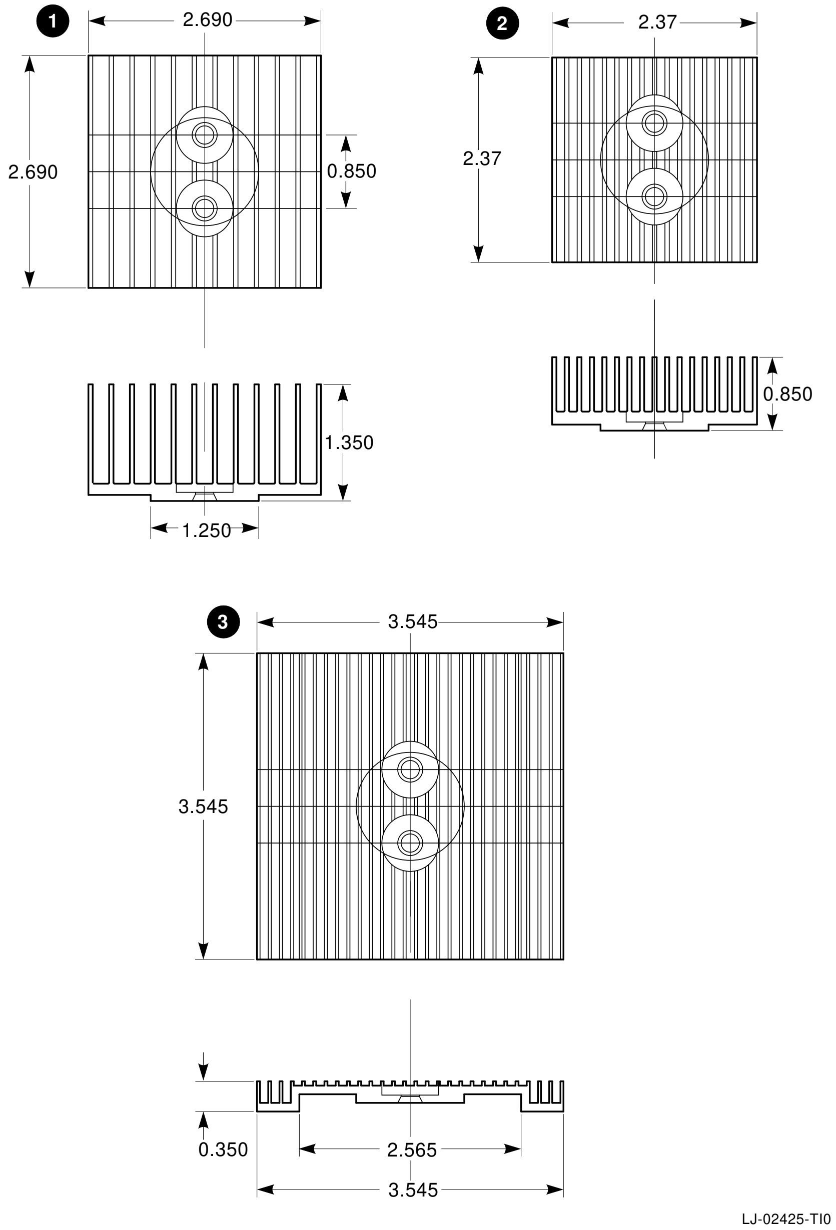


Figure 8–4 compares the thermal performance of a microprocessor using the three different heat sinks. Figure 8–5 compares the maximum ambient temperature allowed in a system as a function of air velocity using the three heat sinks.

Figure 8–4 Microprocessor Thermal Performance

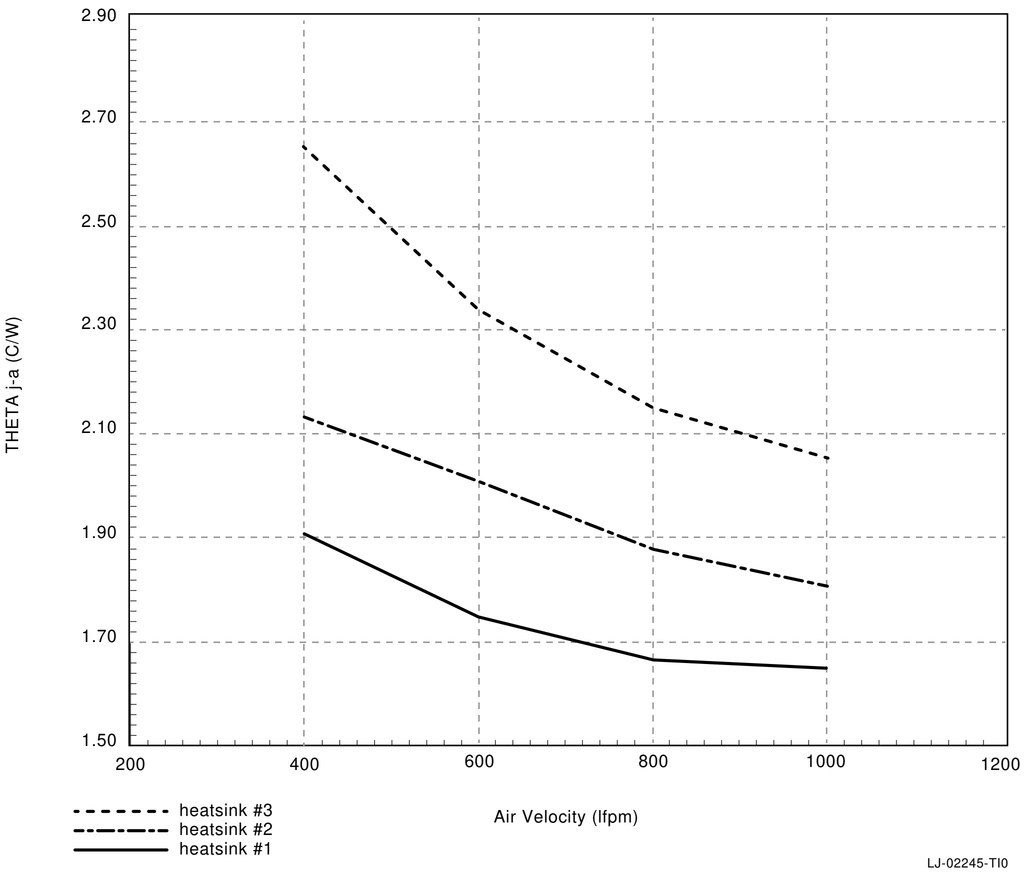
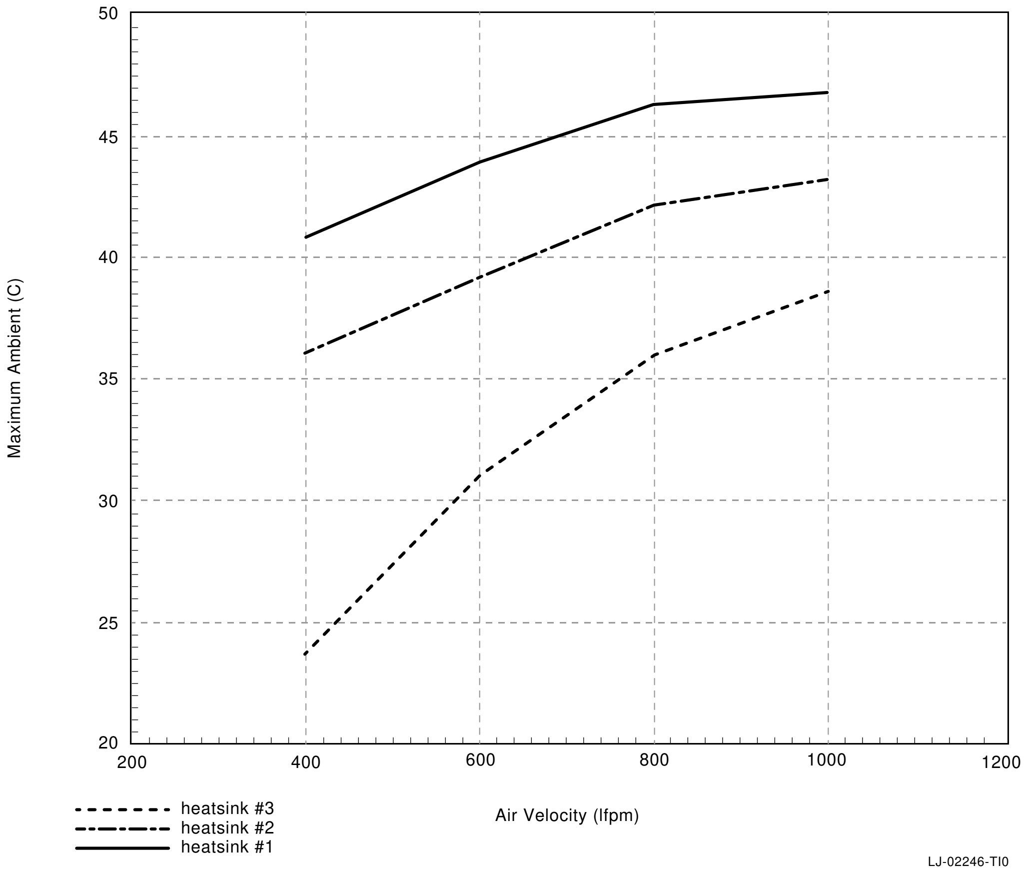


Figure 8–5 Heat Sink Maximum Ambient Temperature



Observing Figure 8–4 and Figure 8–5, it can be seen that as the air velocity increases, the benefit of the primary heat sink (heat sink number 1) over the other two heat sinks is reduced.

As shown in Figure 8–4, the difference in the θ ja value between the primary heat sink and heat sink number 3 is about 0.75 C/W at 400 lfpm, which reduces to 0.4 C/W at 1000 lfpm.

From these figures, it can be seen that the primary heat sink design is the most effective at low air velocity. It is useful for desktop applications where air flow is low but space is usually available. Heat sink number 2 and heat sink number 3 are more useful for applications where space can be a problem but air flow can be much higher.

More heat sink options are available at high air velocity. High air velocity systems not only allow more heat sink options, but they can allow higher system ambient temperature as well.

High air velocity systems will allow more trade-offs in the:

- Heat sink design
- Printed circuit board spacing
- Maximum ambient temperature

8.3.4 Device Thermal Characteristics in Forced Air Without Heat Sink

As a reference point, without heat sink, θ ja was measured at 3.35 C/W at 1000 lfpm air velocity. This would require a maximum ambient temperature of 8° C (46.4° F) to cool the 23 watt device, which clearly indicates why a heat sink is required.

8.4 Critical Parameters of Thermal Design

As the adequate cooling of the 21064/21064A is essential, sufficient attention must be given to the system thermal design. The critical parameters of the system thermal design and verification are listed next.

- Print circuit board component placement:
  - Orient the 21064/21064A on the printed circuit board (PCB) with the heat sink fins aligned with the air flow direction.
  - Avoid preheating ambient air. Place the 21064 /21064A on the PCB so that inlet air is not preheated by any other PCB components.
  - Do not place other high power devices in the vicinity of the 21064 /21064A.
  - Do not restrict the air flow across the 21064/21064A heat sink. Placement of other devices must allow for maximum system air flow in order to maximize the performance of the heat sink.
- System verification test:

  All the thermal verification data provided in this section are based on very controlled environment.

  Note

  System verification tests are highly recommended.

  There could be some secondary heat losses to the PCB and the surrounding components which could vary from system to system. The effect of the secondary heat losses is usually small. The thermal resistance numbers should be verified in the system. The following items should be measured in the system to predict more accurate device junction temperature.

  - The local air velocity should be measured in the vicinity of the 21064 /21064A. The local air velocity at the heat sink in the system might be different from the bulk system air velocity.
  - The temperature at the center of the heat sink pedestal should be measured in the actual system environment. The junction temperature should be calculated by adding junction to heat sink temperature rise from Table 8–1 or Table 8–4. This will provide more accurate junction temperature estimates for the given system. The data provided in Table 8–1 or Table 8–4 should be used as reference only.

9
Signal Integrity

9.1 Introduction

This chapter describes the signal integrity issues that should be considered by a designer when using the 21064/21064A.

Note

The **eclOut_h** signal should be connected to Vss, and **vRef** should be connected to 1.4 V.

Note

SPICE simulation models are available for 21064/21064A I/Os.

This chapter is organized as follows:

- Power Supply Considerations
- I/O Drivers
- Input Clock
- Voltage/Current (VI) Characteristics Curves and Edge Rate Curves
- References

9.2 Power Supply Considerations

For correct operation of the 21064/21064A, all of the Vss pins must be connected to ground and all of the Vdd pins must be connected to a 3.3 V ±5% power source. This source voltage should be guaranteed (even under transient conditions) at the 21064/21064A pins, and not just at the printed circuit board (PCB) edge.

Plus 5 V is not used in the 21064/21064A. The voltage difference between the Vdd pins and Vss pins must never be greater than 3.6 V. See Section 7.2.

9.2.1 Decoupling

Adequate power supply decoupling capacitance is required on the PCB to supply the 21064/21064A's transient currents. The total capacitance should be no less than 2uF. Many small, valued, surface-mount capacitors should be used. These capacitors should be physically placed as close to the 21064 /21064A package power pins as possible.

Note

It is recommended that 20 ceramic 0.1uF surface-mount capacitors be placed on the PCB in the open area of the PGA pin field, under the PGA itself.

Use capacitors that are as physically small as possible. Connect the capacitors directly to the 21064/21064A Vdd and Vss pins (or to their own down by way of the power and ground plane) by short (0.64 cm (0.25 in) or less) surface etch. The small capacitors generally have better electrical characteristics than the larger units, and will more readily fit close to the PGA pin field.

9.2.2 Reference Voltage (vRef)

Most input and I/O pins use the voltage on the **vRef** pin circuit to set the input receiver threshold voltage level. The following pins are exceptions:

- **clkIn_h** and **clkIn_l**
- **testclkIn_h** and **testclkIn_l**
- **tagOk_h** and **tagOk_l**[1]
- **dcOk_h**
- **eclOut_h**
- **tristate_l**
- **cont_l**

For correct operation of the input buffers, a 1.4 V (+/- 10%) reference voltage must be connected to the **vRef** pin.

---

[1] In the **21064A**, **tagOk_h** and **tagOk_l** are referenced to **vRef** and may be driven to a 5 V nominal level by external logic.

The impedance of the **vRef** voltage source is not critical to **vRef** signal integrity because it is filtered on the 21064/21064A. Also see Section 7.4.1.

A voltage divider made from a 150 Ohm resistor to the Vdd supply, and a 110 Ohm resistor to Vss will produce a good low impedance source for **vRef**.

9.2.3 Power Supply Sequencing

Although the 21064/21064A uses a 3.3 V (nominal) power source, most of the other logic on the printed circuit board probably requires a 5 V power supply. These 5 V devices can damage the 21064/21064A's I/O circuits if the 5 V power source powering the PCB logic and the Vdd supply feeding the 21064/21064A are not sequenced correctly.

Caution

To avoid damaging the 21064/21064A's I/O circuits, the I/O pin voltages must not exceed 4 V until the Vdd supply is at least 3 V or greater.

This rule can be satisfied if the Vdd and the 5 V supplies come up together, or if the Vdd supply comes up before the 5 V supply is asserted. Bringing the lower voltage up before the higher voltage is the opposite of the way that CMOS systems with multiple power supplies of different voltages are usually sequenced, but it is required for the 21064/21064A.

A three-terminal voltage regulator can be used to make 3.3 V Vdd from the 5 V supply, provided the output of the regulator (Vdd) tracks the 5 V supply with only a small offset. The requirement is that when the 5 V supply reaches 4 V, Vdd must be 3 V or higher. While the 5 V supply is below 4 V, Vdd can be less than 3 V.

All 5 V sources on the 21064/21064A's I/O pins should be disabled if the power supply sequencing is such that the 5 V supply will exceed 4 V before the Vdd is at least 3 V. The 5 V sources should remain disabled until the Vdd power supply is equal to or greater than 3 V.

Disabling all 5 V sources can be very difficult because there are so many possible sneak paths. Inputs, for example, on bipolar TTL logic can be a source of current, and will put a voltage across a 21064/21064A I/O pin high enough to violate the (no higher than 4 V until there is 3 V) rule. TTL outputs are specified to drive a logic one to at least 2.4 V, but usually drive voltages much higher. CMOS logic and CMOS SRAMs usually drive "full rail" signals that match the value of the 5 V power supply.

Another concern is parallel (DC) terminations or pullups connected between the 21064/21064A and the 5 V supply. The Vdd supply should be used to power parallel terminations. This is one reason that the **vRef** generation circuit should be connected to the Vdd supply and not the 5 V supply.

Disabling the 5 V outputs of PCB logic is generally possible, but raises the PCB complexity and can reduce system performance by increasing critical path timing. If the 5 V logic device has an enable pin, circuits (such as power supply supervisor chips) on the PCB can monitor the Vdd and 5 V supplies. When the supervision circuit detects that 5 V is increasing from zero while the Vdd supply is below 3V, the power supply supervisor circuit produces a disable signal to force all PCB logic with 5 V outputs into the high impedance state. This technique won't prevent bipolar TTL inputs from acting as a 5 V source, but it can be used to disable sources such as cache RAM outputs.

9.3 I/O Drivers

This section describes the 21064/21064A I/O pins.

9.3.1 I/O Driver Pins

All I/O pins, and most input-only pins, are 5 V tolerant. This means that once Vdd is equal to or exceeds 3 V, logic signals from 5 V logic can be received safely, even if the signals exceed Vdd. The input-only pins that can not be exposed to voltages greater than Vdd are:

- **tagOk_h** and **tagOk_l**[1]
- **testClkIn_h** and **testClkIn_l**
- **clkIn_h** and **clkIn_l**
- **dcOk_h**
- **tristate_l**
- **cont_l**
- **eclOut_h**

[1] In the **21064A**, **tagOk_h** and **tagOk_l** are referenced to **vRef**.

9.3.1.1 Maximum Received Voltage Levels

The voltage appearing on any of the 21064/21064A 5 V tolerant I/O pins must not exceed 4 V until the Vdd supply voltage exceeds 3 volts. I/O pin voltages can be allowed to reach 5.5 V (DC) once the 21064/21064A power supply reaches 3 V or greater. Transients (due to ringing) up to 6.5 V are permitted for periods less than 10% of the driven waveforms total period.

I/O pins that are not 5 V tolerant are designed to connect to a 3 V signal. These pins must not be exposed to DC values greater than 3.6 V, or transients higher than 4.5 V. Transients between 3.6 V and 4.5 V are permitted, but must be less than 10% of the driven waveforms total period.

9.3.1.2 Clamping Action of I/Os

The normal parasitic diode to Vdd typically present on CMOS outputs is not present in the 21064/21064A. The printed circuit board designer should not rely on the 21064/21064A I/O pins to clamp high going signal ringing or overshoots to the Vdd rail. There is a parasitic diode between the output and Vss, so low going overshoots below the Vss rail will be clamped to about -500 mV.

9.3.1.3 Pin Capacitances

Each 21064/21064A I/O pin can be modeled as a lumped 10 pF capacitor load in series with a 30 nH inductor. This does not apply to clock input pins.

9.3.2 I/O Driver Characteristics

The driver characteristics of I/O pins is described in this section.

9.3.2.1 Voltage/Current (VI) Curves

Figure 9–1 and Figure 9–2 show typical high and low level output characteristics of the 21064/21064A I/O pins. Figure 9–1 shows the characteristics for a typical 21064/21064A I/O pullup, while Figure 9–2 shows the pulldown. Under no load conditions the pullup pulls the pins to the Vdd rail, and the pulldown pulls to the Vss rail. The VI curves can be used to predict the output levels when the I/O pins are under DC load (DC noise margins). They can be used graphically to perform a load line analysis by use of ladder diagrams or Bergeron diagrams. See Section 9.6 for additional information on these diagrams.

Positive current flow is assumed in both graphs: Figure 9–1 shows the sourcing ability of a 21064/21064A I/O pin, while Figure 9–2 shows the I/O pins sinking ability.

Individual I/O pins are able to source and sink very high currents, but steady state pin currents should not exceed those shown in Section 7.2.

Figure 9–1 High Level Output Voltage versus High Level Output Current

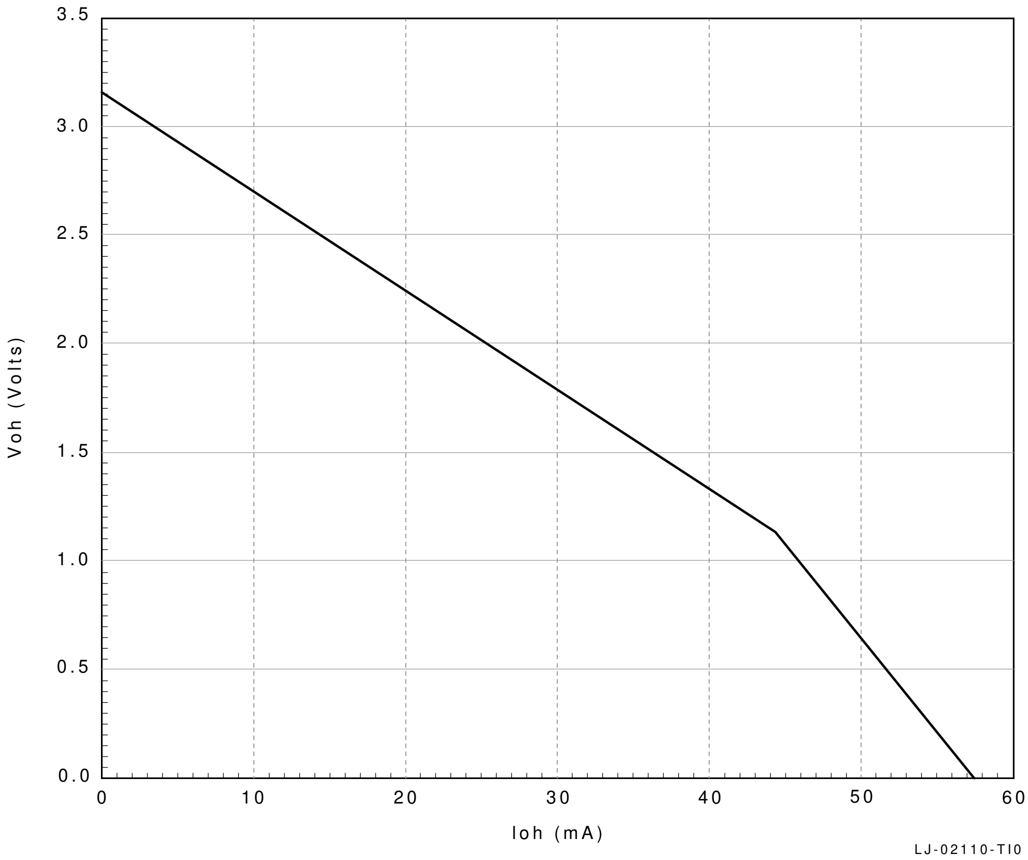
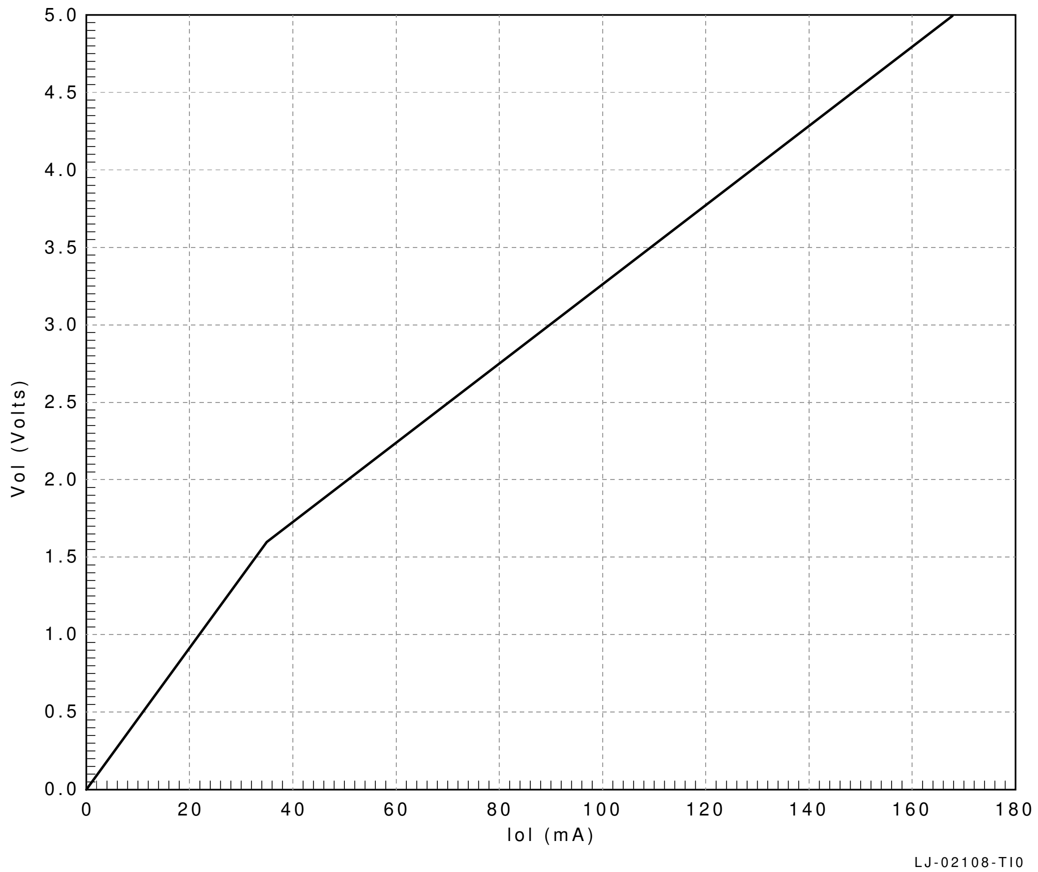


Figure 9–2 Low Level Output Voltage versus Low Level Output Current



9.3.2.2 Switching Characteristics

It is important for a designer to know the 21064/21064A output edge rate characteristics because they are used to decide if a driven line is electrically long. Electrically long lines will behave as transmission lines and can no longer be analyzed as lumped elements. Reflections will travel up and down a transmission line, causing signals to overshoot the Vdd and Vss rails, and to undershoot below logic threshold levels. The severity of the reflections depends on the degree of mismatch between the impedance of the transmission line and the impedances at the source and at the load. Bergeron diagrams (Section 9.6) or lattice diagrams (Section 9.6) can be used to predict the severity of the reflections. The need for terminations can be determined from this analysis.

A line is electrically long when its electrical length (Tpd) is greater than one tenth of the rise time of the edge sent down the line. This rule is overly conservative if the source impedance is closely matched to the line impedance. The impedance of the 21064/21064A I/O pads is process dependent, but will be approximately 40 Ohms.

Figure 9–3 Edge Rate versus Load

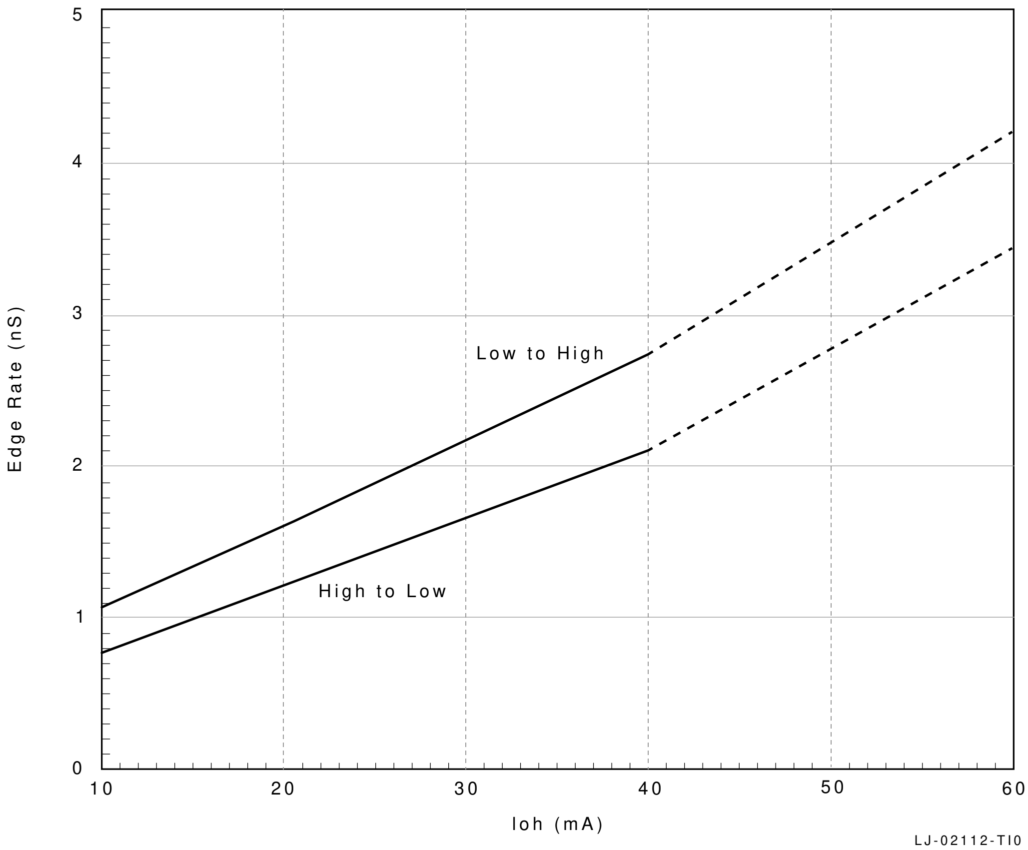


Figure 9–3 is a plot of the typical fastest edge rate (measured from the 10% to 90% points of the signal swing) from the 21064/21064A against lumped load. Both the pullup (labeled ''Low to High'') and the pulldown (labeled ''High to Low'') characteristics are shown.

Lumped loads in excess of 40 pF should not be driven by the 21064/21064A, but are shown in Figure 9–3 for completeness.

9.4 Input Clock

A differential system clock is required to run the 21064/21064A. The system clock is connected to the clock input pins **clkIn_h** and **clkIn_l**. These pins are self-biasing, and can be capacitively coupled to the clock source on the PCB or they can be directly driven. The oscillator must have a duty cycle of 55%/45% or tighter.

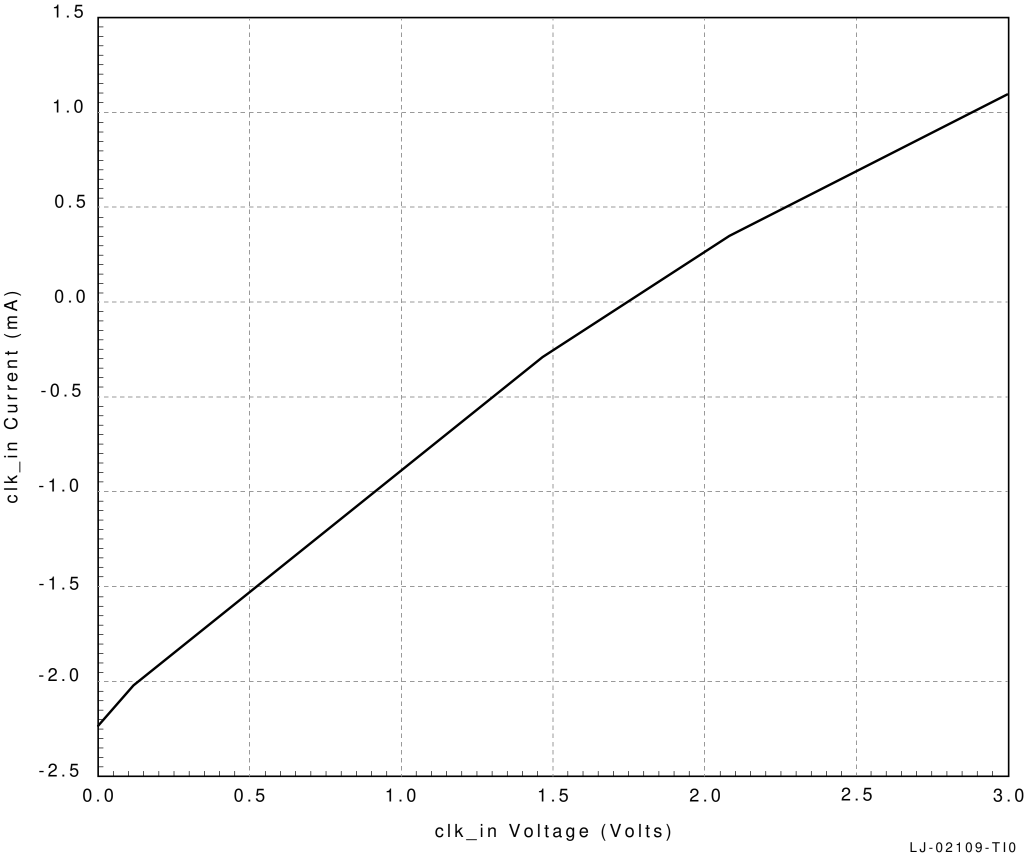
9.4.1 Clock Termination and Impedance Levels

The clock input pins appear as a 50 Ohm series termination resistor connected to a high impedance voltage source. The voltage source produces a nominal voltage value of Vdd/2. The source has an impedance of a few thousand Ohms. This voltage is called the self-bias voltage and sources current when the applied voltage at the clock input pins is less than the self-bias voltage. It sinks current when the applied voltage exceeds the self-bias voltage.

Figure 9–4 shows the input current requirements for the clock inputs (**clkIn_h** and **clkIn_l**). Negative currents indicate that the **clkIn_h** and **clkIn_l** pins are sourcing positive currents into the clock pins.

Very little current is required for small signal swings near the self-bias point, but as the applied voltage swing increases, the input current requirements increase.

Figure 9–4 Clock Current versus Clock Voltage



9.4.1.1 AC Coupling

Using series coupling (blocking) capacitors makes the 21064/21064A clock input pins insensitive to the oscillators DC level. When connected this way, oscillators with any DC offset relative to Vss can be used provided they can drive a signal into the **clkIn_h** and **clkIn_l** pins with a peak-to-peak level of at least 600 mV, but no greater than 3.0 V peak-to-peak.

The value of the coupling capacitor is not overly critical. However, it should be sufficiently low impedance at the clock frequency so that the oscillators output signal (when measured at the **clkIn_h** and **clkIn_l** pins) is not attenuated below the 600 mV peak-to-peak lower limit. For sine waves or oscillators producing nearly sinusoidal (pseudo square wave) outputs, 220 pF is recommended at 250 MHz. A high quality dielectric such as NPO is required to avoid dielectric losses.

Figure 9–4 can be used to determine the oscillators output requirements when the oscillator is ac coupled. The capacitor will center the clock signal around the clock inputs self-bias point, so oscillators that produce a small swing will not have to drive much current into the pins. The self-bias point can be found from Figure 9–4 by noting where the pin current is zero.

9.4.1.2 DC Coupling

If the clock is direct coupled (the blocking capacitor not used) it must provide a swing above and below the self-biasing point by at least 300 mV (for a 600 mV peak-to-peak signal).

Caution

Verify that the clock inputs are not driven below Vss or above Vdd.

If the oscillator output swings from Vss to Vdd, it must be capable of sourcing over 1 mA and sinking nearly 2.5 mA.

9.5 Voltage/Current (VI) Characteristics Curves and Edge Rate Curves

This section has examples of using the VI characteristic curves and the edge rate curves described in Section 9.3.2.

9.5.1 VI and Edge Rate Curves—Example One

Assume that a 21064/21064A I/O pin is driving a 10 pF load. This load is connected to the I/O pin by an etch that has zero length. To determine what the driven signal is like:

1. Determine if the total driven load is greater than the 40 pF suggested upper bound for a 21064/21064A pin. In this example, the total load is 10 pF, well below the suggested 40 pF limit. If the load had been connected to the pin by an etch with a length greater than zero, then the capacitance of the etch would also have to be included. In this example there is no etch capacitance to be concerned with.
2. Check to see if the load is connected to the pin by transmission lines. Transmission line behavior occurs when the line is electrically long. The onset of long line behavior is often estimated for calculation purposes by assuming that it occurs when the one-way electrical length of the line is one tenth or more longer than the rise time (edge rate) of the signal on the line. If a line is long, a load line analysis can be used to determine the voltages on the line and at the loads.

   Figure 9–1 and Figure 9–2 can be used in that analysis. If the line is not long, the load voltages will be the voltage produced by the driver as it charges the load capacitances. The rate of this charging can be determined from Figure 9–3.

In this example, the etch length between the load and the pin is zero, so no transmission line behavior exists. By examining Figure 9–3, you can see that the high-to-low transition will take about 800 ps, and the low-to-high about 1.1 ns.

9.5.2 VI and Edge Rate Curves—Example Two

Connect the 10 pF load to the pin using an etch of one inch in length. Assume that the etch is an embedded microstrip with:

- Z0 (characteristic impedance) of 64 Ohms
- C0 (capacitance per unit length) of 2.5 pF/inch
- A one way propagation delay (Tpd) of 160 ps/inch

To determine how the driven signal from the 21064/21064A will behave:

1. Ensure that the total capacitance is the sum of the 10 pF load and the etch capacitance (1 inch of etch, or 2.5 pF). In this case, the total capacitance is 12.5 pF. The 12.5 pF is below the 40 pF suggested limit, making this configuration acceptable.
2. Check to see if the etch is long enough to act as a transmission line.

   Figure 9–3 shows that the 21064/21064A will move a 12.5 pF load in about 850 ps on a high going edge. Use the high-to-low curve because that transition is sharper than the low-to-high.
3. Apply the one tenth rise time rule to the 850 ps edge. Conclude that under these conditions, lines longer than 85 ps (or about half an inch at 160 ps/inch) will be long. The one-inch etch is longer than the half-inch length used to gauge long line behavior for this load. It can be assumed that the line is indeed long and will act as a transmission line.

At this point circuit simulation can be performed to obtain the precise behavior, or a load line analysis can be used to determine the voltage levels initially transmitted down the line.

9.5.3 VI and Edge Rate Curves—Example Three

Assume that a line has been determined to be long. Determine the magnitude of the voltage launched down the line on low-to-high transition.

Because the line is long, the impedance of the 21064/21064A output pin and the impedance of the line will act as a voltage divider to the wave launched down the line. All practical lines will have an impedance on the same order of magnitude as the output impedance of the 21064/21064A output pin. The driven voltage level will initially be less than the value of Vdd. This reduced level will create a plateau voltage at the pin output (and will be sensed by any logic devices connected to the pin at this point) until modified by the reflections returned from the load located at the far end of the line. If the initial plateau voltage at the 21064/21064A pin (called the near end) is less than a valid logic level, any devices connected there will have to wait until enough reflections have been returned to cause the near end voltage to exceed the required logic level. This can lead to a situation where the loads at the far end of the line will switch before the loads at the near end (those closer to the 21064/21064A I/O pin).

A simple load-line analysis can be made without much effort to determine the magnitude of the first plateau voltage. Perform one of the following graphical representation methods if the effects of reflections are to be investigated:

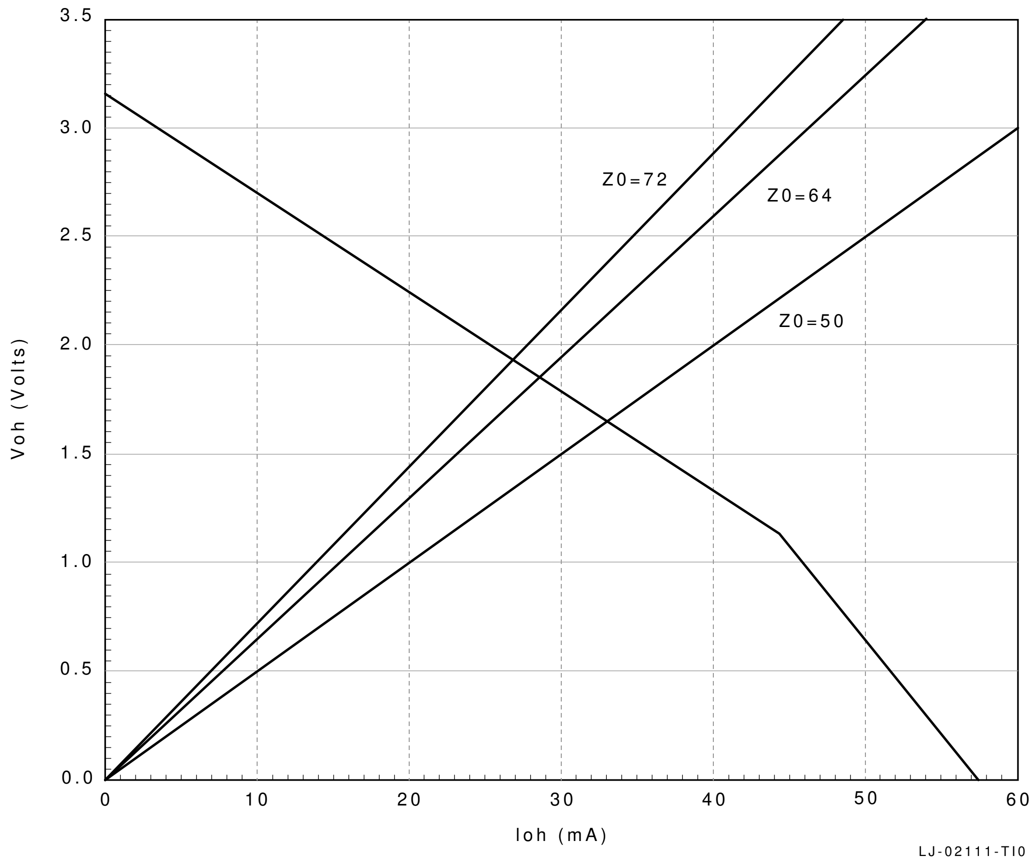
- Ladder diagrams
- Bergeron diagrams
- Circuit simulation

These methods are briefly described in Section 9.5.4.

To determine the magnitude of the near-end plateau, a load line is superimposed (drawn) on the I/O pins VI characteristic. Figure 9–5 shows load lines for 50, 64 and 72 Ohm impedances superimposed on the VI curve shown in Figure 9–1. The operating point for the I/O driver/transmission line system occurs where the load line crosses the VI curve. For example, if the 21064/21064A is connected to a 64 Ohm etch as in Example 3, the first plateau will occur at about 1.8 V (and the pin will be sourcing about 28 mA at that point).

Constructing the load line is easy. Pick a pair of voltages and compute the currents that the impedance will draw at those voltages. Then plot the points on the VI curve, and draw a line between them. For simplicity, zero volts can be used for the second point.

Figure 9–5 Low to High Load Line Analysis



9.5.4 Graphical Representation Methods

Ladder diagrams are sometimes called "line charts" or "reflection charts." Ladder diagrams are a convenient way to record the voltage steps along a transmission line created by reflections due to impedance mismatches. They require the user to know the value for the reflection coefficient of the source and the load. Ladder diagrams have the advantage over Bergeron diagrams at being able to predict voltage levels at points anywhere along a transmission line, including points other than the source and load. Use 40 Ohms for the source impedance of the 21064/21064A when computing the source end reflection coefficient.

Ladder diagrams are not as useful in situations where the source or load impedances are changing or are non-linear. This is often the case when working with CMOS. In situations where the load or driver impedance is non-linear, Bergeron diagrams can be used to determine voltage levels due to ringing and overshoots at either end of the line. Figure 9–1 and Figure 9–2 can be used for the 21064/21064A's output characteristics when plotting the load lines on a Bergeron plot. The inputs (except for the clock inputs) can be assumed to have an impedance of 175 Ohms.

If a number of layout scenarios are to be examined, circuit simulation should be used rather than performing a ladder or Bergeron analysis. Accuracy will improve over the Bergeron analysis, but the real advantage is speed. A circuit simulation can save a great deal of time for the designer who is interested in examining the differences between different layout topologies, including the response of networks that have stubs or complex signal treeing.

9.6 References

Additional information on Bergeron diagrams and ladder diagrams can be found in the following documents:

1. *Lines, Waves and Antennas* (Brown et al., Copyright 1973 Wiley & Sons)
2. *Fairchild ECL Data Book* (Copyright 1977, Fairchild)
3. *Motorola MECL System Designers Handbook* (Copyright 1988, Motorola)
4. *The ALS/AS Logic Data Book* (Copyright 1986 Texas Instruments)

10

Mechanical Data and Packaging Information

10.1 Introduction

This chapter provides detailed information on the chip package and the complete pinout for the 21064/21064A.

10.2 Package Information

Package information for both the **21064** and the **21064A** are included in this section.

10.2.1 21064 Package Information

Figure 10–1 shows the **21064** package physical dimensions without heat sink and Figure 10–3 shows the PGA locations.

10.2.2 21064A Package Information

Figure 10–2 shows the **21064A** package physical dimensions without heat sink and Figure 10–3 shows PGA locations. The PGA locations are identical for the **21064** and the **21064A**.

Figure 10–1 21064 Package Dimensions

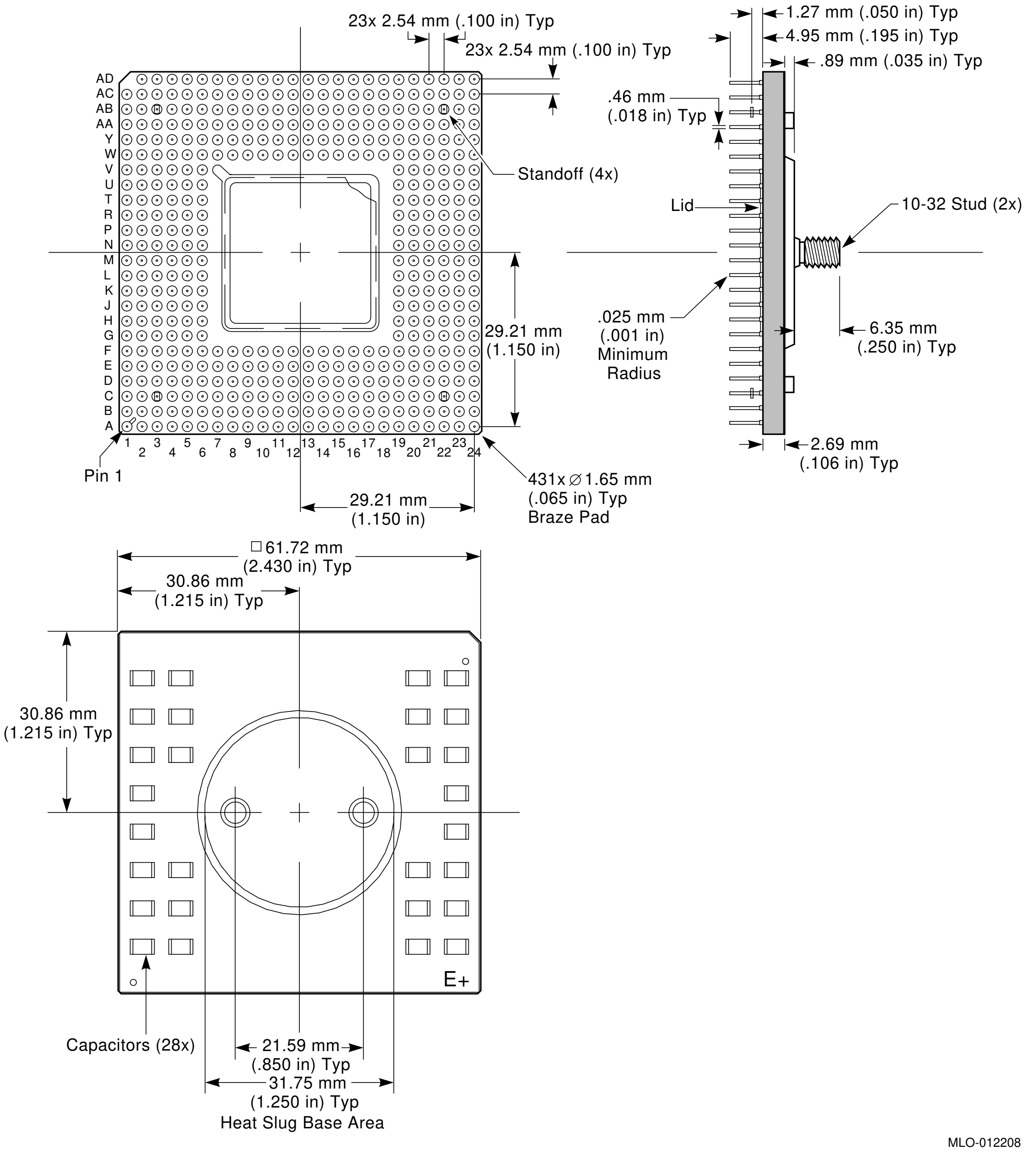


Figure 10–2 21064A Package Dimensions

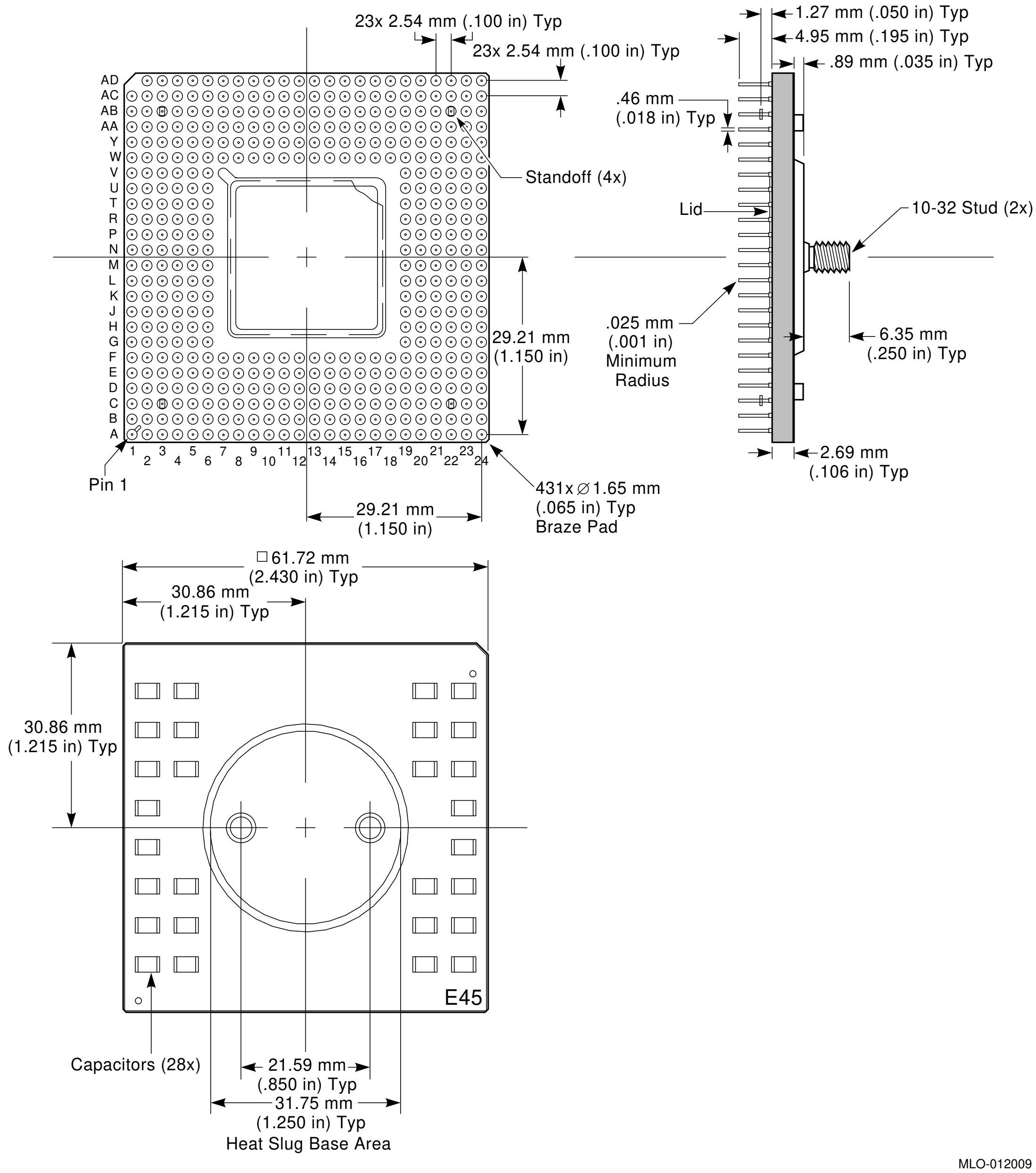
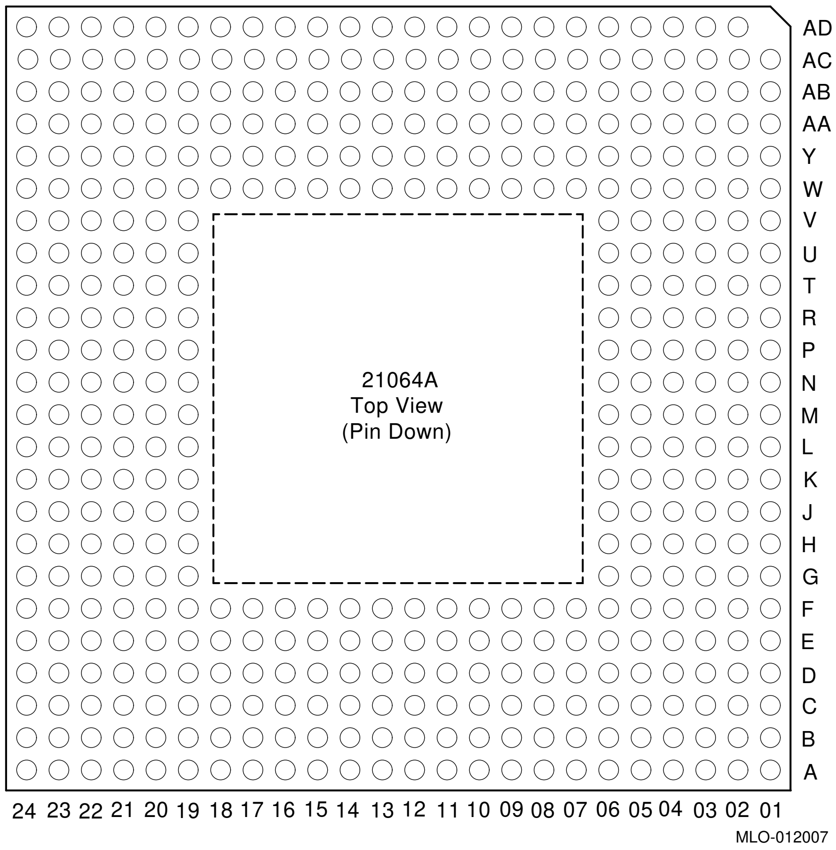


Figure 10–3 21064A PGA Cavity Down View



10.3 21064/21064A Signal Pin Lists

The **21064** and **21064A** pin lists are identical except for the nine pins listed in Table 10–1. These differences are also identified where they occur in all tables in this section of the manual.

Table 10–1 21064 and 21064A Pin List Differences

21064A Name	21064 Name	Type	PGA Location
icMode_h 2[1]	spare 1	I	AD7
dInvReq_h 1[1]	spare 3	I	C24
dInvReq_h 0	dInvReq_h	I	AD9
resetSClk_h[1]	spare 6	I	AA11
sysClkDiv_h[1]	spare 8	I	AA16
dMapWE_h 1[1]	spare 0	O	M24
dMapWE_h 0	dMapWE_h	O	L24
lockWE_h	tagEq_l	O	P24
lockFlag_h	tagAdr_h 17	I	R23

[1]Has internal pulldown drawing a maximum current of 200 uA at 2.4V dc

Table 10–2 through Table 10–17 contain the pin list in functional groups.

The key for the signal type is listed here.

- B = Bidirectional
- I = Input
- N = Not connected
- P = Power or ground
- O = Output

Table 10–2 Data Pin List (Type B)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
data_h 127	A23	data_h 126	C21	data_h 125	B21
data_h 124	C20	data_h 123	D19	data_h 122	A18
data_h 121	C17	data_h 120	A17	data_h 119	C16
data_h 118	D15	data_h 117	E14	data_h 116	C14
data_h 115	E13	data_h 114	C13	data_h 113	A13
data_h 112	C12	data_h 111	E12	data_h 110	B11
data_h 109	A10	data_h 108	D10	data_h 107	B9
data_h 106	D9	data_h 105	C8	data_h 104	A7
data_h 103	C7	data_h 102	D6	data_h 101	B5
data_h 100	A4	data_h 99	C4	data_h 98	A3
data_h 97	A2	data_h 96	C3	data_h 95	F4
data_h 94	D1	data_h 93	F3	data_h 92	F1
data_h 91	G3	data_h 90	J4	data_h 89	J1
data_h 88	K3	data_h 87	K1	data_h 86	L4
data_h 85	M4	data_h 84	M2	data_h 83	N1
data_h 82	N4	data_h 81	P1	data_h 80	P3
data_h 79	P5	data_h 78	R3	data_h 77	T3
data_h 76	U1	data_h 75	U4	data_h 74	V2
data_h 73	V4	data_h 72	W3	data_h 71	Y2
data_h 70	AB1	data_h 69	AB2	data_h 68	Y4
data_h 67	AB3	data_h 66	AA4	data_h 65	AC4
data_h 64	AB5	data_h 63	D20	data_h 62	A22
data_h 61	A21	data_h 60	A20	data_h 59	C19
data_h 58	D17	data_h 57	B17	data_h 56	D16
data_h 55	A16	data_h 54	C15	data_h 53	D14
data_h 52	A14	data_h 51	D13	data_h 50	B13
data_h 49	A12	data_h 48	D12	data_h 47	A11
data_h 46	C11	data_h 45	C10	data_h 44	A9

(continued on next page)

Table 10–2 (Cont.) Data Pin List (Type B)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
data_h 43	C9	data_h 42	A8	data_h 41	D8
data_h 40	B7	data_h 39	D7	data_h 38	A5
data_h 37	C5	data_h 36	D5	data_h 35	B3
data_h 34	D4	data_h 33	A1	data_h 32	E4
data_h 31	E3	data_h 30	E1	data_h 29	F2
data_h 28	G4	data_h 27	G1	data_h 26	J3
data_h 25	K4	data_h 24	K2	data_h 23	L5
data_h 22	L3	data_h 21	M3	data_h 20	M1
data_h 19	N3	data_h 18	N5	data_h 17	P2
data_h 16	P4	data_h 15	R1	data_h 14	R4
data_h 13	T4	data_h 12	U3	data_h 11	V1
data_h 10	V3	data_h 9	W1	data_h 8	Y1
data_h 7	Y3	data_h 6	AC1	data_h 5	AA3
data_h 4	AD2	data_h 3	AD3	data_h 2	AB4
data_h 1	AD4	data_h 0	AA5	–	–

Table 10–3 Address Pin List (Type B)

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
adr_h 33	AD17	adr_h 32	AB17	adr_h 31	AA17
adr_h 30	AD18	adr_h 29	AC18	adr_h 28	AB18
adr_h 27	AA18	adr_h 26	AD19	adr_h 25	AB19
adr_h 24	AA19	adr_h 23	AD20	adr_h 22	AC20
adr_h 21	AB20	adr_h 20	AD21	adr_h 19	AD22
adr_h 18	AB21	adr_h 17	AA20	adr_h 16	AC22
adr_h 15	AA21	adr_h 14	AB22	adr_h 13	AD23
adr_h 12	AD24	adr_h 11	AA22	adr_h 10	AC24
adr_h 9	AB24	adr_h 8	Y21	adr_h 7	AA23

(continued on next page)

Table 10–3 (Cont.) Address Pin List (Type B)

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
adr_h 6	AA24	adr_h 5	Y22	–	–

Table 10–4 Parity/ECC Bus Pin List (Type B)

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
check_h 27	A6	check_h 26	B15	check_h 25	D18
check_h 24	D11	check_h 23	C22	check_h 22	D21
check_h 21	B19	check_h 20	AA1	check_h 19	L1
check_h 18	H2	check_h 17	T1	check_h 16	C1
check_h 15	B1	check_h 14	H4	check_h 13	C6
check_h 12	A15	check_h 11	C18	check_h 10	E11
check_h 9	A24	check_h 8	B24	check_h 7	A19
check_h 6	W4	check_h 5	M5	check_h 4	H1
check_h 3	T2	check_h 2	D2	check_h 1	D3
check_h 0	H3	–	–	–	–

Table 10–5 Primary Cache Invalidate Pin List (Type I)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
iAdr_h 12	AB7	iAdr_h 11	AC8	iAdr_h 10	AA7
iAdr_h 9	AD6	IAdr_h 8	AC6	iAdr_h 7	AB6
iAdr_h 6	AA6	iAdr_h 5	AD5	–	–
dInvReq_h[1]	AD9	dInvReq_h 1[2]	C24	–	–

[1]dInvReq_h for **21064**—dInvReq_h 0 for **21064A**

[2]**21064A** only—spare 3 on **21064**

Table 10–6 External Cache Control Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
tagCEOE_h	N24	O	tagCtlWE_h	H22	O
tagCtlV_h	R24	B	tagCtlD_h	P21	B
tagCtlS_h	P20	B	tagCtlP_h	P22	B
tagadr_h 33	Y24	I	tagadr_h 32	W22	I
tagadr_h 31	W23	I	tagadr_h 30	W24	I
tagadr_h 29	V21	I	tagadr_h 28	V22	I
tagadr_h 27	V24	I	tagadr_h 26	U21	I
tagadr_h 25	U22	I	tagadr_h 24	U23	I
tagadr_h 23	U24	I	tagadr_h 22	T21	I
tagadr_h 21	T22	I	tagadr_h 20	T24	I
tagadr_h 19	R21	I	tagadr_h 18	R22	I
tagadr_h 17[1]	R23	I	tagadrP_h	W21	I
tagOk_h	N21	I	tagOk_l	N20	I
tagEq_l[2]	P24	O	–	–	–
dataCEOE_h 3	H21	O	dataCEOE_h 1	G23	O
dataCEOE_h 2	G24	O	dataCEOE_h 0	G22	O
dataWE_h 3	L23	O	dataWE_h 1	L21	O
dataWE_h 2	L22	O	dataWE_h 0	L20	O
dataA_h 4	N22	O	dataA_h 3	N23	O

[1] **21064** only—used for lockFlag_h input on 21064A

[2] **21064** only—used for lockWE_h output on 21064A

(continued on next page)

Table 10–6 (Cont.) External Cache Control Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
holdReq_h	F24	I	holdAck_h	G21	O
dMapWE_h[3]	L24	O	dOE_l	D24	I
dMapWE_h 1[4]	M24	O	–	–	–
dWSel_h 1	E23	I	dWSel_h 0	E22	I
dRAck_h 2	D22	I	dRAck_h 0	C23	I
dRAck_h 1	E21	I	–	–	–
cReq_h 2	M22	O	cReq_h 0	M20	O
cReq_h 1	M21	O	–	–	–
cWMask_h 7	K24	O	cWMask_h 6	K22	O
cWMask_h 5	K21	O	cWMask_h 4	J24	O
cWMask_h 3	J23	O	cWMask_h 2	J22	O
cWMask_h 1	J21	O	cWMask_h 0	H24	O
cAck_h 2	F22	I	cAck_h 0	E24	I
cAck_h 1	F21	I	–	–	–

[3] **21064** name—named dMapWE_h 0 on 21064A

[4] **21064A** only—spare 0 on **21064**

Table 10–7 Interrupts Pin List (Type I)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
irq_h 5	AA15	irq_h 4	AB15	irq_h 3	AD15
irq_h 2	AC14	irq_h 1	AD14	irq_h 0	AD13

Table 10–8 Instruction Cache Initialization Pin List (Type I)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
icMode_h 2[1]	AD7	icMode_h 1	AD12	icMode_h 0	AB14

[1] **21064A** only—spare 1 on **21064**

Table 10–9 Serial ROM Interface Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
sRomOE_l	AB10	O	sRomD_h	AB9	I
sRomClk_h	AC10	O	–	–	–

Table 10–10 Initialization Pin List (Type I)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
dcOk_h	AB12	reset_l	AB8	reset_SClk_h[1]	AA11

[1] **21064A** only—spare 6 on **21064**

Table 10–11 21064 Clock Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
clkIn_h	W12	I	clkIn_l	W13	I
testClkIn_h	W9	I	testClkIn_l	W10	I
cpuClkOut_h	AB11	O	sysClkDiv_h[1]	AA11	I
sysClkOut1_h	AA12	O	sysClkOut1_l	AA13	O
sysClkOut2_h	AA9	O	sysClkOut2_l	AA10	O

[1]**21064A** only—spare 8 **21064**

Table 10–12 21064A Load/Lock and Store/Conditional Fast Lock Mode

Signal	PGA Location	Type	Signal	PGA Location	Type
lockFlag_h[1]	R23	O	lockWE_h[2]	P24	O

[1]**21064A** only—tagEq_l on **21064**

[2]**21064A** only—tagAdr_h 17 on **21064**

Table 10–13 Performance Monitoring Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
perf_cnt_h 1	AC16	I	perf_cnt_h 0	AB16	I

Table 10–14 Other Signals Pin List

Signal Name	PGA Location	Type	Signal Name	PGA Location	Type
triState_l	AB13	I	vRef	AA8	I
cont_l	AA14	I	eclOut_h	AD8	I

Table 10–15 Power Pin List (Type P)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
Vdd plane	B2	Vdd plane	N2	Vdd plane	B6
Vdd plane	N19	Vdd plane	B10	Vdd plane	P6
Vdd plane	B14	Vdd plane	R5	Vdd plane	B18
Vdd plane	R19	Vdd plane	B22	Vdd plane	T6
Vdd plane	D23	Vdd plane	T20	Vdd plane	E2
Vdd plane	T23	Vdd plane	E5	Vdd plane	U2
Vdd plane	E7	Vdd plane	U5	Vdd plane	E9
Vdd plane	U19	Vdd plane	E15	Vdd plane	V6
Vdd plane	E17	Vdd plane	V20	Vdd plane	E19
Vdd plane	W5	Vdd plane	F6	Vdd plane	W7
Vdd plane	F8	Vdd plane	W11	Vdd plane	F10
Vdd plane	W15	Vdd plane	F12	Vdd plane	W17
Vdd plane	F14	Vdd plane	W19	Vdd plane	F16
Vdd plane	Y6	Vdd plane	F18	Vdd plane	Y8
Vdd plane	F20	Vdd plane	Y10	Vdd plane	G5
Vdd plane	Y12	Vdd plane	G19	Vdd plane	Y14
Vdd plane	H6	Vdd plane	Y16	Vdd plane	H20
Vdd plane	Y18	Vdd plane	H23	Vdd plane	Y20
Vdd plane	J2	Vdd plane	Y23	Vdd plane	J5
Vdd plane	AA2	Vdd plane	J19	Vdd plane	AC3
Vdd plane	K6	Vdd plane	AC7	Vdd plane	K20
Vdd plane	AC11	Vdd plane	L19	Vdd plane	AC15
Vdd plane	M6	Vdd plane	AC19	Vdd plane	M23
Vdd plane	AC23	–	–	–	–

Table 10–16 Ground Pin List

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
Vss plane	B4	Vss plane	N6	Vss plane	B8
Vss plane	P19	Vss plane	B12	Vss plane	P23
Vss plane	B16	Vss plane	R2	Vss plane	B20
Vss plane	R6	Vss plane	B23	Vss plane	R20
Vss plane	C2	Vss plane	T5	Vss plane	E6
Vss plane	T19	Vss plane	E8	Vss plane	U6
Vss plane	E10	Vss plane	U20	Vss plane	E16
Vss plane	V5	Vss plane	E18	Vss plane	V19
Vss plane	E20	Vss plane	V23	Vss plane	F5
Vss plane	W2	Vss plane	F7	Vss plane	W6
Vss plane	F9	Vss plane	W8	Vss plane	F11
Vss plane	W14	Vss plane	F13	Vss plane	W16
Vss plane	F15	Vss plane	W18	Vss plane	F17
Vss plane	W20	Vss plane	F19	Vss plane	Y5
Vss plane	F23	Vss plane	Y7	Vss plane	G2
Vss plane	Y9	Vss plane	G6	Vss plane	Y11
Vss plane	G20	Vss plane	Y13	Vss plane	H5
Vss plane	Y15	Vss plane	H19	Vss plane	Y17
Vss plane	J6	Vss plane	Y19	Vss plane	J20
Vss plane	AB23	Vss plane	K5	Vss plane	AC2
Vss plane	K19	Vss plane	AC5	Vss plane	K23
Vss plane	AC9	Vss plane	L2	Vss plane	AC13
Vss plane	L6	Vss plane	AC17	Vss plane	M19
Vss plane	AC21	–	–	–	–

Table 10–17 Spare Pin List (Type N)

Signal Name	PGA Location	Signal Name	PGA Location	Signal Name	PGA Location
spare 8[1]	AA16	spare 7	AD16	spare 6[1]	AA11
spare 5	AC12	spare 4	AD11	spare 3[1]	C24
spare 2	AD10	spare 1[1]	AD7	spare 0[1]	M24

[1] **21064** only—used for other signals on **21064A**

10.4 PGA Pin List

Table 10–18 lists the 21064/21064A pinout in two alphabetic sequences of PGA location, A to Y then AA to AD.

The key for the signal type is listed here.

- B = Bidirectional
- I = Input
- N = Not connected
- P = Power or Ground
- O = Output

Table 10–18 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
A1	B	data_h 33	A2	B	data_h 97
A3	B	data_h 98	A4	B	data_h 100
A5	B	data_h 38	A6	B	check_h 27
A7	B	data_h 104	A8	B	data_h 42
A9	B	data_h 44	A10	B	data_h 109
A11	B	data_h 47	A12	B	data_h 49
A13	B	data_h 113	A14	B	data_h 52
A15	B	check_h 12	A16	B	data_h 55
A17	B	data_h 120	A18	B	data_h 122
A19	B	check_h 7	A20	B	data_h 60
A21	B	data_h 61	A22	B	data_h 62
A23	B	data_h 127	A24	B	check_h 9
B1	B	check_h 15	B2	P	Vdd plane
B3	B	data_h 35	B4	P	Vss plane
B5	B	data_h 101	B6	P	Vdd plane
B7	B	data_h 40	B8	P	Vss plane
B9	B	data_h 107	B10	P	Vdd plane
B11	B	data_h 110	B12	P	Vss plane
B13	B	data_h 50	B14	P	Vdd plane
B15	B	check_h 26	B16	P	Vss plane
B17	B	data_h 57	B18	P	Vdd plane
B19	B	check_h 21	B20	P	Vss plane
B21	B	data_h 125	B22	P	Vdd plane
B23	P	Vss plane	B24	B	check_h 8
C1	B	check_h 16	C2	P	Vss plane
C3	B	data_h 96	C4	B	data_h 99
C5	B	data_h 37	C6	B	check_h 13
C7	B	data_h 103	C8	B	data_h 105

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
C9	B	data_h 43	C10	B	data_h 45
C11	B	data_h 46	C12	B	data_h 112
C13	B	data_h 114	C14	B	data_h 116
C15	B	data_h 54	C16	B	data_h 119
C17	B	data_h 121	C18	B	check_h 11
C19	B	data_h 59	C20	B	data_h 124
C21	B	data_h 126	C22	B	check_h 23
C23	I	dRAck_h 0	C24	N	spare 3[1]
D1	B	data_h 94	D2	B	check_h 2
D3	B	check_h 1	D4	B	data_h 34
D5	B	data_h 36	D6	B	data_h 102
D7	B	data_h 39	D8	B	data_h 41
D9	B	data_h 106	D10	B	data_h 108
D11	B	check_h 24	D12	B	data_h 48
D13	B	data_h 51	D14	B	data_h 53
D15	B	data_h 118	D16	B	data_h 56
D17	B	data_h 58	D18	B	check_h 25
D19	B	data_h 123	D20	B	data_h 63
D21	B	check_h 22	D22	I	dRAck_h 2
D23	P	Vdd plane	D24	I	dOE_l
E1	B	data_h 30	E2	P	Vdd plane
E3	B	data_h 31	E4	B	data_h 32
E5	P	Vdd plane	E6	P	Vss plane
E7	P	Vdd plane	E8	P	Vss plane
E9	P	Vdd plane	E10	P	Vss plane
E11	B	check_h 10	E12	B	data_h 111
E13	B	data_h 115	E14	B	data_h 117

[1]**spare 3** for **21064— dInvReq_h 1** for **21064A**

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
E15	P	Vdd plane	E16	P	Vss plane
E17	P	Vdd plane	E18	P	Vss plane
E19	P	Vdd plane	E20	P	Vss plane
E21	I	dRAck_h 1	E22	I	dWSel_h 0
E23	I	dWSel_h 1	E24	I	cAck_h 0
F1	B	data_h 92	F2	B	data_h 29
F3	B	data_h 93	F4	B	data_h 95
F5	P	Vss plane	F6	P	Vdd plane
F7	P	Vss plane	F8	P	Vdd plane
F9	P	Vss plane	F10	P	Vdd plane
F11	P	Vss plane	F12	P	Vdd plane
F13	P	Vss plane	F14	P	Vdd plane
F15	P	Vss plane	F16	P	Vdd plane
F17	P	Vss plane	F18	P	Vdd plane
F19	P	Vss plane	F20	P	Vdd plane
F21	I	cAck_h 1	F22	I	cAck_h 2
F23	P	Vss plane	F24	I	holdReq_h
G1	B	data_h 27	G2	P	Vss plane
G3	B	data_h 91	G4	B	data_h 28
G5	P	Vdd plane	G6	P	Vss plane
G19	P	Vdd plane	G20	P	Vss plane
G21	O	holdAck_h	G22	O	dataCEOE_h 0
G23	O	dataCEOE_h 1	G24	O	dataCEOE_h 2
H1	B	check_h 4	H2	B	check_h 18
H3	B	check_h 0	H4	B	check_h 14
H5	P	Vss plane	H6	P	Vdd plane
H19	P	Vss plane	H20	P	Vdd plane
H21	O	dataCEOE_h 3	H22	O	tagCtlWE_h

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
H23	P	Vdd plane	H24	O	cWMask_h 0
J1	B	data_h 89	J2	P	Vdd plane
J3	B	data_h 26	J4	B	data_h 90
J5	P	Vdd plane	J6	P	Vss plane
J19	P	Vss plane	J20	P	Vss plane
J21	O	cWMask_h 1	J22	O	cWMask_h 2
J23	O	cWMask_h 3	J24	O	cWMask_h 4
K1	B	data_h 87	K2	B	data_h 24
K3	B	data_h 88	K4	B	data_h 25
K5	P	Vss plane	K6	P	Vdd plane
K19	P	Vss plane	K20	P	Vdd plane
K21	O	cWMask_h 5	K22	O	cWMask_h 6
K23	P	Vss plane	K24	O	cWMask_h 7
L1	B	check_h 19	L2	P	Vss plane
L3	B	data_h 22	L4	B	data_h 86
L5	B	data_h 23	L6	P	Vss plane
L19	P	Vdd plane	L20	O	dataWE_h 0
L21	O	dataWE_h 1	L22	O	dataWE_h 2
L23	O	dataWE_h 3	L24	O	dMapWE_h[2]
M1	B	data_h 20	M2	B	data_h 84
M3	B	data_h 21	M4	B	data_h 85
M5	B	check_h 5	M6	P	Vdd plane
M19	P	Vss plane	M20	O	cReq_h 0
M21	O	cReq_h 1	M22	O	cReq_h 2
M23	P	Vdd plane	M24	N	spare 0[3]
N1	B	data_h 83	N2	P	Vdd plane
N3	B	data_h 19	N4	B	data_h 82

[2]**dMapWE_h** for **21064— dMapWE_h 0** for **21064A**

[3]**spare 0** for **21064— dMapWE_h 1** for **21064A**

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
N5	B	data_h 18	N6	P	Vss plane
N19	P	Vdd plane	N20	I	tagOk_l
N21	I	tagOk_h	N22	O	dataA_h 4
N23	O	dataA_h 3	N24	O	tagCEOE_h
P1	B	data_h 81	P2	B	data_h 17
P3	B	data_h 80	P4	B	data_h 16
P5	B	data_h 79	P6	P	Vdd plane
P19	P	Vss plane	P20	B	tagCtlS_h
P21	B	tagCtlD_h	P22	B	tagCtlP_h
P23	P	Vss plane	P24	O	tagEq_l[4]
R1	B	data_h 15	R2	P	Vss plane
R3	B	data_h 78	R4	B	data_h 14
R5	P	Vdd plane	R6	P	Vss plane
R19	P	Vdd plane	R20	P	Vss plane
R21	I	tagadr_h 19	R22	I	tagadr_h 18
R23	I	tagadr_h 17[5]	R24	B	tagCtlV_h
T1	B	check_h 17	T2	B	check_h 3
T3	B	data_h 77	T4	B	data_h 13
T5	P	Vss plane	T6	P	Vdd plane
T19	P	Vss plane	T20	P	Vdd plane
T21	I	tagadr_h 22	T22	I	tagadr_h 21
T23	P	Vdd plane	T24	I	tagadr_h 20
U1	B	data_h 76	U2	P	Vdd plane
U3	B	data_h 12	U4	B	data_h 75
U5	P	Vdd plane	U6	P	Vss plane
U19	P	Vdd plane	U20	P	Vss plane
U21	I	tagadr_h 26	U22	I	tagadr_h 25

[4] **tagEq_l** for **21064— lockWE_h** for **21064A**

[5] **tagadr_h 17** for **21064— lockFlag_h** for **21064A**

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
U23	I	tagadr_h 24	U24	I	tagadr_h 23
V1	B	data_h 11	V2	B	data_h 74
V3	B	data_h 10	V4	B	data_h 73
V5	P	Vss plane	V6	P	Vdd plane
V19	P	Vss plane	V20	P	Vdd plane
V21	I	tagadr_h 29	V22	I	tagadr_h 28
V23	P	Vss plane	V24	I	tagadr_h 27
W1	B	data_h 9	W2	P	Vss plane
W3	B	data_h 72	W4	B	check_h 6
W5	P	Vdd plane	W6	P	Vss plane
W7	P	Vdd plane	W8	P	Vss plane
W9	I	testClkIn_h	W10	I	testClkIn_l
W11	P	Vdd plane	W12	I	clkIn_h
W13	I	clkIn_l	W14	P	Vss plane
W15	P	Vdd plane	W16	P	Vss plane
W17	P	Vdd plane	W18	P	Vss plane
W19	P	Vdd plane	W20	P	Vss plane
W21	I	tagadrP_h	W22	I	tagadr_h 32
W23	I	tagadr_h 31	W24	I	tagadr_h 30
Y1	B	data_h 8	Y2	B	data_h 71
Y3	B	data_h 7	Y4	B	data_h 68
Y5	P	Vss plane	Y6	P	Vdd plane
Y7	P	Vss plane	Y8	P	Vdd plane
Y9	P	Vss plane	Y10	P	Vdd plane
Y11	P	Vss plane	Y12	P	Vdd plane
Y13	P	Vss plane	Y14	P	Vdd plane
Y15	P	Vss plane	Y16	P	Vdd plane
Y17	P	Vss plane	Y18	P	Vdd plane

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
Y19	P	Vss plane	Y20	P	Vdd plane
Y21	B	adr_h 8	Y22	B	adr_h 5
Y23	P	Vdd plane	Y24	I	tagadr_h 33
AA1	B	check_h 20	AA2	P	Vdd plane
AA3	B	data_h 5	AA4	B	data_h 66
AA5	B	data_h 0	AA6	I	iAdr_h 6
AA7	I	iAdr_h 10	AA8	I	vRef
AA9	O	sysClkOut2_h	AA10	O	sysClkOut2_l
AA11	N	spare 6[6]	AA12	O	sysClkOut1_h
AA13	O	sysClkOut1_l	AA14	I	cont_l
AA15	I	irq_h 5	AA16	N	spare 8[7]
AA17	B	adr_h 31	AA18	B	adr_h 27
AA19	B	adr_h 24	AA20	B	adr_h 17
AA21	B	adr_h 15	AA22	B	adr_h 11
AA23	B	adr_h 7	AA24	B	adr_h 6
AB1	B	data_h 70	AB2	B	data_h 69
AB3	B	data_h 67	AB4	B	data_h 2
AB5	B	data_h 64	AB6	I	iAdr_h 7
AB7	I	iAdr_h 12	AB8	I	reset_l
AB9	I	sRomD_h	AB10	O	sRomOE_l
AB11	O	cpuClkOut_h	AB12	I	dcOk_h
AB13	I	triState_l	AB14	I	icMode_h 0
AB15	I	irq_h 4	AB16	I	perf_cnt_h 0
AB17	B	adr_h 32	AB18	B	adr_h 28
AB19	B	adr_h 25	AB20	B	adr_h 21
AB21	B	adr_h 18	AB22	B	adr_h 14
AB23	P	Vss plane	AB24	B	adr_h 9

[6] **spare 6** for **21064— resetSClk_h** for **21064A**

[7] **spare 8** for **21064— sysClkDiv_h** for **21064A**

(continued on next page)

Table 10–18 (Cont.) 21064/21064A PGA Pin List

PGA Location	Type	Name	PGA Location	Type	Name
AC1	B	data_h 6	AC2	P	Vss plane
AC3	P	Vdd plane	AC4	B	data_h 65
AC5	P	Vss plane	AC6	I	iAdr_h 8
AC7	P	Vdd plane	AC8	I	iAdr_h 11
AC9	P	Vss plane	AC10	O	sRomClk_h
AC11	P	Vdd plane	AC12	N	spare 5
AC13	P	Vss plane	AC14	I	irq_h 2
AC15	P	Vdd plane	AC16	I	perf_cnt_h 1
AC17	P	Vss plane	AC18	B	adr_h 29
AC19	P	Vdd plane	AC20	B	adr_h 22
AC21	P	Vss plane	AC22	B	adr_h 16
AC23	P	Vdd plane	AC24	B	adr_h 10
AD2	B	data_h 4	AD3	B	data_h 3
AD4	B	data_h 1	AD5	I	iAdr_h 5
AD6	I	iAdr_h 9	AD7	N	spare 1[8]
AD8	I	eclOut_h	AD9	I	dInvReq_h[9]
AD10	N	spare 2	AD11	N	spare 4
AD12	I	icMode_h 1	AD13	I	irq_h 0
AD14	I	irq_h 1	AD15	I	irq_h 3
AD16	N	spare 7	AD17	B	adr_h 33
AD18	B	adr_h 30	AD19	B	adr_h 26
AD20	B	adr_h 23	AD21	B	adr_h 20
AD22	B	adr_h 19	AD23	B	adr_h 13
AD24	B	adr_h 12	–	–	–

[8]**spare 1** for **21064— icMode_h 2** for **21064A**

[9]**dInvReq_h** for **21064— dInvReq_h 0** for **21064A**

A

Designing a System with the 21064

A.1 Introduction

This appendix provides a basic description of how to integrate the DECchip **21064** Alpha AXP microprocessor chip into a printed circuit board (PCB) or system. It describes how the processor reacts to the chip reset condition, and explains how to connect and control the chip interface signals. The **21064** chip allows maximum flexibility while providing the ability to easily create a computing system with generally available PCB parts.

The examples in the text are used to clarify meaning only; what is described is not the only way to use the chip. An attempt has been made to describe real, usable circuits and techniques, but the chip is flexible and the designer is encouraged to investigate other implementations. Chapter 1 through Chapter 10 describes the details and additional features of the chip.

The following major topics are described in this appendix:

- General Concepts
- Basic Power, Input Level, and Clock Issues
- Booting the **21064**
- Cache/Memory Interface Details
- Load Locked and Store Conditional
- Special Request Cycles
- DMA Access
- Backmapping the Internal **21064** Dcache
- I/O Interface

A.2 General Concepts

Some important design concepts common to many **21064**-based system designs are described in this section. The chip external interface is flexible and mandates few design rules, leaving open a wide range of prospective systems. Figure A–1 is a diagram of the **21064** external interface, showing the major signal groups.

A system designed with the **21064** chip can be divided into three major sections:

- **21064** processor itself
- System control logic
- External backup cache

The chip interface provides address and control signals, and transfers data through a 128-bit bidirectional data bus.

The Bcache is optional, though most systems will see a performance benefit if it is included.

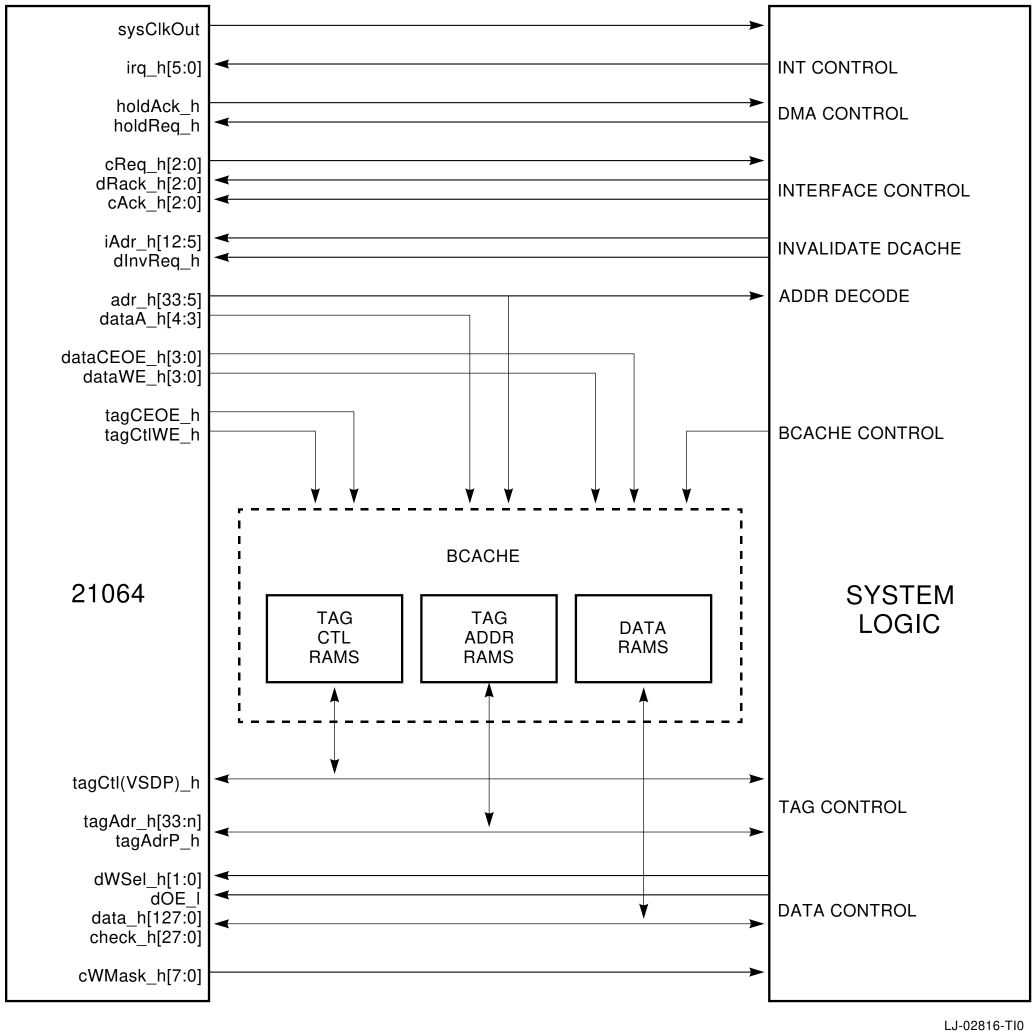
The signals between the three parts is shown in Figure A–1. Chapter 6 describes the function and purpose of the signals.

The processor controls the Bcache when its initial tag probe finds that the information is valid and unshared. The Bcache access is under control of the CPU, and the external system logic is not involved. The processor starts an external cycle when:

- The CPU does a Bcache probe and misses.
- A lock-associated command is invoked.
- A Bcache write is directed at a shared block.
- The Bcache is turned off.
- The CPU addresses a non-cached quadrant of memory.

During the external cycle, the Bcache is controlled by the system logic. The system logic either returns the data to or accepts the data from the processor (depending upon the cycle type), and acknowledges the cycle to give control back to the CPU. If the cycle necessitates a Bcache fill, it is up to the system logic to load the data into the Bcache RAMs, the upper address bits (with good parity) into the tag address RAMs, and the proper valid and parity bits into the tag control RAMs.

Figure A–1 21064 External Interface



To help design engineers create high performance systems more easily, the Bcache is controlled by the **21064** during probes that hit (except for writes directed at a shared cache block). This allows off-the-shelf SRAMs to be connected to the chip without many extra components. The Bcache interface signals are programmable through an internal processor register (IPR), so that the Bcache size, access speed, and write timing can be set with complete flexibility.

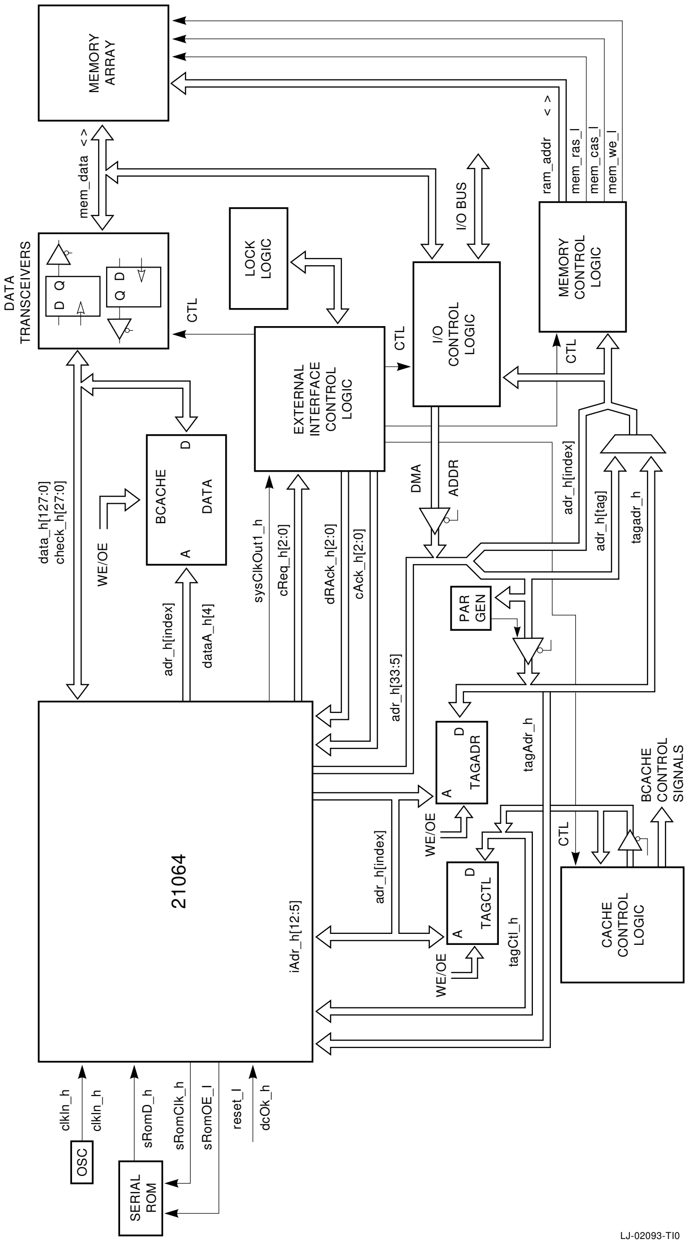
This external cache control is performed without affecting the internal CPU clock speed. That is, the **21064** can be running at its nominal 6.6 ns internal cycle time, but the Bcache can run slower if required without slowing down the internal timing. There are two internal caches in the **21064** chip: an I-stream read-only cache (Icache) and a D-stream write-through cache (Dcache). The speed of the Bcache does not affect the internal caches, which use the internal clock.

Figure A–2 is a block diagram of a system that can be created using the **21064** microprocessor. The major sections are shown, along with many of the buses that would run through such a system. In the center of the diagram is the external interface control, which directs the other system logic subsections that interface to memory, I/O, Bcache, and so on.

The Bcache, when included in a system, can be as small as 128 KB or as large as 16 MB. The size is under program control. The **adr_h [33:5]** bus in Figure A–2 is shown partitioned into an [index] field and a [tag] field. The size of each field depends upon the Bcache size. The smallest Bcache (128 KB) uses **adr_h [16:5]** to index into the cache block, and the tag field would be **adr_h [33:17]**. Only those bits that are actually needed for the amount of potentially cached system main memory need to be stored in the Bcache tag, although the **21064** uses all the relevant tag address bits for that Bcache size on its tag compare. A larger Bcache uses more index bits and fewer tag address bits.

On an external request (read or write), the **21064** sends out the address and cycle type (and data for a write cycle), then waits until the system logic sends back the acknowledge handshake that the cycle is complete. On a read request cycle, the system logic tags each data word as it comes back with information about whether the data should be checked for ECC (or parity, depending upon which mode of operation has been selected for the chip), and whether the data should be cached inside the chip. On a write request, the system logic merely notifies the chip that the write has been accepted for processing.

Figure A–2 21064-Based System Block Diagram



The Bcache is shared between the **21064** and the system logic. Although the processor directly manipulates the Bcache for read and write hits, it is up to the system logic to:

- Fill the Bcache with memory data.
- Load the tag address and tag address parity.
- Load tag control bits and parity on fills (valid and non-dirty).
- Write data back to memory when necessary.
- Probe the Bcache for lock/unlock transactions.
- Probe and control the Bcache for DMA transactions.

The Bcache control signals are therefore under potential control of the **21064** or the system logic. When the CPU chip determines that an external cycle is necessary, it drives the Bcache control signals to false. This allows the system logic to read and write the Bcache RAMs. Figure A–3 shows the expected configuration for the Bcache. The figure shows a data line, but the tag address and control lines are expected to be connected similarly.

The signal **mem_data** in Figure A–3 is a bidirectional memory data bus that connects to the main storage. When it is necessary to load the contents of memory into the Bcache, the system logic drives the memory bus control signals such that a read cycle is performed. In this example, the signal **read_mem_L** is being used to drive the Bcache (and **21064**) data bus. The system logic appropriately drives the Bcache RAM write enable signal, and once the data is stable on the **data_h [x]** bus, it is strobed into the Bcache.

When the Bcache contents need to be written back to memory, the system logic controls the RAM output enable signal to access the Bcache data. The signal **read_mem_L** is then de-asserted, and the memory control signals also correctly tristate the **mem_data** bus so that the data can be written to the memory storage elements. The system logic must de-assert the **21064** signal **dOE_l** so that the CPU does not drive the **data_h [x]** lines.

Figure A–3 Bcache Control Logic

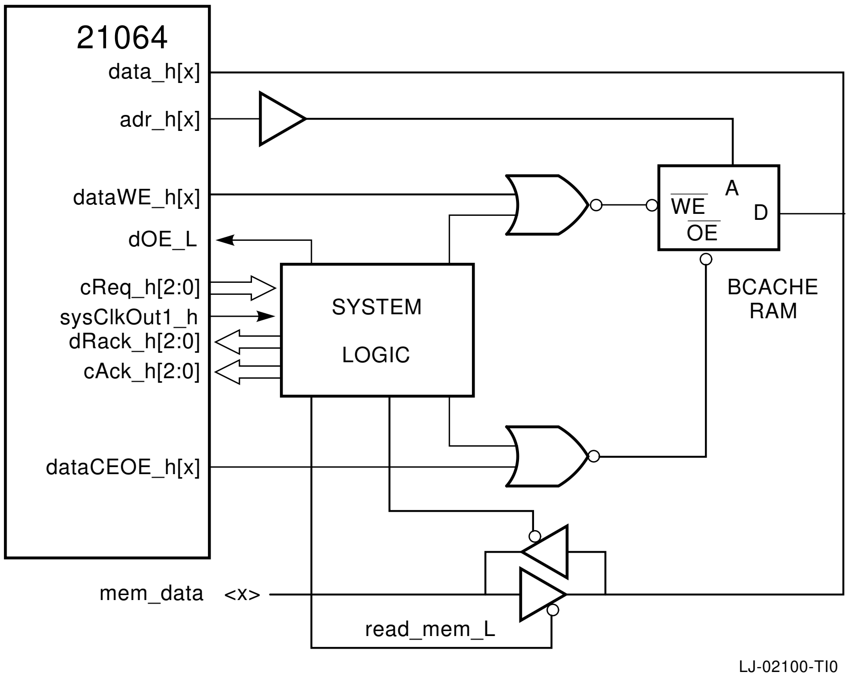
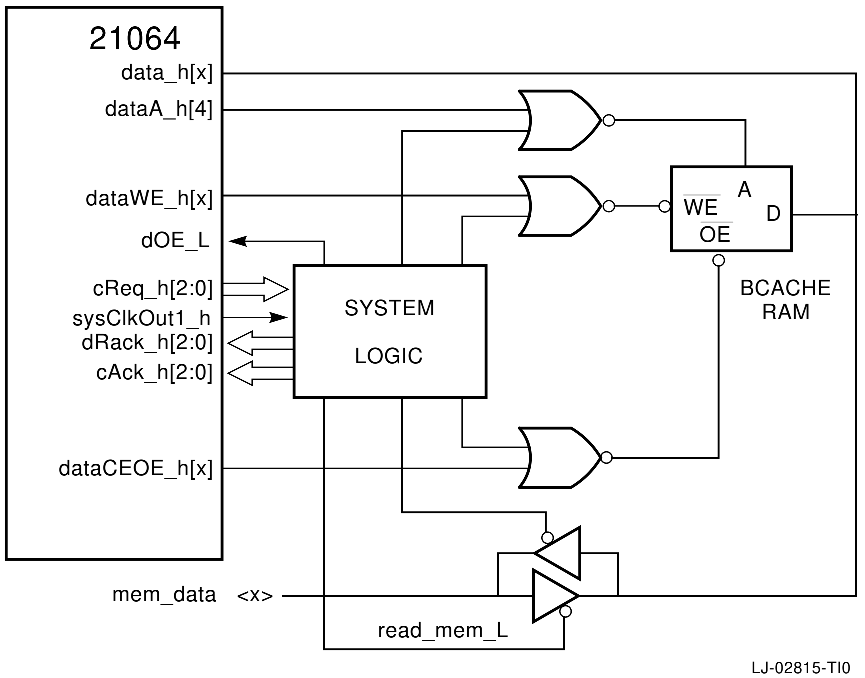


Figure A–4 Lower Bcache Address



The Bcache consists of 32-byte blocks. As such, the **21064** supplies address bits [33:5] to select which Bcache block is currently being accessed. The CPU data bus is 16 bytes wide, and thus each Bcache cycle requires two accesses. The CPU outputs the signal **dataA_h [4]** to control which 16-byte data half is being written to or read from.

Figure A–4 shows the expected configuration for the lower address bit. As with the chip output enable and write pulse, the lower Bcache address bit is under control of either the **21064** or the system logic. When the CPU is in external system logic mode, it drives the **dataA_h [4]** signal low (along with the other Bcache control signals).

Some general cycle types, including timing diagrams, are described in later sections to better explain how a **21064**-based system functions.

A.3 Basic 21064 Power, Input Level, and Clock Issues

This section provides an overview of these issues, and some example circuits that can be used.

A.3.1 Power Supply and Input Levels

The **21064** is powered from a +3.3 V supply (+/- 5%), but can drive and accept CMOS/TTL-compatible levels once the chip has been correctly stabilized. It is *mandatory* that no input or bidirectional pin be allowed to rise above 4.0 V until the 3.3 V power to the chip is stable.

Caution

Failure to follow this rule can damage the chip.

Although power sequencing can be used to ensure that this restriction is met, the rule itself does not mandate power sequencing. It only means that other module parts capable of driving the input pins of the **21064** must be kept from doing so until the **21064** has stable power. In practice this can often be accomplished by keeping the potentially offending outputs in tristate until the **21064** has a stable +3.3 V voltage. For example, a **dcOK** signal can be used to prevent components such as SRAMs, MUXes, and buffers from driving the chip.

There are some caveats associated with this approach. The **dcOK** signal might be generated from the central power supply, whereas the +3.3 V might be generated as a by-product of another voltage (perhaps the +5 V supply). If the regulator that creates the 3.3 V is faulty, the **dcOK** signal might allow the inputs of the **21064** to be driven, possibly damaging the chip. A power supply supervisor, actually sampling the +3.3 V and generating a tristate enable based upon it, is a safer approach.

There are other potential voltage paths that must be removed if power sequencing is not used. Termination resistors or pullups to +5 V can be a source of voltage to the **21064** input pins, as can some bipolar TTL inputs (since they can source current). Care must be taken to eliminate all potential **21064** input voltage sources if power supply sequencing is not used.

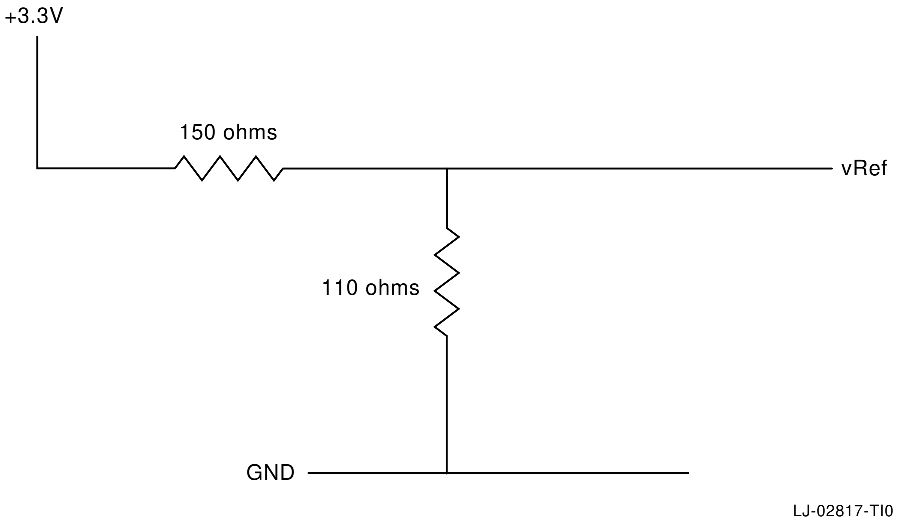
A.3.2 Input Level Sensing

The **21064** uses a reference input pin, **vRef**, to supply the threshold level for all chip inputs except:

- **clkIn_h** and **clkIn_l**
- **testclkIn_h** and **testclkIn_l**
- **tagOk_h** and **tagOk_l**
- **dcOk_h**
- **eclOut_h**
- **tristate_l**
- **cont_l**

These pins should never be driven at a higher voltage than the **21064** power supply. Since the nominal voltage to the chip is 3.3 V, care must be taken if any of these signals are generated from logic that has a 5 V supply. Note especially that **dcOk_h** is one of the signals that must never be driven higher than the nominal 3.3 V level, since it is likely that it will be generated from a higher voltage.

Figure A–5 Input Reference Voltage Circuit



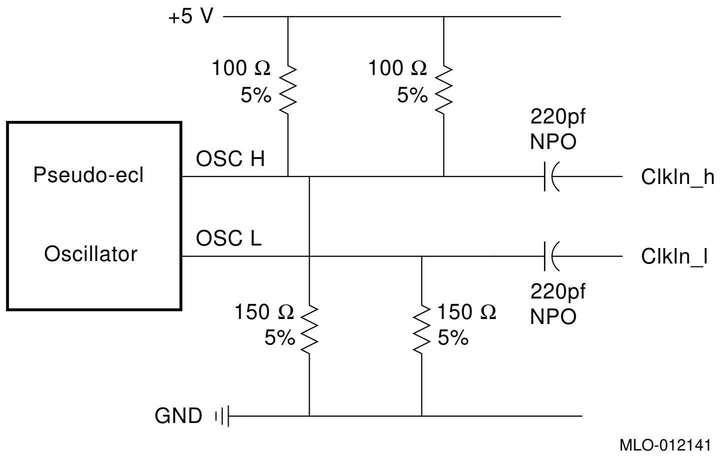
The input pin **vRef** should be connected to a stable 1.4 V (+/-10%) source. The circuit shown in Figure A–5 can be used to supply this voltage level. **vRef** has a large capacitance on it inside the chip, and there is an RC delay between its pin and the other input buffers. Therefore, **dcOk_h** should not be asserted until there has been enough time for the **vRef** input to stabilize. See Section 6.5.2 for more information about the assertion of **dcOk_h**.

Note that **reset_l** *is* one of the input pins that uses **vRef** for its threshold level, so it cannot be relied upon until **vRef** is stable. Because of **dcOk_h** being false (low), the chip is kept in reset mode.

A.3.3 Input Clocks

The **21064** expects differential clock signals between 0.6 V and 3.0 V for the **clkIn_h** and **clkIn_l** inputs. A correctly terminated pseudo-ECL oscillator can be ac-coupled to the clock inputs for this purpose. Using a pseudo-ECL oscillator means you do not have to design a special ECL power supply to clock the chip. Figure A–6 is an example of a working circuit. Note that the series capacitor should use an NPO dielectric.

Figure A–6 Input Clock Circuit



For up to 200 MHz (translating to a 10 ns internal CPU clock cycle), a lower-cost 10K-series oscillator can work fine. Greater than that speed, a 100K-series oscillator should be used.

Due to internal chip circuitry, the test clock input signals (**testClkIn_h** and **testClkIn_l**) should be pulled to the appropriate level using small resistors (100 ohms maximum). **testClkIn_h** should be pulled high (that is, to 3.3 V through a small resistor) and **testClkIn_l** should be pulled low (to ground).

A.3.4 Unused Inputs

There are several inputs that are not used in a **21064**-based system, but must be tied off either high or low. The following inputs should be pulled to 3.3 V through a resistor:

- **tagOk_h** (unless using the tagOk function)
- **tristate_l**
- **cont_l**
- **perfCntIn_h [1:0]** (unless using the performance counter inputs)

Note

Any input on the **21064** that is pulled high must use the +3.3 V rail, and *not* the +5 V rail.

The following inputs should be pulled to ground:

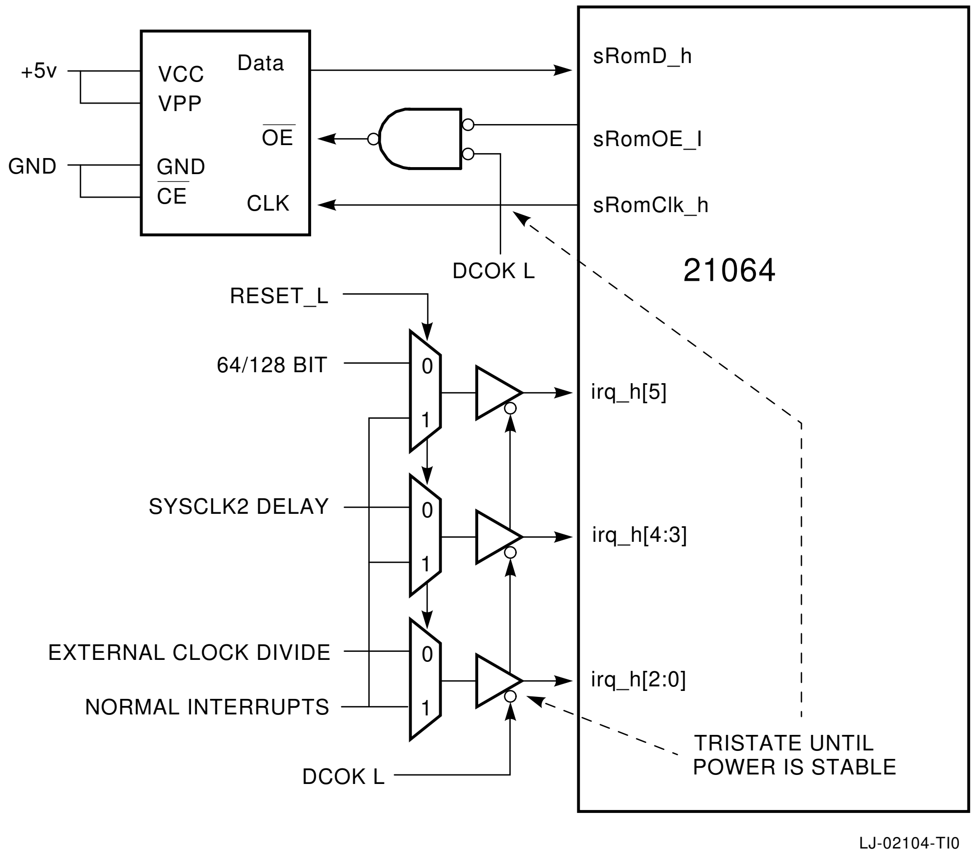
- **tagOk_l** (unless using the tagOk function)
- **dWSel [0]** (unless in 64-bit data bus mode)
- **eclOut_h**
- **icMode_h [1:0]**

The **tagOk_h** and **tagOk_l** signals are used to stall the **21064** so that the Bcache can be controlled by the system logic. They are optimized for very high performance systems, and are not included in this appendix. See Chapter 6 for more details about the signals and their use. This appendix includes the simpler **holdReq_h** method for the system logic to take control of the Bcache (see Section A.8).

A.4 Booting the 21064

The **21064** uses a flexible method to bootstrap the processor. Instead of always jumping to a fixed I/O address upon reset, the chip can load its initial I-stream from a compact serial ROM (SROM). As well, the configuration of the external interface is programmable by setting up certain input pins at reset time. Figure A–7 shows how the serial ROM and the configuration inputs are used at reset time.

Figure A–7 Serial ROM and Programmable Clock Inputs



While the **21064** is in reset mode, the interrupt request input lines **irq_h [5:0]** are inspected to determine how the chip should configure the external interface logic. There are three configurable areas:

- The **21064** can accommodate either a high-performance 128-bit external data bus or a lower-cost 64-bit data bus. **irq_h [5]** determines which of the two is selected, and is asserted high to choose the 128-bit mode. This appendix describes the **21064** in 128-bit mode.
- The external interface runs synchronously to the external system clock, **sysClkOut1_h**. This external clock is generated from the internal clock, which can be divided by any value from 2 to 8 generating **sysClkOut1_h**. For example, the **21064** chip running at its nominal 6.6 ns internal clock cycle time can be divided by 4 to allow an external interface to run at 26.4 ns. **irq_h [2:0]** selects the external interface division factor. Table A–1 is a chart of the clock divisor decode.

Table A–1 System Clock Divisor

irq_h [2]	irq_h [1]	irq_h [0]	Ratio
0	0	0	2
0	0	1	3
0	1	0	4
0	1	1	5
1	0	0	6
1	0	1	7
1	1	0	8
1	1	1	8

- The external interface logic is supplied with two differential clocks from the **21064**:
  - **sysClkOut1_h** and **sysClkOut1_l**
  - **sysClkOut2_h** and **sysClkOut2_l**

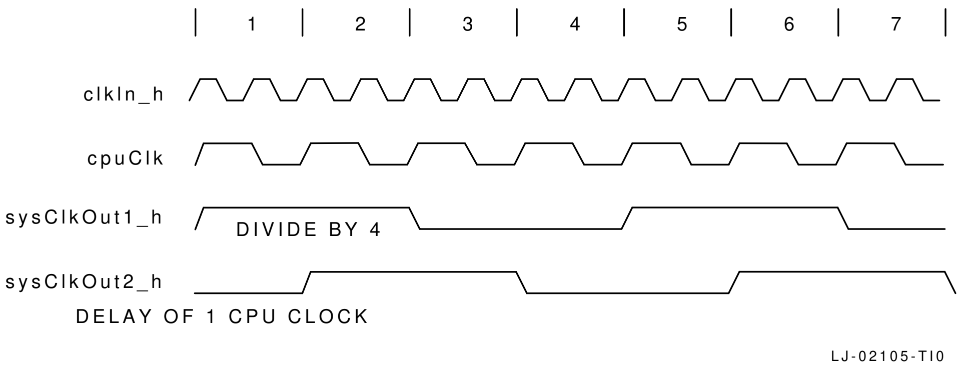
  Each external clock runs at the external cycle time selected. **sysClkOut2** can also be delayed from **sysClkOut1** by a programmable value selected from **irq_h [4:3]**. The second clock can be delayed from 0 to 3 internal CPU clocks based upon this selection. Table A–2 lists the delay times possible and their decode meaning.

Table A–2 System Clock Delay

irq_h [4]	irq_h [3]	Delay
0	0	0
0	1	1
1	0	2
1	1	3

Figure A–8 shows how the clock configuration works. The input clock that is provided to the **21064** chip is divided by 2 to create the internal CPU clock. The CPU clock is the reference to all the other clocks that the chip outputs. In the example, the clock divisor is 4, so the system output clocks run at 1/4 of the internal CPU clock time. The figure shows that **sysClkOut2** has been delayed by 1 CPU clock from **sysClkOut1**. Since the external output clocks are differential, a two-phase clock is also available by using **sysClkOut1_h** and **sysClkOut1_l**.

Figure A–8 Example of 21064 Clock Configuration



Note

Figure A–8 is only meant to show the general method by which the clocks are created within the **21064**. The *phase* relationships shown, especially between **clkIn_h** and **cpuClk**, are not guaranteed.

When the **reset_l** signal de-asserts, the serial ROM is loaded into the processor Icache. The CPU controls the output enable and the clock for the ROM, and accepts the bit serial data. Chapter 6 provides information about the timing of the SROM control signals. After the SROM data has been loaded into the Icache, the processor jumps to location 0, which hits inside the Icache. The SROM code is expected to perform chip and system initialization, preparing the system for external operation.

After the SROM code has been executed, it is assumed that the external interface is ready to supply I-stream data to the **21064** processor. The Bcache can be on or off at this point (in fact, there is no need to have a Bcache if the user has no performance reason to include it). A general system might include a more complete boot/diagnostic ROM (BDROM) after the SROM has done its job.

Once the **21064** is executing in I-stream mode from an external interface, it expects full 32-byte fills. The normal data path of the **21064** is 128 bits (16 bytes), so two complete fill cycles are necessary to provide the 32 bytes of data. The BDROM code can be loaded and executed in several ways, though the suggested method is to move the BDROM code into RAM memory, then execute it from there. This can be easily handled by the serial ROM, which can read the BDROM byte-by-byte, pack it into appropriate memory words, move it into main memory, then jump to it in RAM.

A.5 Cache/Memory Interface Details

The Bcache subsystem is carefully integrated into the **21064**, therefore the Bcache SRAMs can be directly controlled by the **21064** interface, and the Bcache data lines are connected to the **21064** data bus, as shown in Figure A–2.

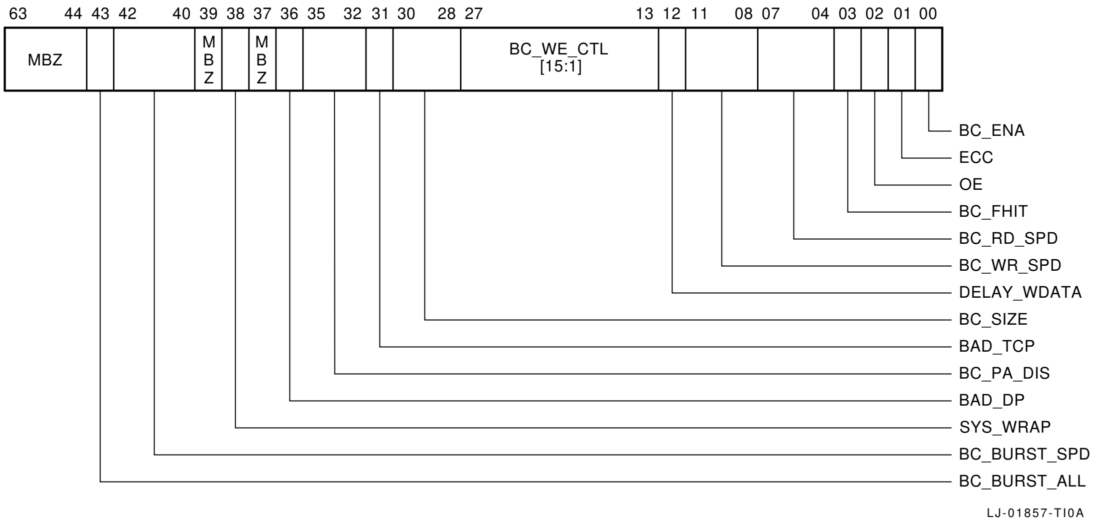
The Bcache is organized into 32-byte blocks, with parity or ECC on 4-byte (32-bit) segments. When the Bcache is enabled, the **21064** generally probes it for each memory access (lock-related cycles are an exception). The tag and control SRAMs are first enabled at the appropriate address, and if the probe finds a valid match the cycle finishes without performing a main memory read or write cycle. The first 128-bit (16-byte) data segment is read at the same time as the Bcache tag probe, and is ready if the probe is successful. The **21064** then reads the second 128-bit segment. If the internal cache is enabled, the data is saved inside the chip.

The Bcache is best utilized in writeback mode, which means that both reads and writes are normally serviced from the Bcache without external logic intervention. This implies that the Bcache has the only valid copy of a data block after it’s been modified. The **21064** manipulates the Bcache DIRTY bit to signify that the block has been written since it was initially read from memory. There is a method that the system logic can use to force non-writeback behavior, but its use is beyond the scope of this appendix.

A.5.1 Bcache Timing for 21064 Access

The Bcache timing is under complete control of the user through the BIU_CTL internal processor register (IPR). Figure A–9 shows the layout of this register, which is normally set up as part of the chip initialization code. The number of internal CPU cycles to allocate for Bcache reads and writes can be specified, along with the exact representation of where the Bcache write pulse is asserted for Bcache writes.

Figure A–9 21064 BIU_CTL Internal Processor Register[1]



[1] Previous versions of the **21064** did not implement BIU_CTL [43, 42:40, 38, 12]. PALcode for these previous processors is upwards compatible if the PALcode did not set these bits.

The register fields shown in Figure A–9 are described in Table A–3.

Table A–3 Bus Interface Unit Control Register Fields

Field	Type	Description
BC_ENA	WO,0	External cache enable. When this bit is cleared, the bit disables the external cache. When the Bcache is disabled, the BIU does not probe the external cache tag store for read/write references; it launches a request on **cReq_h** immediately.
ECC	WO,0	When this bit is set, the **21064** generates/expects ECC on the **check_h** pins. When this bit is cleared, the **21064** generates /expects parity on four of the **check_h** pins.
OE	WO,0	When this bit is set, the **21064** does not assert its chip enable pins during RAM write cycles, thus enabling these pins to be connected to the output enable pins of the cache RAMs.

Caution

The output enable bit in the BIU_CTL register (BIU_CTL [2]) must be set if the system uses SRAMs in the output enable mode (that is, if the **tagCEOE** and/or **dates** signals are connected to the output enable input of the SRAM and the **21064** enable is always enabled). If this bit is inadvertently cleared, the tag and data SRAMs will be enabled during writes, and damage can result.

(continued on next page)

Table A–3 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
BC_FHIT	WO,0	External cache force hit. When this bit is set and the BC_ENA bit is also set, all pin bus READ_BLOCK and WRITE_BLOCK transactions are forced to hit in external cache. Tag and tag control parity are ignored. The BC_ENA takes precedence over BC_FHIT. When BC_ENA is cleared and BC_FHIT is set, no tag probes occur and external requests are directed to the **cReq_h** pins. **Note** The BC_PA_DIS field takes precedence over the BC_FHIT bit.
BC_RD_SPD	WO,0	External cache read speed. This field indicates to the BIU the read access time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less the read access time of the external cache RAMs. The access times for reads must be in the range [16:4] CPU cycles, which means the values for the BC_RD_SPD field are in the range of [15:3].
BC_WR_SPD	WO,0	External cache write speed. This field indicates to the BIU the write cycle time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less the write cycle time of the external cache RAMs. The access times for writes must be in the range [16:2] CPU cycles, which means the values for the BC_WR_SPD field are in the range of [15:1].
DELAY_WDATA	WO	When this bit is set, it changes the timing of the data bus during external cache writes. This bit is not initialized by chip reset. See Section 6.4.4.

(continued on next page)

Table A–3 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
BC_WE_CTL	WO,0	External cache write enable control. This field is used to control the timing of the write enable and chip enable pins during writes into the data and tag control RAMs. It consists of 15 bits, where each bit determines the value placed on the write enable and chip enable pins during a given CPU cycle of the RAM write access. When a given bit of the BC_WE_CTL is set, the write enable and chip enable pins are asserted during the corresponding CPU cycle of the RAM access. The BC_WE_CTL bit [0] (bit [13] in BIU_CTL) corresponds to the second cycle of the write access, BC_WE_CTL [1] (bit [14] in BIU_CTL) to the third CPU cycle, and so on. The write enable pins will never be asserted in the first CPU cycle of a RAM write access. Unused bits in the BC_WE_CTL field must be written with zeros.
BC_SIZE	WO,0	This field is used to indicate the size of the external cache. See Table A–4 for the encodings.
BAD_TCP	WO,0	When set, this bit causes the **21064** to write bad parity into the tag control RAM whenever it does a fast external RAM write. (Diagnostic use only.)
BC_PA_DIS	WO,0	This 4-bit field may be used to prevent the CPU chip from using the external cache to service reads and writes based upon the quadrant of physical address space that they reference. The correspondence between this bit field and the physical address space is shown in Table A–5. When a read or write reference is presented to the BIU the values of BC_PA_DIS, BC_ENA, and the physical address bits [33:32] determine whether to attempt to use the external cache to satisfy the reference. If the external cache is not to be used for a given reference the BIU does not probe the tag store and makes the appropriate system request immediately. The value of BC_PA_DIS has NO impact on which portions of the physical address space can be cached in the primary caches. System components control this by way of the **dRAck_h** field of the pin bus.
BAD_DP	WO,0	When this bit is set, the BAD_DP causes the **21064** to invert the value placed on bits [0], [7], [14] and [21] of the **check_h [27:0]** field during off-chip writes. This produces bad parity when the **21064** is in parity mode, and bad check bit codes when in ECC mode. (Diagnostic use only.)

(continued on next page)

Table A–3 (Cont.) Bus Interface Unit Control Register Fields

Field	Type	Description
SYS_WRAP[2]	WO,0	When this bit is set, it indicates that the system returns read response data wrapped around the requested chunk. This bit is cleared by chip reset.
BC_BURST_SPD[2]	WO,0	When these bits are cleared, this field is ignored. Bcache is timed as it always is. When these bits are set in 128-bit mode, the second half of read takes BC_BURST_SPD+1 cycles. When these bits are set in 64-bit mode, the second and fourth reads take BC_BURST_SPD+1 cycles. If BC_BURST_ALL is set, the third read takes BC_BURST_SPD+1 cycles also.
BC_BURST_ALL[2]	WO,0	In 64-bit mode this bit is set if BC_BURST_SPD should be used to time the third (of four) RAM read cycle.

[2]BC_BURST_ALL, BC_BURST_SPD, SYS_WRAP, and DELAY_WDATA were not implemented in previous **21064** chip designs. PALcode which did not set these bits may be used without change.

Table A–4 lists the encoding for BC_SIZE. Table A–5 lists the BIU_CTL physical addresses.

Table A–4 BC_SIZE

BC_SIZE	Cache Size	BC_SIZE	Cache Size
0 0 0	128 KB	1 0 0	2 MB
0 0 1	256 KB	1 0 1	4 MB
0 1 0	512 KB	1 1 0	8 MB
0 1 1	1 MB	1 1 1	16 MB

Table A–5 BC_PA_DIS

BIU_CTL Bits	Physical Address	BIU_CTL Bits	Physical Address
32	PA [33:32] = 0	33	PA [33:32] = 1
34	PA [33:32] = 2	35	PA [33:32] = 3

Note

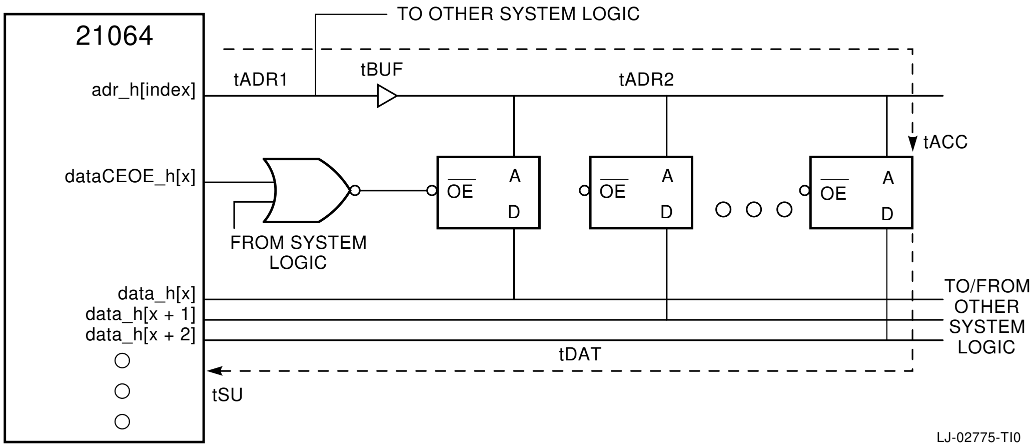
Writing ones to the BC_PA_DIS bit causes reads and writes to the corresponding physical address ranges not to be mapped into the external Bcache.

A.5.1.1 Bcache Read Cycle

For a Bcache read cycle, the access/cycle time is determined by adding the complete address or control path from the **21064** output pins until the data is valid at the **21064** data bus input pins. There is a 4.5 ns data setup requirement inside the **21064** that must also be considered. A system designed with the **21064** must provide access to the Bcache address and control signals from the printed circuit board (PCB) logic, so there is a NOR-type gate in the path. Furthermore, the **21064** output buffers are characterized driving a 40 pF load, so any large fanout must be accomplished without exceeding this value. This usually means that buffers should be added to the address and control paths.

An example of a Bcache read access time calculation is provided here to clarify the steps. Figure A–10 shows the general circuit assumed for this example. The address path drive signals are normally treated as transmission lines in a real high-performance Bcache. In the example, the address buffer has a specified propagation delay of 5 ns. One of the address lines is a fast, high-drive capability NOR-gate, and for our purposes is treated like the address buffer. Many devices specify the maximum propagation delay with only one output switching, and in the case of an address buffer all the outputs might switch simultaneously. To account for this, extra buffer delay should be added to the assumed propagation delay through the device. For this example, the 5 ns buffer delay takes this into account. Figure A–10 does not show any termination on tADR2, while in a real system some kind of parallel or series termination would be needed. A few general points about termination are described in this document. The best way to determine the actual delay time associated with tADR2, and to decide on the type of termination and the component values, is to use an analog simulation program such as SPICE.

Figure A–10 Bcache Access Path for 21064



If parallel termination is used (ac termination at the end of the line can be used with TTL drivers), then the address buffer must be able to drive a low impedance line to a proper level on the incident wave. If your driver cannot do this, or if your simulation finds that this is not the best termination method, then a series termination resistor should be used. Series termination usually implies that the delay time is increased due to the necessary reflection for a correct signal level.

All the calculations shown are based upon the stated assumptions. The system or board designer is responsible for analyzing any particular implementation, and determining the correct delays and signal integrity issues. The purposes of this example are to show a general Bcache circuit that can be implemented with the **21064**, and to explain how to program the IPR that controls the Bcache. Faster and slower systems can be built with the **21064** processor.

The SRAMs in the example have a specified access time of 20 ns from address stable to data valid at their output pins. SRAM devices often have a faster specification from output enable to data valid, and it is assumed that the *address* path, not the output enable path, is the critical one. The designer should ensure that this is true for any specific implementation. The output enable path can be analyzed similarly to the address path. The general components of delay for this calculation are:

```
tADR1 [delay from CPU to input of address buffer]
tBUF  [buffer gate delay]
tADR2 [address delay from buffer to SRAM inputs]
tACC  [SRAM access time from address valid to data valid]
tDAT  [data return path from SRAM to 21064 input pins]
tSU   [internal 21064 data setup time]
```

Figure A–11 Timing Diagram for Bcache Read Access

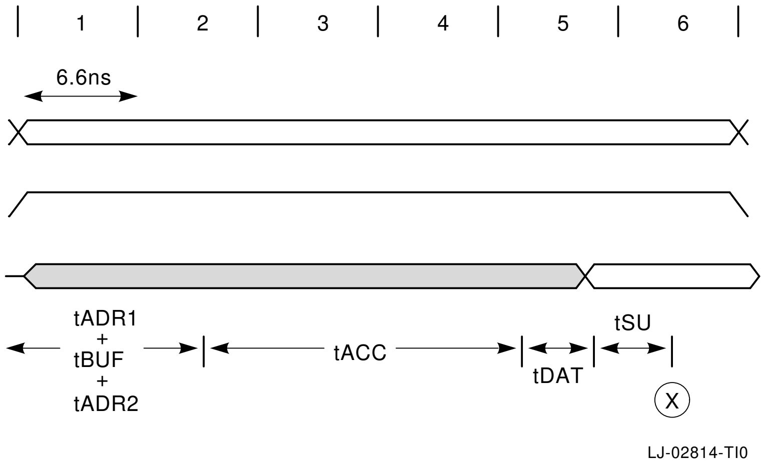


Figure A–11 is a timing diagram showing the **21064** signals and their delay components. The valid data cannot be sampled until the point labeled ‘‘X’’ in the figure. Chapter 6 provides more detailed timing diagrams of the fast Bcache access path. The Bcache probe and each data read access would have the timing shown in Figure A–11, and they are controlled by the same programmable BIU_CTL field.

The three unknown delay components are the address paths (tADR1, tADR2) and the data return path (tDAT). The tDAT depends on the edge rate of the SRAM output, the length of the data line, and the other loads that are connected to the data line. As such, it is impossible to specify a ‘‘normal’’ delay time. For this exercise, it is assumed to be 2 ns.

The address delay path from the **21064** address output to the buffer (tADR1) is similar to the data path. It is unlikely to be a classical transmission line, due to the line length in relation to the edge rate of the **21064** output. However, other loads are on the address line, and the etch itself causes a delay of approximately 160 ps to 200 ps per inch. For this example, tADR1 is specified as 2 ns.

The path tADR2 needs a more classical transmission line analysis, since the buffers have a fast switching time in relation to the line length. Even if the address drivers can switch the line to an appropriate level on the incident wave, the wave propagates along the transmission line more slowly than if it was unloaded. Each SRAM contributes some capacitance to the line, which slows the wave down according to the next formula:

```
tPL = tPD * SQRT(1+Ca/Co)
```

tPL	The loaded propagation delay per unit length
tPD	The propagation delay per unit length of the unloaded line
Ca	The added capacitance per unit length due to the SRAM inputs
Co	The unloaded transmission line capacitance per unit length

In this example, it takes the wave 2 ns to reach the last SRAM address input, where there is no reflection. If the address driver cannot switch the line on the incident wave, a series termination scheme is used instead, and the delay value might be higher.

The full trip from address valid at the **21064** output pin to data valid at the **21064** input pin (plus data setup) is:

```
 2    ns tADR1
 5    ns tBUF
 2    ns tADR2
20    ns tACC
 2    ns tDAT
 4.5  ns tSU
--------
35.5  ns
```

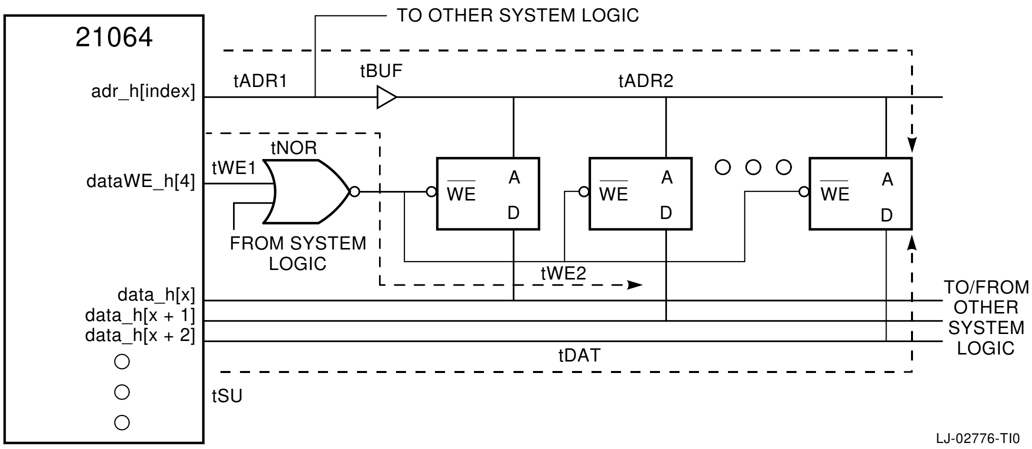
These numbers apply to this example only. If the designer uses different buffers, or splits the address drivers differently, or uses drivers that cannot switch the low impedance line on the incident wave, the analysis changes accordingly. We assume that the **21064** is using an internal cycle time of 6.6 ns, which means that the chip must allocate 6 cycles for the Bcache read given the conditions specified. This is programmed into the BIU_CTL register by setting the BC_RD_SPD field to 5, since the actual cycle count is one more than the one specified in the register. This value works for any round trip delay that is less than or equal to 39.6 ns.

It should be noted that using SRAMs with an access time of 17 ns reduces the number of internal CPU cycles to 5, assuming that everything else remains constant.

A.5.1.2 Bcache Write Cycle

A fast CPU-activated Bcache write cycle can be similarly analyzed. The BC_WR_SPD field in the BIU_CTL register should be programmed so that the SRAM write cycle finishes, and the BC_WE_CTL field should place the write pulse so that the timing and width do not violate the SRAM specifications.

Figure A–12 Cache Write Path for 21064



An example of this calculation is provided next. Figure A–12 shows the circuit that is assumed for the Bcache write path.

Figure A–13 Timing Diagram for Bcache Write Access

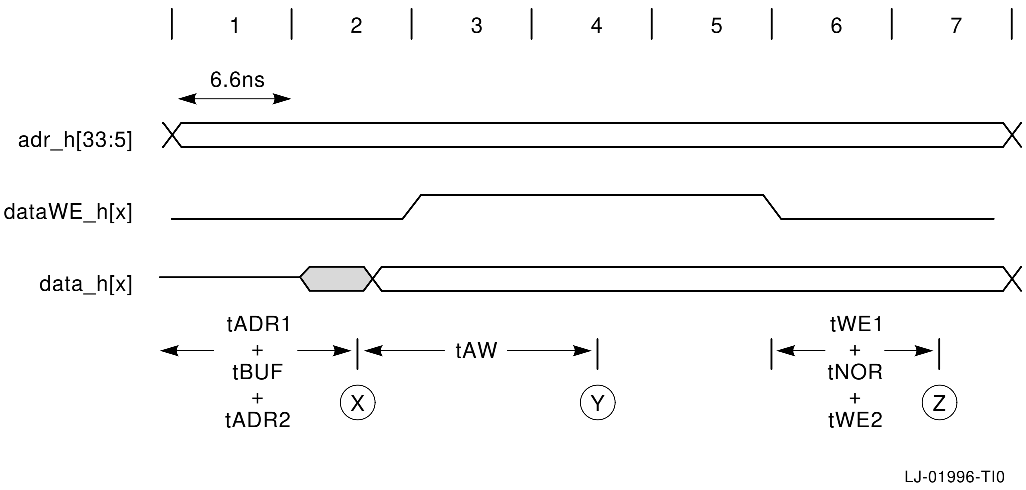


Figure A–13 shows a timing diagram of the write path signals as viewed from the **21064**. Chapter 6 provides a detailed timing diagram of a fast Bcache write access. The tag probe follows the timing for a fast Bcache read, and each write access follows the timing as shown in Figure A–13. The write pulse cannot assert until point ''X'' in the figure, and it cannot de-assert until point ''Y'' in the figure. The cycle cannot end until point ''Z'' in the figure.

In this example, the minimum write pulse width for the SRAM (tWM) is 15 ns. The **21064** can have 1 ns of skew between the rising and falling edges of the pulse it generates. Furthermore, although the rise and fall delays through the NOR gate in the figure should be close, some skew must be added to account for:

- Potential input threshold differences inside the SRAM
- Differences that result in a rise propagation delay that is different than the fall propagation delay

In the example, 2 ns of skew were added between the rising and falling edges of the write pulse (1 ns for the **21064**, and 1 ns for the logic and threshold differences). The following SRAM specifications are used in this example:

```
tWC = 20 ns  [Write cycle time]
tWP = 15 ns  [Write pulse width]
tDW =  8 ns  [Data setup time to write pulse de-assertion]
tDH =  0 ns  [Data hold time from write pulse de-assertion]
tAW = 15 ns  [Address setup time to write pulse de-assertion]
tWR =  0 ns  [Address hold time from write pulse de-assertion]
tAS =  0 ns  [Address setup time to write pulse assertion]
```

These specifications are only a subset of the total device specifications for a real device, and are used to show the general technique used to determine how to program the BIU_CTL IPR. Designers should closely analyze their own systems, including the device support logic, etch paths, and RAM specifications, in order to determine exactly which paths are critical. This should include running SPICE, or some other accurate analog simulator, to determine the delay times and transmission properties of the signals involved.

The first BIU_CTL field to calculate is the BC_WE_CTL, which determines where the write enable pulse is asserted. The field is 15 bits wide, and each bit represents an internal CPU cycle that asserts the write enable pulse (starting with the second cycle, since the first cycle never drives the write enable pulse).

The write enable pulse has to provide enough setup time for both the address and data paths. The most stringent of the two determines how early the write enable pulse can be de-asserted, which further limits how early the write cycle can end. First calculate how early the write enable pulse can be de-asserted (point ''Y'' in Figure A–13), based upon the address path. The address delay calculation is similar to the read case, and it should be added to the address setup time as follows:

```
 2 ns   tADR1
 5 ns   tBUF
 2 ns   tADR2
15 ns   tAW for SRAM
-------
24 ns
```

By this calculation then, the earliest that the write pulse can be de-asserted is 24 ns from the start of the write cycle, based upon the address setup requirement.

The next calculation determines how early the write enable pulse can be de-asserted, based upon the data setup requirement. There are two types of ''data'' that need setup and hold time for the write cycle. The actual Bcache data is the first type, and the tag control inputs (VALID, DIRTY, SHARED, and PARITY) are the second type. The **21064** drives the tag control inputs one CPU cycle later than the actual data, and we assume that they are the critical path. The chip provides stable data at most 2.5 ns after the nominal edge that drives the data (in this case, tag control) lines. We assume that the data take

2 ns to get to the SRAMs and be stable. If the CPU clock cycle is 6.6 ns, then the earliest that the write pulse can de-assert is calculated as follows:

```
6.6 ns   [1 CPU clock cycle]
2.5 ns   [21064 data stable time]
2.0 ns   tDAT
8.0 ns   tDW for SRAM
-------
19.1 ns
```

Since the value of 19.1 ns is less than the previously calculated value of 24 ns, it would appear that in this example the address path is the critical one. The write pulse cannot de-assert until 24 ns after the start of the write cycle. The minimum pulse width is specified to be 15 ns, which must be extended to $(15 + 2 =)$ 17 ns to account for the pulse width skew in the **21064** and the external logic (calculated previously in this section). At an internal 6.6 ns CPU cycle time, 3 cycles must be used for the write pulse.

Since the earliest that the write pulse can de-assert is 24 ns after the start of the write cycle, the latest that it can assert (in order to meet that de-assertion time) is $(24 - 17 =)$ 7 ns after the cycle start. We should now determine if other factors allow the write enable pulse to start that early. We see that it cannot, and then calculate how early it *can* start, and with what effect on the rest of the cycle.

We have specified here that the write pulse cannot assert until the address is stable (tAS), and this puts a bound on how early the write pulse is asserted. It was determined previously that the address reaches the last SRAM $(tADR1 + tBUF + tADR2 = 2 + 5 + 2 =)$ 9 ns after the start of the cycle. Since there is also 1 ns of skew between the address signal and the write pulse signal coming from the **21064**, the real minimum time is $(9 + 1 =)$ 10 ns from the cycle start.

The earliest that the **21064** can assert the write pulse is 10 ns after the cycle start, which puts it past the start of the second CPU cycle. The earliest that the write enable pulse can assert is the start of the third CPU cycle, which appears $(6.6 * 2 =)$ 13.2 ns into the Bcache write. To meet the minimum pulse width while asserting at the start of the third CPU cycle, the pulse must extend until the end of the fifth CPU cycle. The BC_WE_CTL field should be programmed to be 000000000001110 (bin). This means that the write pulse remains asserted until $(6.6\ ns * 5 =)$ 33 ns into the Bcache write, which puts it beyond the 24 ns address stable setup limit previously calculated. Everything works out for this example.

The other programmable field of interest in the BIU_CTL is the BC_WR_SPD field, which determines the entire write cycle time. The write pulse itself is de-asserted at the end of the fifth CPU cycle into the Bcache write in this example, which means it nominally de-asserts $(6.6 * 5 =)$ 33 ns from the start of the cycle. It might be 1 ns later than that due to **21064** output skew. There is also a NOR gate in the path (tNOR), and some wire travel time associated with the signal (tWE1 and tWE2).

There are three components of delay for the write enable pulse. The two write delay components (tWE1 and tWE2) might or might not be transmission lines. Designers should analyze the particular implementation to see what the correct configuration should be, and if one of them is a transmission line it should be terminated appropriately (this analysis is similar to the address calculation in the previous section).

We assume that tWE1 is 1 ns, tNOR is 5 ns, and tWE2 is 2 ns for this example. The latest that the write pulse can de-assert at the last SRAM (and thus the earliest that the cycle can end) is:

```
33 ns   [nominal write pulse de-assertion from start of write]
 1 ns   [21064 skew from nominal edge]
 1 ns   tWE1
 5 ns   tNOR
 2 ns   tWE2
-------
42 ns
```

At a 6.6 ns cycle time this translates to 7 cycles, so the value of 6 should be programmed into the BC_WR_SPD field (this cycle value is always 1 less than the actual write cycle time). The nominal write cycle speed is 46.2 ns for this example. As with the read cycle, note here that if the write enable pulse requirement was shorter (11 ns rather than 15 ns), the fast Bcache write could be reduced to 6 cycles.

A.5.2 Bcache Miss and External Request

An initial Bcache fill operation is executed when the **21064** attempts to read or write a block that misses in the Bcache (the write fill operation assumes a write-allocate Bcache policy). The miss can be caused for several reasons:

- The Bcache block for that index is not valid.
- The Bcache block for that index is valid, but the tag misses.

The first scenario is the simplest. When a Bcache probe results in a miss, an external READ_BLOCK or WRITE_BLOCK operation is initiated by the **21064** external interface logic. The READ_BLOCK and WRITE_BLOCK external cycles are the most basic method of transferring data between the **21064** and the system, and are discussed in some detail in this appendix.

The command is initiated when the **21064** places the appropriate code on the **cReq_h [2:0]** lines during the rising edge of **sysClkOut1_h**. Timing for external cycles is synchronous to **sysClkOut1_h**, and all setup and hold times are referenced to the rising edge of this clock.

The **21064** control signals change simultaneously with **sysClkOut1_h**, and therefore cannot be sampled on that same edge. In general, this is only a concern for those lines that are used to determine if a cycle should begin, such as the request lines **cReq_h [2:0]** (the **holdAck_h** line is also in this category, as explained later). A delayed version of **cReq_h [2:0]**, perhaps sampled by **sysClkOut2_h**, should be used to feed any state machines that run on **sysClkOut1_h** if they use the request lines.

Note

The signals **adr_h [33:5]**, **data_h [127:0]**, and **check_h [27:0]** are only synchronous to **sysClkOut1_h** during an external cycle. During the time that the **cReq_h [2:0]** field is IDLE, the signals can change without regard to the clocks that drive the external system logic. During the time that the field **cReq_h [2:0]** is *not* IDLE (that is, non-zero), the signals conform to the setup and hold times specified in Chapter 7 of this manual.

The signals **cReq_h [2:0]**, **holdAck_h**, and **cWMask_h [7:0]** are always synchronous to the external system clocks, even during those times when no external cycle is in progress. The signals always conform to the ac specifications in Chapter 7 of this manual.

Figure A–14 External Cycle

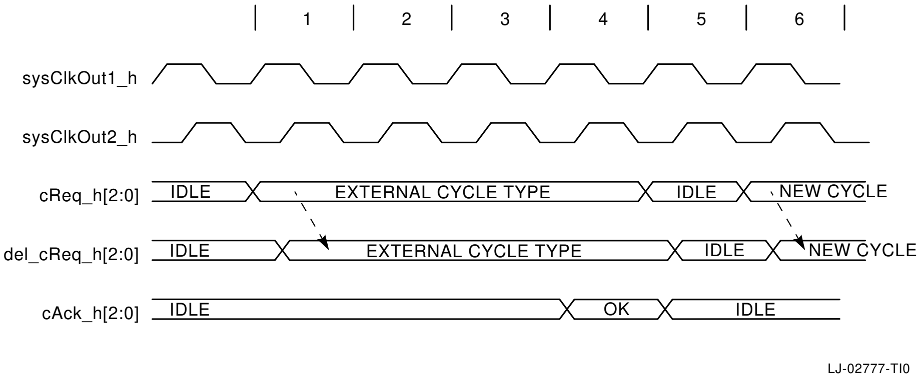


Figure A–14 shows the relationship between the signals and the external system clocks. The **21064** places the external cycle type on the **cReq_h [2:0]** lines at the start of cycle 1 in the figure. **sysClkOut2_h** is used to sample the request lines, and the system logic uses this delayed version to start its state machines at the start of cycle 2. After the external logic has performed the appropriate function, it changes the **cAck_h [2:0]** lines, which are sampled by the **21064** at the start of cycle 5. The CPU removes the request lines at that same time, and could start a Bcache access immediately (at the start of cycle 5). The earliest that the CPU can start another external cycle is one system clock cycle later, at the start of cycle 6 (as shown).

It is important to keep in mind how quickly a Bcache access could begin once the **21064** senses that the external cycle is over. If the external logic is driving the data lines when the **21064** samples the **cAck_h [2:0]** lines and determines that the cycle is over, the **21064** could start its Bcache access by turning on the SRAMs. If the cache RAMs turn on fast enough, and if the system logic continues to drive the data lines too long, there will be contention on the data lines. This effect is worse if the version of **sysClkOut1_h** that is distributed to the system logic is delayed by the version used by the **21064** (many clock buffers have a latency delay associated with them). The designer should ensure that the data lines are not being driven when **cAck_h [2:0]** notifies the **21064** that the external cycle is over.

Note

The Bcache **dataA_h [4]** control line, shared by the external system logic and the **21064**, must also be de-asserted before transferring Bcache control back to the CPU. If this address line is allowed to linger past the time that the **21064** senses the acknowledge on **cAck [2:0]** the next Bcache access by the CPU might return the wrong data.

In the example, the Bcache block is invalid, but the external logic would have no way to know that. So, the external logic must have some way to determine if the current Bcache block occupant needs to be written back to the main memory. One method to do this is to have the system logic perform its own Bcache tag probe. Only the VALID and DIRTY bits need to be inspected, so the external logic probe does not have to wait the entire time necessary to compare the tag address field in the Bcache.

A critical path in this external logic probe is the SRAM output enable circuitry. The **21064** leaves the Bcache RAMs disabled after its own probe, and the external logic must drive the output enable again in order to inspect the VALID and DIRTY bits. One way to do this is to allow an early version of the **cReq_h [2:0]** signals to turn on the SRAM output enables by default, assuming that a probe will be necessary. For those cycles where the external logic later needs to write the Bcache, another logic path is necessary to turn the output enable back off. The de-assertion path is not time-critical, but does need to be implemented for cache fill operations.

Figure A–15 Tag Control Probe Before External Cycle

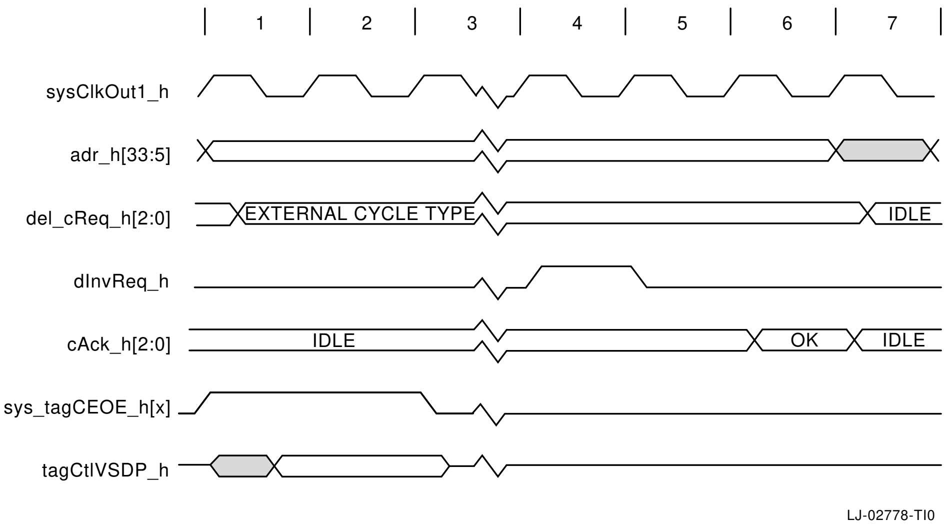


Figure A–15 shows the timing for the entire cycle, including the tag control check. The figure shows the delayed **cReq_h [2:0]** lines changing state at the start of cycle 1, and the tag probe occurring during that cycle. The signal **sys_tagCEOE_h [x]** is the system logic version of the **21064 tagCEOE_h [x]** signal. It is the other input to the NOR gate shown in Figure A–3. That nomenclature is used throughout this appendix.

In this case, the Bcache block is invalid, so no victim write needs to be performed. Section A.5.5 explains the details of a victim write. If the Bcache probe found that the block was valid but not dirty (that is, it had not been modified since being read from main memory), then the outcome is the same. In both cases, the block can safely be invalidated without a victim write.

Figure A–15 shows the tag inspection being implemented in one cycle, so that the read command can start at the beginning of cycle 2. This might not be possible on any particular implementation, and must be carefully analyzed to ensure that the data is stable when the clock asserts in the system control logic.

The external cycle (READ_BLOCK or WRITE_BLOCK) overwrites the data in the Bcache, and asserts the **dInvReq_h** signal if appropriate during the fill, so the internal Dcache block is invalidated later. The lower address bits are directly connected to the **iAdr_h [12:5]** invalidate address input lines in this example; that ensures that the correct cache block is invalidated. Some implementations might require better control over the invalidate bus, and must ensure that the **iAdr_h** lines accurately reflect the lower index value on the asserting edge of **sysClkOut1_h** that samples **dInvReq_h**.

A.5.3 Read Block Request

If the external cycle is a READ_BLOCK, a 32-byte block of memory information is returned to the **21064**. The external logic has complete control of the **21064** interface during the transfer. The data is returned to the **21064** and simultaneously loaded into the Bcache. It is the external logic that writes the data into the Bcache during the read cycle, and *not* the **21064**.

The minimum amount of data that can be written to the Bcache is 32 bytes, but the system logic controls the Bcache until the **cAck_h [2:0]** lines are changed from their IDLE state. As such, it can load and validate more than that if the system designer believes prefetching more blocks is appropriate. Any prefetching must be done in 32-byte increments.

The external logic is responsible for loading the tag address and the tag control fields of the Bcache (with correct parity on both) along with the data. The tag control field should be written as VALID and CLEAN.

Figure A–16 Tag Access and Write Circuit

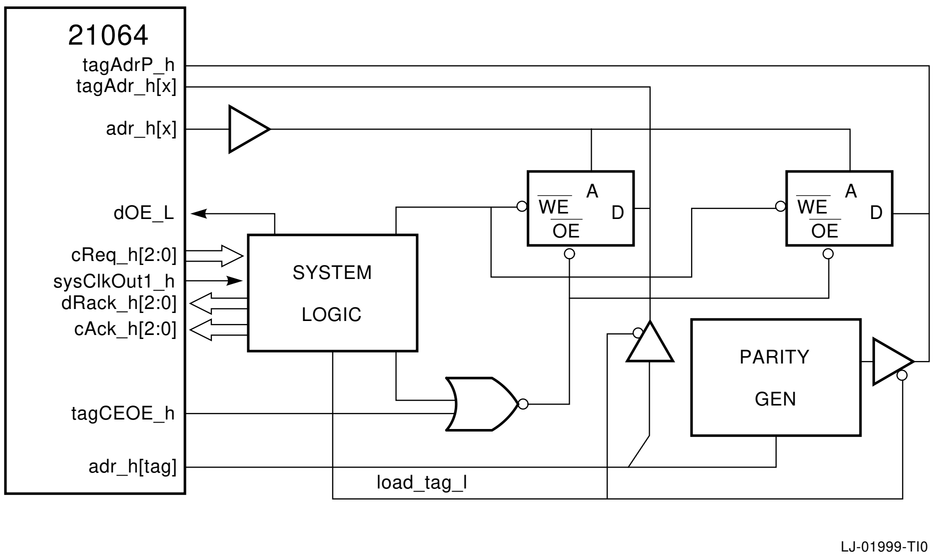
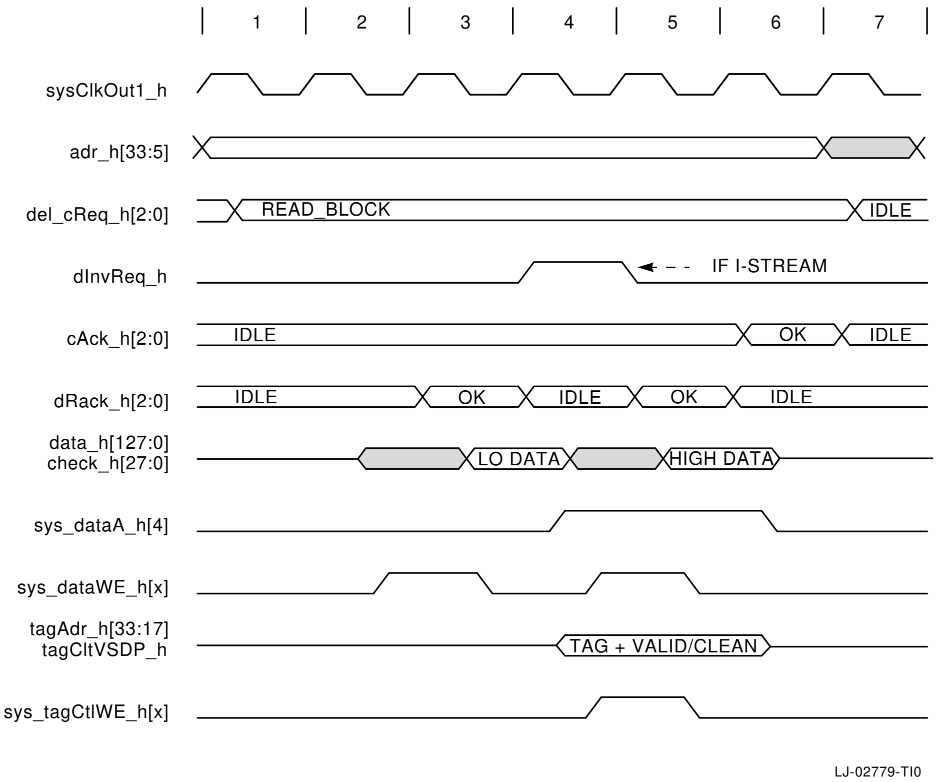


Figure A–16 shows an example of the logic that is expected for tag address control. One of the **tagAdr_h** lines is shown connected to its Bcache RAM. All the tag address lines in use by the implementation go to the parity generator. The **tagAdr_h [tag]** signals and the **tagAdrP_h** parity lines are all driven by tristate buffers. On probe reads, the SRAM output enable allows the Bcache to drive the signals, where they are compared by the **21064** or the system logic. On a fill operation, the **load_tag_l** signal in Figure A–16 causes the Bcache tag RAMs to be loaded with the upper address bits. The **load_tag_l** signal is not part of the **21064** external interface, but rather is a signal created for this particular example to show how the system interface might perform the task. Notice that the RAM write enable input is not connected to the **21064**, since the processor never writes those RAMs.

Figure A–17 Timing Diagram of READ_BLOCK Cycle



As each data word is returned to the **21064**, the **dRack_h [2:0]** field is changed from IDLE to non-IDLE. Normally, the non-IDLE state is OK, which instructs the **21064** to both check the ECC (or parity) on the returned data and cache the data internally. Chapter 6 provides more information on the **dRack_h [2:0]** field. Figure A–17 is a timing diagram for a READ_BLOCK data transfer, showing the **21064** control signals.

Note

All I-stream read accesses (recognized by **cWMask_h [2]** as false during the cycle) must return each **dRack_h [2:0]** with a cached status.

The data in the example is assumed to be ready at the start of cycles 4 and 6, but in another implementation the data might be ready before or after that time. The **dRack_h [2:0]** lines should change to the non-IDLE state whenever the data is ready, with enough setup time so that the lines are sensed by the **21064** at the assertion of **sysClkOut1_h**. The **cAck_h [2:0]** lines can also change to their non-IDLE state (signifying the end of the cycle) during the last **dRack_h [2:0]** data phase if desired. As previously stated, care must be taken to prevent tristate overlap if the **cAck_h [2:0]** lines are asserted coincidentally with the last data **dRack_h [2:0]**.

The timing of the Bcache write signal might be tight in relation to the data arriving from the memory. If the memory is a DRAM array, for example, the CAS signal should be de-asserted as quickly as possible after the DRAM data is stable, in order to start the next memory access during a page mode read. The Bcache, however, might need the data held stable. Using a bidirectional clocked memory data transceiver, as shown in Figure A–2, can help in some cases.

Figure A–17 shows the **dInvReq_h** signal asserting, which invalidates the internal Dcache block corresponding to the lower index bits in the address. This is only needed if the external data fetch is for I-stream (indicated by a false **cWMask_h [2]** on READ_BLOCK cycles), and if the internal Dcache is being kept as a subset of the Bcache. The block that is being filled into the Bcache might be in the Dcache, and it is not otherwise invalidated on an I-stream fetch.

The **21064** can potentially drive its own Bcache control signals a few CPU cycles into the external cycle. As such, the Bcache SRAMs might still be driving the data bus as the external cycle starts. On a read cycle, the system logic might turn on its own data transceivers early in the access, and should be aware that a system cycle should be allowed before this is done. This eliminates any tristate overlap between the SRAMs and the data transceiver.

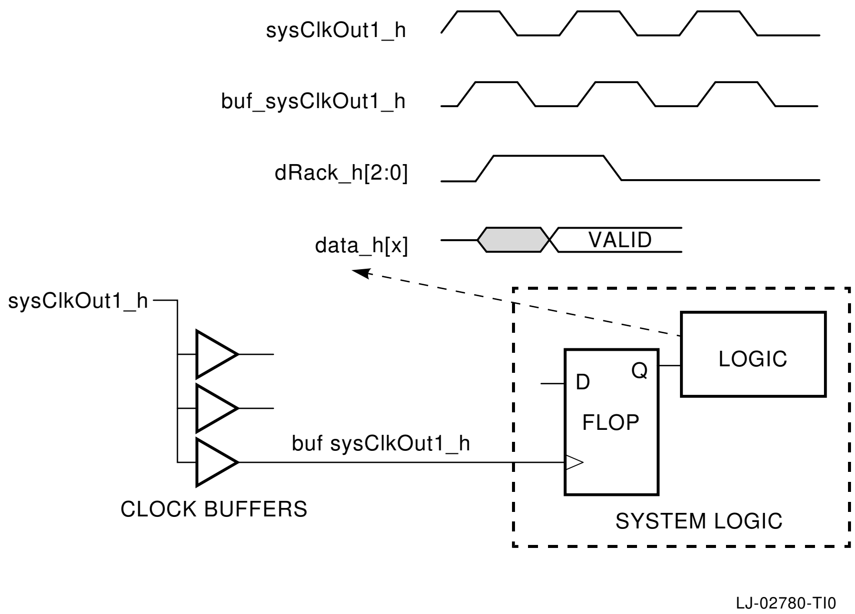
For a system without a Bcache, the **21064** signals would be the same as Figure A–17, but none of the Bcache related lines would be asserted by the system logic. If the system has a Bcache but it is not enabled, the external system logic needs to have some mechanism to turn off the Bcache fill logic, since the **21064** does not broadcast its internal Bcache enable signal to the external interface.

If the read cycle is to an area of memory that has been defined as I/O, it is likely that another bus is involved with the transfer. In this case, the timing is also similar, and the Bcache control signals are also not asserted. A further modification in this case might be to change the **dRack_h [2:0]** field to indicate that no error checking be performed and that the data should not be loaded into the internal chip Dcache.

Note

All I-stream read accesses (recognized by **cWMask_h [2]** being false during the cycle) must return each **dRack_h [2:0]** with a cached status.

Figure A–18 Clock Skew from System to 21064

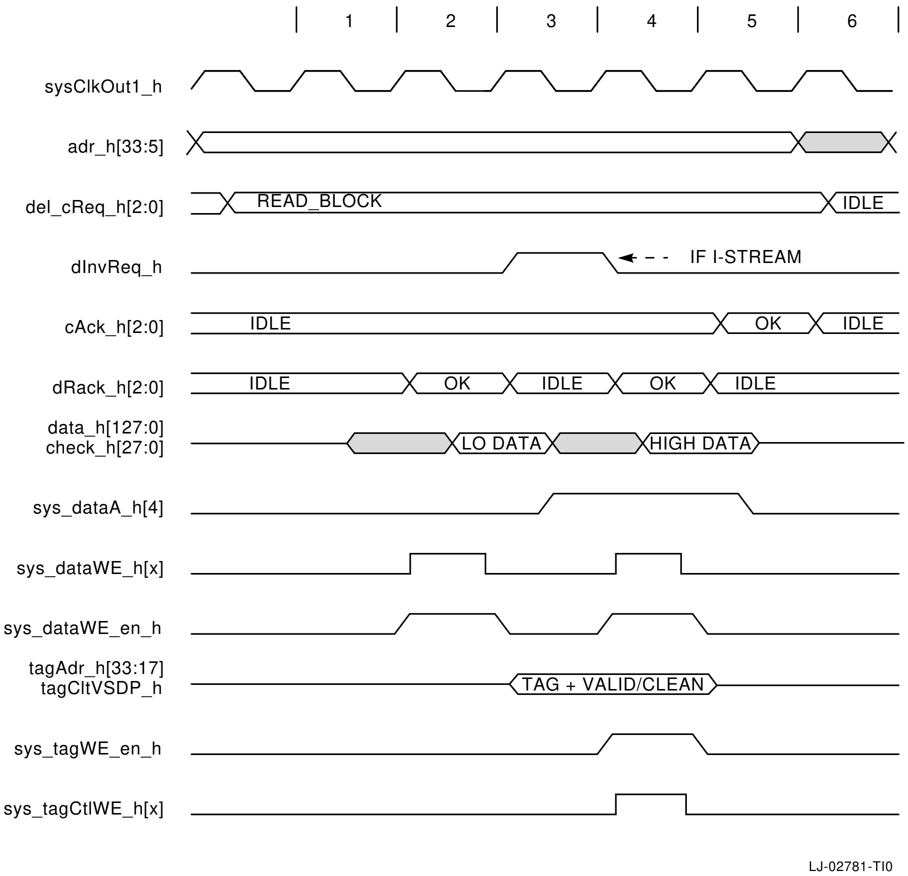


The **21064** system clocks, such as **sysClkOut1_h**, are specified to drive only 40 pF. Because of this, clock buffers are normally used to drive the system logic. The clock buffers might add skew between the **21064** and the system logic. Figure A–18 shows a timing diagram and a small circuit section that might be used to create the signals in the diagram. The buffered version of the system clock **buf_sysClkOut1_h** drives the system state machines that eventually cause the **data_h** lines to be valid at the **21064** input pins.

The **data_h** must be setup at least 3.5 ns before the assertion of **sysClkOut1_h**. In this example, the delay of the buffer must be added to that setup time, since the **21064** sees its reference clock some time before the system logic. This delay should include the entire path for the buffered clocks, including wire delay, device propagation delay, simultaneous switching increases, transmission line effects, and so on. The example in Figure A–18 shows only one instance of this consideration. Others must be analyzed based upon the implementation.

It should be noted here that the skew helps signals like **dRack_h [2:0]** and **cAck_h [2:0]**, since they can be asserted on the system logic version of the clock and meet both the setup and hold times in reference to the **21064**.

Figure A–19 READ_BLOCK Cycle with Write Pulse



If the external timing allows it, a write pulse can be created by delaying **sysClkOut2_h** by 1 CPU cycle, and by using **sysClkOut1_h** to create an enable signal for it. Figure A–19 shows a READ_BLOCK cycle with a cache fill that uses a write pulse to load the Bcache. The signal **sys_dataWE_en_h** enables **sysClkOut2_h** when the Bcache needs to be written.

Figure A–20 Write Pulse Circuit

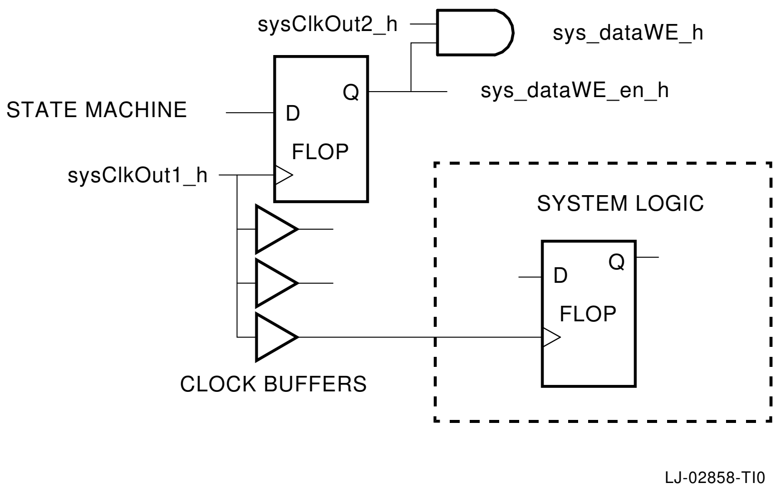


Figure A–20 is an example of how the write pulse can be created, showing the circuit paths of interest. The clock buffers are shown that are expected to drive the system logic, in part to show that skew must be carefully considered if a write pulse-like scheme is attempted. If the clock buffers add enough delay to the path, and the delayed version of the clock is used to create the **sys_dataWE_en_h** signal, the leading edge of the enable can overlap with **sysClkOut2_h**. To prevent this from happening, a non-buffered version of **sysClkOut1_h** might be used to create **sys_dataWE_en_h**. The same argument applies to the tag control write pulse.

A.5.4 Write Block Request

If the external cycle is a WRITE_BLOCK, the system logic must perform a different set of functions. The initial tag probe must still be done by the external logic, and it is still assumed here that the current block is either not valid, or it is valid but not dirty (no victim write needed).

If it is assumed that the Bcache is used as a writeback cache (the normal mode), and that the design is using a write-allocate Bcache policy, then the write data should go into the Bcache, even though an external WRITE_BLOCK cycle is being executed. The most reasonable way to accomplish this is to read the entire block from memory into the Bcache, then write the masked 8-byte into that same Bcache block. For systems without a Bcache, the external memory should be writable on 4-byte (32-bit) boundaries, since the Bcache merge could not then be performed.

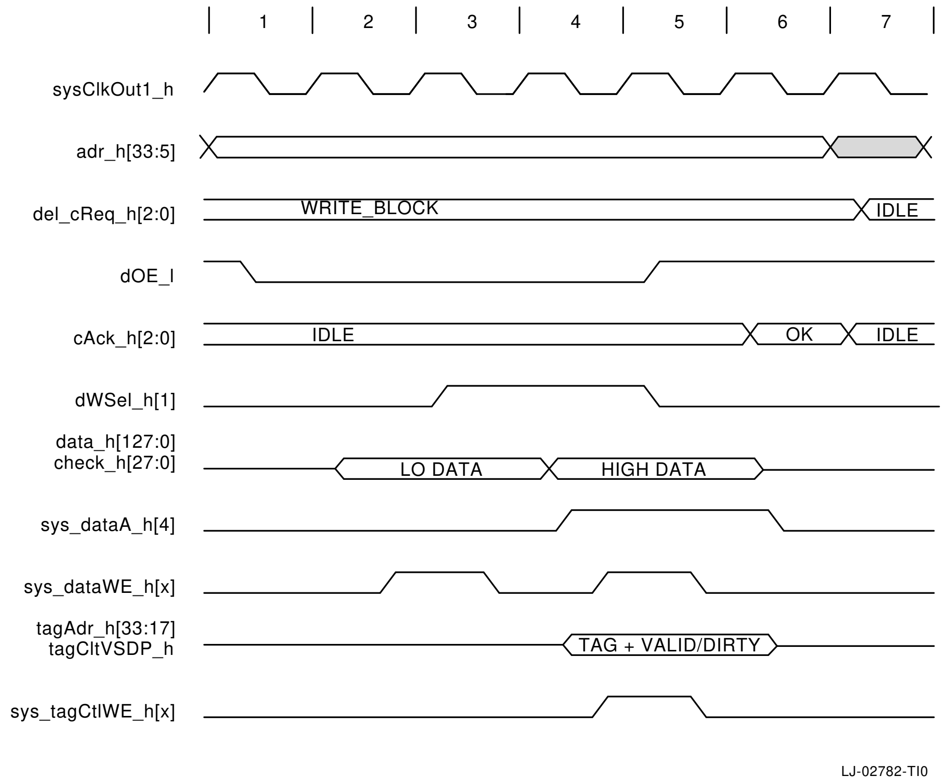
The **21064** is attempting to perform a WRITE_BLOCK cycle in this case, and doesn't even know about the memory read cycle. The **dRack_h [2:0]** and **cAck_h [2:0]** signals should remain IDLE throughout the read transfer. During the fill operation, the **dInvReq_h** signal should be asserted if the old Bcache block was valid, since the internal Dcache might also have the replaced block valid. As a practical matter, the entire Bcache is valid shortly after system initialization, so every read fill on behalf of a write cycle must assert **dInvReq_h**, unless a Dcache backmap can be consulted to determine the block's validity.

After the read has been accomplished and the main memory data has been placed in the Bcache block, the system logic should cycle the **21064** through its write data by using the **dWSel_h [1]** line. The **21064** input signal **dOE_l** is used to instruct the chip to drive the data lines for the write portion of the cycle. Only the masked 4-byte segments should have their write enable inputs asserted during the cycle, based upon the **cWMask_h [7:0]** signals. The lower 128 bits of data (during which **dataA_h [4]** is low) are controlled by **cWMask_h [3:0]**, and the upper 128 bits of data (during which **dataA_h [4]** is asserted high) are controlled by **cWMask_h [7:4]**. Within each data section, the lower bits in the **cWMask_h** field control the lower 4-byte segment. For example, **cWMask_h [0]** controls bits [31:0], **cWMask_h [1]** controls bits [63:32], and so on.

Note

The signal **dWSEL_h** must not be asserted during the same cycle that **cAck_h** notifies the **21064** that the external access is over.

Figure A–21 Timing Diagram of WRITE_BLOCK Cycle



After the entire read and write cycle has finished, the tag control should be written as VALID and DIRTY, and the tag address should be written with the correct upper address bits. Figure A–21 shows the Bcache write portion of the WRITE_BLOCK cycle. The read portion looks like Figure A–17, except that the CPU acknowledge signals should not be changed from IDLE, and the **dInvReq_h** signal should be asserted.

Note when the data actually changes relative to the signals **dWSel_h** and **dOE_l**. All the signals are synchronous to the leading edge of **sysClkOut1_h**, so the inputs are not acted upon until the next system clock edge. The end of the external write in Figure A–21 is the start of cycle 7, at which time the **21064** removes the address and potentially starts the next Bcache probe.

Note

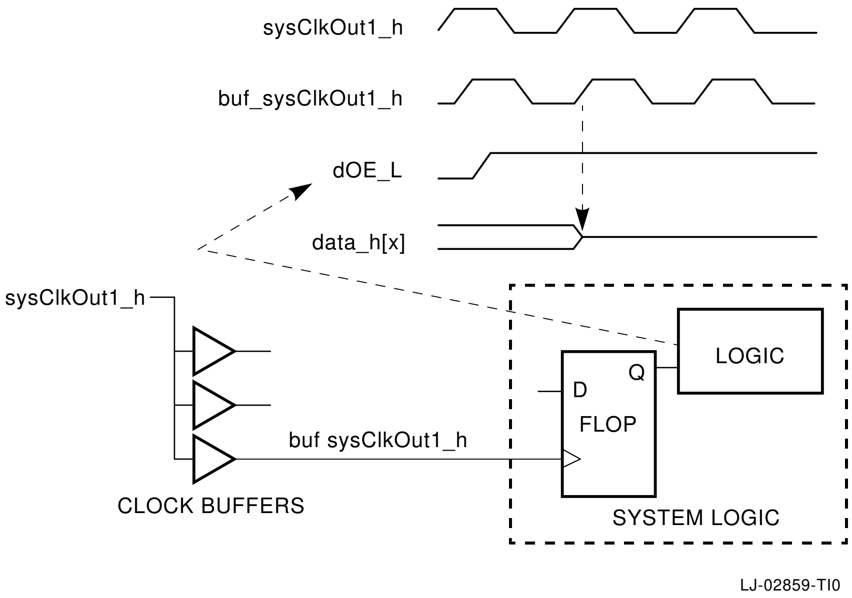
The signals **adr_h [33:5]**, **data_h [127:0]**, and **check_h [27:0]** are only synchronous to **sysClkOut1_h** during an external cycle. During the time that the **cReq_h [2:0]** field is IDLE, the signals can change without regard to the clocks that drive the external system logic. During the time that the field **cReq_h [2:0]** is not IDLE (that is, non-zero), they conform to the setup and hold times specified in the ac specifications in Chapter 7 of this manual.

The signals **cReq_h [2:0]**, **holdAck_h**, and **cWMask_h [7:0]** are always synchronous to the external system clocks, even during those times when no external cycle is in progress. They always conform to the ac specifications in Chapter 7.

There are several techniques that can be used on the write cycle:

- The **cWMask_h [7:0]** signals can be inspected, and if they are asserted the read portion of the cycle does not have to be performed. In this case, every byte is written anyway, so the Bcache write cycle can be performed from the start. If this optimization is taken, it is important that any necessary functions normally performed during a read fill are still performed. For example, the tag address and control SRAMs must still be loaded with the new block address. If this is normally done during the read fill, that function must be duplicated for this situation. Also, the internal Dcache invalidate signal **dInvReq_h** must be asserted if the current Bcache block is being replaced. This function might also be performed during a read fill, and needs to be duplicated here when appropriate.
- The tag Bcache RAMs don't have to be written on both the read and write portions of the cycle. It may be easier to do it during the read cycle so that it is the same as a normal read. The same caveats apply to this optimization as the last one. If the Bcache tag SRAMs are loaded only during the read, then the read fill cannot be eliminated without adding that function to the write as well.
- Both 128-bit data segments don't need to be written if the lower mask bits show that there are no 4-byte segments enabled. The signal **dWSel_h [1]** can be asserted earlier to write the upper 128 bits only. If both segments *are* written, however, the lower address must be written before the upper address (as shown in Figure A–21).

Figure A–22 Clock Skew from System to 21064 for Write



As with the read cycle, the write cycle must take into account the clock skew between the **21064** and the system logic. Figure A–22 shows an example of a potential problem. The **21064** signal **dOE_l** is asserted by the system logic to instruct the chip to drive the **data_h** lines during the write cycle. But **dOE_l** is sampled by the chip on the earlier, unbuffered version of **sysClkOut1_h**. In the figure, the data is removed on the asserting edge of **sysClkOut1_h**, which might be too soon. If the system logic uses its version of **buf_sysClkOut1_h** to sample the write data, then it should cause **dOE_l** to remain asserted low one extra cycle to accommodate the clock skew. This same argument applies to **dWSel_h [1]**.

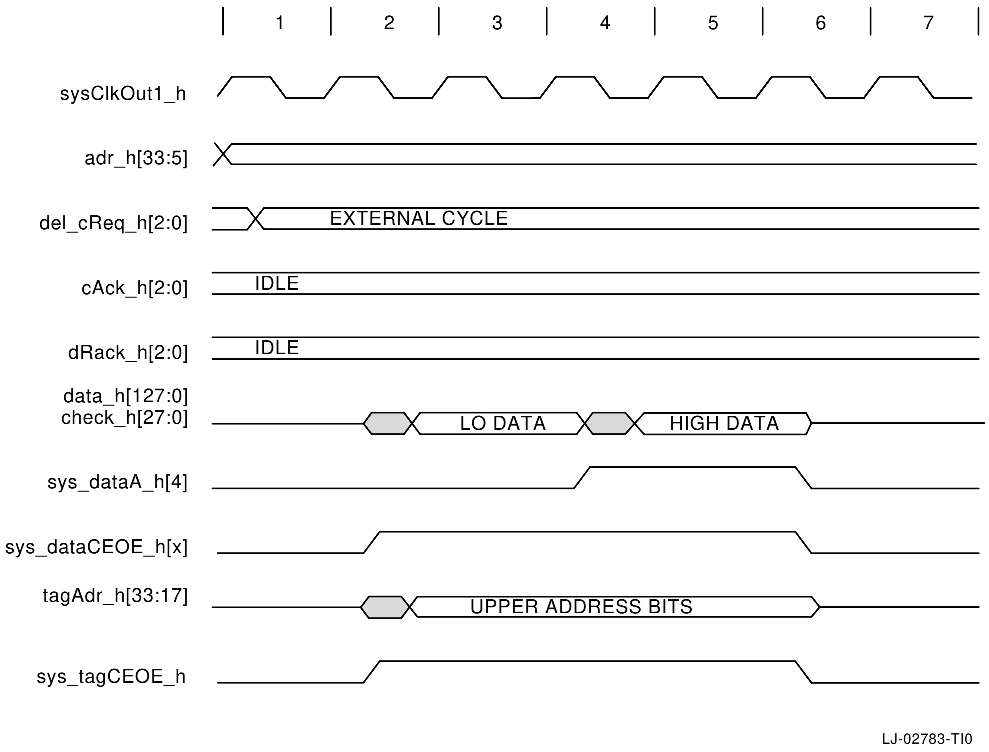
Note

On a Bcache probe and miss, control is not passed to the system logic instantaneously. The **21064** de-asserts its Bcache output enable signals at least 1 CPU cycle before it begins an external write cycle. The path from the **21064** to the SRAMs should be kept as short as possible to minimize the chance that the SRAMs are still driving their outputs onto the data lines when the **21064** turns on its own data drivers for the write cycle. This is discussed in more detail in the application note *Designing a Memory/Cache Subsystem for the DECchip 21064 Microprocessor.*

A.5.5 Victim Write

The second possibility for the original Bcache miss is that the data currently occupying the Bcache block is VALID and DIRTY, but the upper address bits do not match the tag address. The **21064** activates the external logic with a READ_BLOCK or WRITE_BLOCK, just as in the previous description. When the external logic does the Bcache VALID/DIRTY probe, however, the outcome is different. Since the data in the Bcache block has been modified since it was read from the main memory, it must be written back to memory before the external read or write cycle can continue. The act of writing the block back to memory is called a victim write.

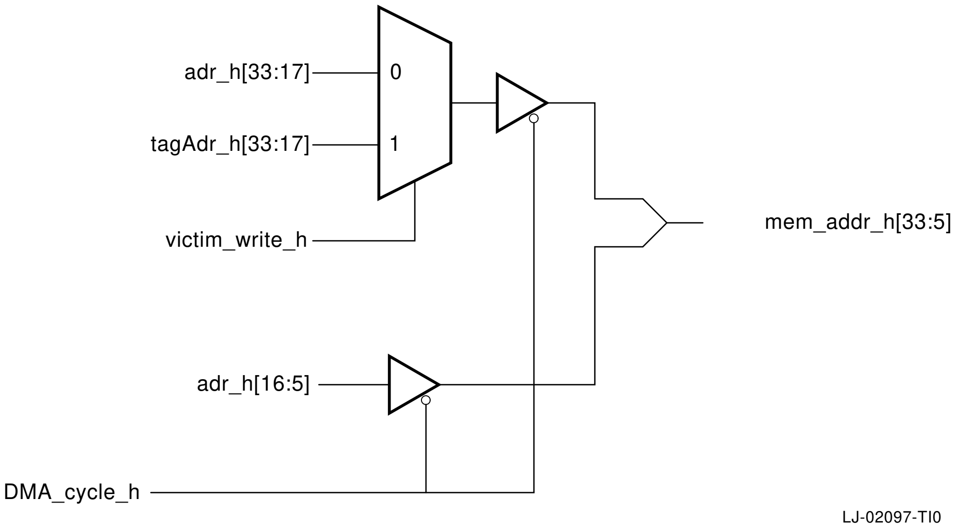
Figure A–23 Timing Diagram of Victim Write Cycle



The external control logic for a victim write is straightforward. The 128-bit data segments are read from the Bcache, and the data is sent to the external memory. After the victim is safely back in memory, the READ_BLOCK or WRITE_BLOCK is performed, exactly as described in the previous sections. Some time during the entire cycle (including the victim write and subsequent read or write cycle), the internal Dcache line must be invalidated, or updated with the new tag and data information. On a D-stream read cycle, this happens during the read fill operation. On I-stream read cycles and on read fill operations on behalf of write cycles, the **dInvReq_h** signal should be asserted to invalidate the internal Dcache block for that index.

Figure A–23 shows the victim write cycle. The tag address is used as the high memory address bits for the write, so the **sys_tagCEOE_h** signal is asserted to enable their outputs. The Bcache data RAMs are enabled, and each data segment is selected in turn by **sys_dataA_h [4]**. In this example, two cycles are necessary for the main memory to be written. If the memory is slower, more cycles should be allocated. At the start of cycle 7, the actual read or write cycle proceeds.

Figure A–24 Address MUX for Victim Write



A MUX gate is needed to choose between the normal **21064** memory write and the victim write, where the upper address bits are taken from the tag field of the Bcache. Figure A–24 shows the expected circuit. Normally, the MUX selects the **adr_h** lines, but during victim write cycles the **tagAdr_h** lines are chosen as the memory address. The figure also shows that the entire address bus should have the ability to tristate for DMA access. During DMA transfers, the **21064** is forced off the address lines, and the external logic controls the entire address. The MUX and tristatable gate can be one physical device.

The signals **victim_write_h** and **DMA_cycle_h** are expected to be created by the system logic. They do not originate from the **21064**. The tag address field in Figure A–24 is shown for the smallest Bcache size. Other Bcache sizes have different relative widths for the tag and index fields.

For high performance systems, a victim queue (or silo) is an option. Instead of writing the victim and reading the new data word serially, the Bcache and the memory can be read simultaneously. The information in the Bcache can be stored in a silo while the memory data is loaded into the Bcache. The silo can then be used to write the previous Bcache contents to memory. This has the effect of reducing the read latency on the miss and subsequent fill operation.

A.5.6 Non-Cached Memory Write

There might be non-cached memory space in your **21064** system design. When that area is a write target, the data should bypass the Bcache and be written directly to the system memory. If non-cacheable memory is included in the system, it is best to make it writable on 4-byte segments. Otherwise, a full read/modify/write cycle is needed to store non-fully masked data.

A memory write on a system that allows masking on 4-byte segments is only a minor variant on the victim write function. The difference is that the information to be written to memory is coming from the **21064** rather than the Bcache. The Bcache is not invoked at all in this situation, and the **dWSEl_h [1]** signal is used to instruct the **21064** which 128-bit data segment to provide.

Figure A–25 Timing Diagram of Direct Memory Write Cycle

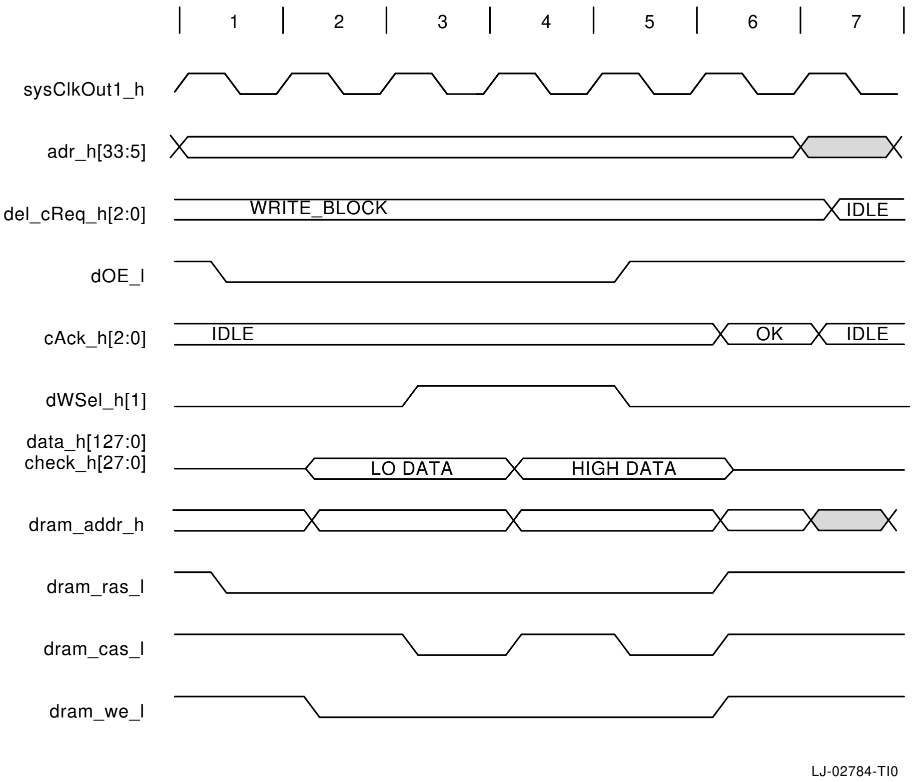


Figure A–25 shows the timing for such a write cycle. In this example, a more complete memory control flow is shown. The memory is a DRAM array, and a representative set of memory control signals are provided. The designer should work out the exact timing on a particular implementation in order to ensure that the memory parts are accessed within specification.

The **adr_h** lines should be stable at the start of the cycle, since they are changed by the **21064** before the cycle is started. If the DRAM address MUX points to the row address by default, the memory control can assert RAS at the start of the cycle. At the end of the cycle, the DRAM RAS precharge time must be accounted for. The **21064** allows at least one idle cycle after it senses **cAck_h** as non-IDLE before it starts the next external command (although a Bcache cycle can proceed immediately). In the example, RAS de-asserts at the start of cycle 6, which means that it cannot re-assert until the start of cycle 8. The changing of **cAck_h** so that it is sensed at the start of cycle 7 meets the RAS precharge time for the part in this implementation.

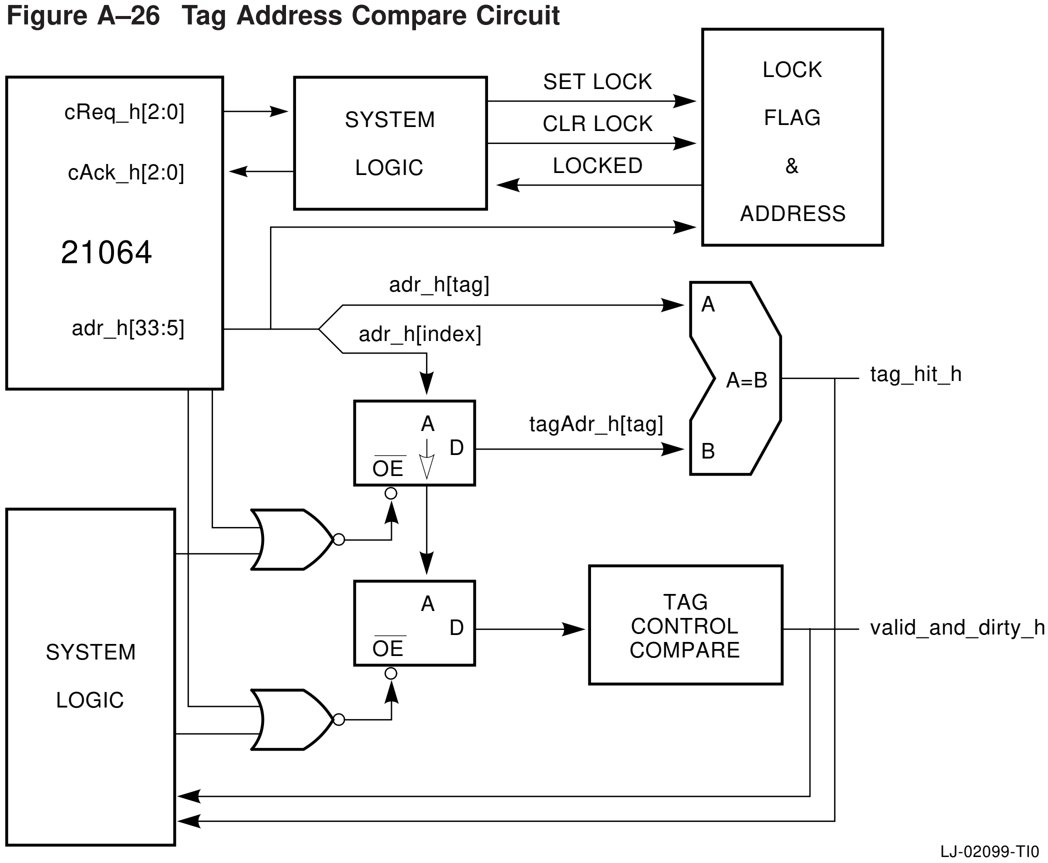
A.6 Load Locked and Store Conditional

The **21064** provides the ability to perform locked memory accesses through the LDxL (Load_Locked) and STxC (Store_Conditional) cycle command pair. The LDxL command forces the **21064** to bypass the Bcache and request data directly from the external memory interface. The memory interface logic must set a special interlock flag as it returns the data, and may optionally keep the locked address.

The data requested for the LDxL access might be in the Bcache, since it has not been probed, so the external memory logic must do its own probe to determine where to obtain the information. In previous descriptions, the system logic only had to probe the tag control VALID and DIRTY RAMs to determine if a victim write was necessary. For the LDxL and STxC probe, the entire tag address must be compared, since the data that is being accessed might be in the Bcache.

Figure A–26 shows a diagram of the probe and compare logic. On the initial request (the **cReq_h[2:0]** lines specify that the external LDxL must be performed), the system logic enables the tag RAMs and compares them to the tag field of the address for the **21064**. If they compare and the block is valid, the data requested is already in the Bcache. If the tag compare also shows that the block is dirty, then the *only* place the data resides is in the Bcache.

Figure A–26 Tag Address Compare Circuit



There are two choices:

- The data can be accessed from the Bcache.
- The data can be written back to memory, then accessed from there.

If the tag compares and the block is valid, but it is not dirty, then both the Bcache and the memory contain the data. It can be accessed from either place. If the tag fails or the block is *not* valid, then the data is only available from memory and must be accessed from there. In all the above cases, a flag must be set that signifies the location is locked.

Every design needs to provide a lock flag, but the amount of address information latched is completely up to the designer. On a uniprocessor system that does not expect much lock contention, simply having the lock flag with no address information might be enough. If any device accesses a memory location, the flag can be cleared, which causes the subsequent store cycle to fail. On a multiprocessor system that expects real lock contention, lock address information can be saved so that different processors can lock different areas.

The *Alpha Architecture Reference Manual* discusses the guidelines that pertain to the lock and its associated address information.

The store_conditional instruction is executed by the **21064** to clear the lock (and to find out if the code that was executed did so without contention). It is a write-type request, where the processor again bypasses the Bcache without a probe. If no other access has been made to the locked data, the STxC is treated similarly to a regular external memory write, though the Bcache must be probed by the system logic to determine where the most up-to-date data is located. The locked flag is also cleared.

If the Bcache probe finds that the data is both valid and dirty, the choices are similar to the read case:

- The data can be written into the Bcache, using the **cWMask_h [7:0]** to determine which 4-byte segments should be modified. The STxC command never validates more than a single 4-byte or 8-byte segment of data, and this can be used to optimize the cycle if desired.
- The data can be written back to memory with a victim write, and modified there.

If the locked flag is cleared before the start of the STxC cycle, meaning that the data location has been written between the LDxL and STxC commands, the external memory logic must return a special acknowledge code that notifies the **21064** of this fact. In this case, no Bcache probe or actual external cycle needs to be performed.

A.7 Special Request Cycles

There are some external request cycles that might not actually perform any work, but must still provide the **21064** with an acknowledge. BARRIER, FETCH, and FETCH_M cycles are described in the hardware reference manual, and perform a system-specific function. When they are sensed by the external control logic, the system must minimally acknowledge on **cAck_h** with an OK code.

Note

The address that accompanies the BARRIER cycle (invoked by the MB instruction) is undefined in the Alpha AXP architecture, and thus not under program control in the **21064**. As such, a decode of the BARRIER instruction *must* not include any conditional filtering based on a particular memory or I/O address range, since the BARRIER address can take any value.

A.8 DMA Access

There are situations where a device connected to an I/O bus needs direct access to the **21064** cache/memory subsystem. In the most general case the data could be in the Bcache, which is described in this section.

There are several ways that the external logic can perform a DMA access, the most straightforward of which is the use of the **holdReq_h** line. When a DMA device requires access to the **21064** cache/memory subsystem, it can notify the chip of that fact by asserting the **holdReq_h** signal. The **21064** replies to this request by asserting the **holdAck_h** signal. This signifies that the **21064** is no longer asserting the address, data, or Bcache control signals. The entire memory subsystem and Bcache are now under control of the external system logic.

The signal **holdAck_h** changes simultaneously with **sysClkOut1_h**. As such it should be sampled on an edge other than **sysClkOut1_h** if used as an input into state machines that run on **sysClkOut1_h**. This is similar to how **cReq_h [2:0]** must be used, as shown in Figure A–14.

If it is assumed that the DMA target data (read or write) might be in the Bcache, the external logic must do a Bcache probe. This is similar to the probe necessary to determine if the data is in the Bcache when a LDxL or STxC is executed. The tag address and control RAMs should be compared to determine if the requested data is in the Bcache, and if it is dirty. The DMA logic can use the LDxL/STxC compare logic shown in Figure A–26, or it can duplicate that logic for its own comparison.

The **21064** provides a third option for the tag address comparison, and this is the **tagEq_l** signal. When the chip is in **holdReq_h** mode, the **adr_h [33:5]** signals become inputs. The DMA device can drive its address on those lines and simultaneously enable the tag address RAMs. If the tag address compares with good parity, the signal **tagEq_l** will be asserted low.

For DMA read cycles where the probe shows that the data is valid in the Bcache, the choices are similar to what they were for the LDxL/STxC probe. If the data is valid but not dirty, it can be accessed from wherever it is most convenient. If the data is valid and dirty, it can be accessed directly from the Bcache or written back to memory and accessed there.

For a DMA write that hits in the Bcache, there are several choices:

- The data can be written directly into the Bcache with the correct ECC or parity. In this case, the tag control should be made DIRTY, and the **dInvReq_h** signal should invalidate the cache line in the internal Dcache.
- The data can be written back to memory with a victim write, and it can be modified there. The **dInvReq_h** signal should be asserted during the victim write or the DMA memory write to invalidate any stale Dcache data.

If the Bcache probe misses, or if the DMA access is defined to be only in the memory, then it is most sensibly accessed or modified there.

After the read or write cycle is complete, the **holdReq_h** signal can be de-asserted, which causes the **21064** to de-assert the **holdAck_h** signal. The **21064** then takes control of the bus again, after a short delay.

There is one subtlety that should be mentioned here in regard to DMA access design. The **21064** might be in the middle of its own external (non-Bcache) access when it receives the **holdReq_h** request signal. If this happens, the chip might be waiting for data, and has really only stalled the external cycle. As such, the data and cycle acknowledge signals are ‘‘live.’’ The external logic must be careful not to assert the **dOE_l**, **dWSEL_h**, **dRack_h**, or **cAck_h** signals during its access cycle. Furthermore, there is a 2-CPU cycle delay between the time that the **21064** de-asserts the **holdAck_h** signal and when it re-enables its own address and data lines. This must be factored into the external logic for cycles that continue after the DMA stall.

To simplify the design, it is possible to filter the **holdReq_h** signal going to the **21064**. If the external logic ensures that the **holdReq_h** signal only gets to the **21064** between cycles, then the problem of external cycles stalling in the middle is eliminated.

A.9 Backmapping the Internal 21064 Dcache

The **21064** provides the ability to keep a ‘‘backmap’’ of the internal Dcache tag address in external logic. In effect, the module adds enough extra information about the Dcache tag address to filter the invalidates that are sent to the **21064** Dcache. This can be used in multiprocessor systems or to filter DMA writes.

The processor outputs the signal **dMapWE_h** when it loads a block into the Dcache. This is meant to control an external memory array that takes the address from the appropriate **adr_h** lines and updates the external tag address memory location.

The external tag address does not have to contain the entire Dcache tag field, but rather needs only the difference between the Bcache and Dcache tag widths. If the Dcache is being kept as a subset of the Bcache, and if the Bcache is first probed, then the Dcache backmap is only responsible for knowing if a Bcache hit is also a Dcache hit.

A.10 I/O Interface

The input/output function of the **21064** is in some ways a subset of the memory function. I/O is normally not cached, so the probe misses, or is not performed at all, for that memory quadrant. The access goes directly to the external interface bus as a READ_BLOCK or WRITE_BLOCK.

On a read cycle, the data is returned as in the memory access already described, with the **dRack_h [2:0]** signals indicating that the data should be neither error-checked nor cached inside the chip. Since the return data is under complete control of the system interface logic, the Bcache is not filled. On a write cycle, the steps are similar to a direct memory write cycle. The external logic can take the appropriate number of data words, then acknowledge the cycle.

The Alpha architecture provides an approach to I/O called a ‘‘mailbox.’’ A description of the read or write is set up in memory. The description includes the full address, data, and mask information. A special mailbox register is then accessed to invoke the I/O transaction. This approach implies a smart I/O controller, and allows access to the full address range of the I/O bus.

If the mailbox option is not implemented, there are some techniques that can be employed when interfacing the **21064** to an I/O bus:

- Address or data bits can be used to create byte masks and encode system level functions.
- The **21064** address lines **adr_h** can be shifted right when accessing external buses that need the lower address bits. So, for example, **adr_h [20:5]** can translate to I/O address bits [15:0].
- Reads and writes to I/O space can use the low bytes for all transactions, rather than pack the data into the appropriate field within the 32-byte block.
- The **cWMask_h** field can normally be ignored for I/O writes.

B
Technical Support, Ordering, and Associated Literature

This appendix describes how to obtain DECchip information and technical support, as well as how to order DECchip products and associated literature.

B.1 Calling the DECchip Information Line for Information and Technical Support

Use the following numbers to call the DECchip Information Line:

United States and Canada	**1–800–332–2717**
TTY (United States only)	**1–800–332–2515**
Outside North America	**+1–508–568–6868**

B.2 Ordering DECchip Products

To order the DECchip 21064 and DECchip 21064A Alpha AXP microprocessors and evaluation boards, contact your local Digital sales office. Your sales representative may be able to help you take advantage of discounts and volume pricing.

You can order the following DECchip products from Digital.

Product	Order Number
DECchip 21064A–233 Microprocessor	21064–BB
DECchip 21064A–275 Microprocessor	21064–DB
DECchip 21064–150 Microprocessor	21064–AA
DECchip 21064–166 Microprocessor	21064–CA
DECchip 21064–200 Microprocessor	21064–BA

Product	Order Number
DECchip 21064 Evaluation Board 150 MHz Kit (Supports OSF/1 and Windows NT operating systems.)	21A01–03
DECchip 21064 Evaluation Board Design Package	21A01–13
Heat Sink Assembly	2106H–AA

B.3 Ordering DECchip Sample Kits

To order a DECchip 21064 Sample Kit, which contains one DECchip 21064, one heat sink, and supporting documentation, call **1–800–DIGITAL**. You will need a purchase order number or credit card to order the following DECchip products.

Product	Order Number
DECchip 21064–150 Sample Kit	21064–SA
DECchip 21064–166 Sample Kit	21064–SC
DECchip 21064–200 Sample Kit	21064–SB

B.4 Ordering Associated DECchip Literature

The following table lists some of the DECchip literature which is available. For a complete list and information about ordering, contact the DECchip Information Line.

Title	Order Number
DECchip 21064/21064A Literature	
Alpha Architecture Reference Manual[1]	EY–L520E–DP–YCH
DECchip 21064A–233, –275 Alpha AXP Microprocessor Data Sheet	EC–N0699–72
DECchip 21064–150, –166, –200 Alpha AXP Microprocessor Data Sheet	EB–N0136–72
PALcode for Alpha AXP Microprocessors System Design Guide	EC–N0660–72

[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.

Title	Order Number
DECchip 21064/21064A Literature	
Designing a Memory/Cache Subsystem for the DECchip 21064 Microprocessor: An Application Note	EC–N0301–72
Designing a System with the DECchip 21064 Microprocessor: An Application Note	EC–N0107–72
DECchip 21064 Evaluation Board Literature	
Calculating a System I/O Address for the DECchip 21064 Evaluation Board: An Application Note	EC–N0567–72
DECchip 21064 Bus Transactor User's Guide	EC–N0448–72
Alpha AXP Evaluation Board Debug Monitor User's Guide	EC–N0661–72
DECchip 21064 and DECchip 21064A PCI Evaluation Board Design Package Read Me First	EC–N0748–72
DECchip 21064 and DECchip 21064A PCI Evaluation Board Product Brief	EC–N0639–72
Alpha AXP SROM Mini-Debugger User's Guide	EC–N0750–72
DECchip 21064 and DECchip 21064A PCI Evaluation Board User's Guide	EC–N0640–72
Alpha AXP Evaluation Board Software Design Tools User's Guide	EC–N0662–72
DECchip Marketing Literature	
AMP PGA Burn-In and Product Sockets for DECchip 21064 with Alpha AXP Architecture AXP	EC–X3013–72
DECchip Information Line Brochure	EB–N0109–72
DECchip Preprocessor for Hewlett-Packard Logic Analyzer	EC–X2454–72
DECchip Sample Kit Brochure	EB–N0108–72

Glossary

Abort

The unit stops the operation it is performing, without saving status, and begins to perform some other operation.

Abox

This section of the processor unit performs address translation, interfaces to the pin bus, and performs several other functions. Also called load/store unit.

Aligned

A datum of size 2**N is stored in memory at a byte address that is a multiple of 2**N (that is, one that has N low-order zeros).

ANSI

American National Standards Institute, an organization that develops and publishes standards for the computer industry.

ASM

address space match—defined by Alpha AXP architecture

ASN

address space number—defined by Alpha AXP architecture

Assert

To cause a signal to change to its logical true state.

AST

See asynchronous system trap.

Asynchronous System Trap (AST)

A software-simulated interrupt to a user-defined routine. ASTs enable a user process to be notified asynchronously, with respect to that process, of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes execution of the process at the point where it was interrupted.

autoboot

The process by which the system boots automatically.

Backmap

A memory unit which is used to note addresses of valid entries within a cache.

Bandwidth

Bandwidth is often used to express ‘‘high rate of data transfer’’ in an I/O channel. This usage assumes that a wide bandwidth may contain a high frequency, which can accommodate a high rate of data transfer.

Barrier Transaction

A transaction on the external interface as a result of an MB instruction.

Bcache

A second, very fast memory that is used in combination with slower large-capacity memories.

bht

See branch history table.

Bidirectional

Flowing in two directions. The buses are bidirectional; they carry both input and output signals.

Bit

Binary digit. The smallest unit of data in a binary notation system, designated as 0 or 1.

BIU

See Bus Interface Unit.

Block Exchange

Memory feature that improves bus bandwidth by paralleling a cache victim write-back with a cache miss fill.

Boot

Short for bootstrap. Loading an operating system into memory is called booting.

Branch history table

A table in the Icache that has an entry associated with each instruction. The entry has one bit for the **21064** and two bits for the **21064A**. The entry is used by the 21064/21064A when predicting branch action.

Buffer

An internal memory area used for temporary storage of data records during input or output operations.

Bugcheck

A software condition, usually the response to software's detection of an ''internal inconsistency,'' which results in the execution of the system bugcheck code.

Bus

A group of signals that consists of many transmission lines or wires. It interconnects computer system components to provide communications paths for addresses, data, and control information.

Bus Interface Unit

Logic unit which provides 21064 processor with interface to pin bus. The bus interface unit is a part of the Abox.

Byte

Eight contiguous bits starting on an addressable byte boundary. The bits are numbered right to left, 0 through 7.

byte granularity

Memory systems are said to have byte granularity if adjacent bytes can be written concurrently and independently by different processes or processors.

Cache

See Cache memory.

Cache block

The fundamental unit of manipulation in a cache. Also known as cache line.

Cache hit

The status returned when a logic unit probes a cache memory and finds a valid cache entry at the probed address.

Cache Interference

The result of an operation that adversely affects the mechanisms and procedures used to keep frequently used items in a cache. Such interference may cause frequently used items to be removed from a cache or incur significant overhead operations to ensure correct results. Either action hampers performance.

Cache line

The fundamental unit of manipulation in a cache. Also known as cache block.

Cache Line Buffer

A buffer used to store a block of cache memory.

Cache memory

A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes.

Cache miss

The status returned when a logic unit probes a cache memory and does not find a valid cache entry at the probed address.

CALL_PAL Instructions

Special instructions used to invoke PALcode.

Central Processing Unit (CPU)

The unit of the computer that is responsible for interpreting and executing instructions.

CISC

Complex instruction set computer. An instruction set consisting of a large number of complex instructions that are managed by microcode. *Contrast with* RISC.

Clean

In the cache of a system bus node, refers to a cache line that is valid but has not been written.

Clock

A signal used to synchronize the circuits in a computer

CMOS

Complementary metal-oxide semiconductor. A silicon device formed by a process that combines PMOS and NMOS semiconductor material.

Conditonal Branch Instructions

Instructions that test a register for positive/negative or for zero/nonzero. They can also test integer registers for even/odd.

Control and Status Register (CSR)

A device or controller register that resides in the processor's I/O space. The CSR initiates device activity and records its status.

CPU

See central processing unit.

CSR

See control and status register.

Cycle

One clock interval.

Data Bus

The bus used to carry data between the 21064 and external devices. Also called the pin bus.

Dcache

Data cache. A cache reserved for storage of data. The Dcache does not contain instructions.

Direct-mapping Cache

A cache organization in which only one address comparison is needed to locate any data in the cache, because any block of main memory data can be placed in only one possible position in the cache.

Direct Memory Access (DMA)

Access to memory by an I/O device that does not require processor intervention.

Dirty

One status item for a cache block. The cache block is valid and has been written so that it may differ from the copy in system main memory.

Dirty Victim

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict. The data must therefore be written to memory.

Dual Issue

Two instructions are issued, in parallel, during the same microprocessor cycle. The instructions use different resources and so do not conflict.

Ebox

The Ebox contains the 64-bit integer execution data path.

ECC

Error correction code. Code and algorithms used by logic to facilitate error detection and correction. *See also* ECC error.

ECC error

An error detected by ECC logic, to indicate that data (or the protected ‘‘entity’’ has been corrupted. The error may be correctable (ECC error) or uncorrectable (ECCU error).

Fbox

The unit within the 21064 which performs floating-point calculations.

Firmware

Machine instructions stored in hardware.

Floating-point

A number system in which the position of the radix point is indicated by the exponent part and another part represents the significant digits or fractional part.

Granularity

A characteristic of storage systems that defines the amount of data that can be read and/or written with a single instruction, or read and/or written independently. VAX systems have byte or multibyte granularities, whereas disk systems typically have 512-byte or greater granularities. For a given storage device, a higher granularity generally yields a greater throughput.

Hardware Interrupt Request (HIR)

An interrupt generated by a peripheral device.

High-impedance State

An electrical state of high resistance to current flow, which makes the device appear not physically connected to the circuit.

Hit

See cache hit.

Ibox

A logic unit within the 21064 which fetches, decodes and issues instructions. It also controls the microprocessor pipeline.

Icache

Instruction cache. A cache reserved for storage of instructions.

Internal Processor Register (IPR)

A register internal to the CPU chip.

Latency

The amount of time it takes the system to respond to an event.

Load/Store Architecture

A characteristic of a machine architecture where data items are first loaded into a processor register, operated on, and then stored back to memory. No operations on memory other than load and store are provided by the instruction set.

longword

Four contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 31.

machine check

An operating system action triggered by certain system hardware-detected errors that can be fatal to system operation. Once triggered, machine check handler software analyzes the error.

Masked write

A write cycle that only updates a subset of a nominal data block.

MBO

See must be one.

MBZ

See must be zero.

MIPS

Millions of instructions per second.

Miss

See cache miss.

Module

A board on which logic devices (such as transistors, resistors, and memory chips) are mounted and connected to perform a specific system function.

Multiprocessing

A processing method that replicates the sequential computer and interconnects the collection so that each processor can execute the same or a different program at the same time.

Must Be One (MBO)

A field that must be supplied as one.

Must Be Zero (MBZ)

A field that is reserved and must be supplied as zero. If examined, it must be assumed to be undefined.

Naturally Aligned

See aligned.

Naturally Aligned Data

Data stored in memory such that the address of the data is evenly divisible by the size of the data in bytes. For example, an aligned longword is stored such that the address of the longword is evenly divisible by 4.

Octaword

Sixteen contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 127.

OpenVMS AXP Operating System

Digital's open version of the VMS operating system, which runs on Alpha AXP machines.

Operand

The data or register upon which an operation is performed.

PALcode

Alpha AXP Privileged Architecture Library code, written to support Alpha AXP processors. PALcode implements architecturally defined behavior.

PALmode

A special environment for running PALcode routines.

Parameter

A variable that is given a specific value that is passed to a program before execution.

Parity

A method for checking the accuracy of data by calculating the sum of the number of ones in a piece of binary data. Even parity requires the correct sum to be an even number, odd parity requires the correct sum to be an odd number.

Pipeline

A CPU design technique whereby multiple instructions are simultaneously overlapped in execution.

Primary Cache

The cache that is the fastest and closest to the processor.

The first-level cache, located on the CPU chip, composed of the D-cache and the I-cache.

Program Counter

That portion of the CPU that contains the virtual address of the next instruction to be executed. Most current CPUs implement the program counter (PC) as a register. This register may be visible to the programmer through the instruction set.

Pulldown Resistor

A resistor placed between a signal line and a negative voltage.

Pullup Resistor

A resistor placed between a signal line to a positive voltage.

Quadword

Eight contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 63.

READ_BLOCK

A transaction where the 21064 requests that an external logic unit fetch read data.

Read Data Wrapping

System feature that reduces apparent memory latency by allowing read data cycles to differ the usual low-to-high sequence. Requires cooperation between the 21064 and external hardware.

Read Stream Buffers

Arrangement whereby each memory module independently prefetches DRAM data prior to an actual read request for that data. Reduces average memory latency while improving total memory bandwidth.

Register

A temporary storage or control location in hardware logic.

Reliability

The probability a device or system will not fail to perform its intended functions during a specified time interval when operated under stated conditions.

reset

An action which causes a logic unit to interrupt the task it is performing and go to its' initialized state.

RISC

Reduced instruction set computer. A computer with an instruction set that is reduced in complexity.

ROM

Read-only memory.

SBO

Should be one.

SBZ

Should be zero.

serial ROM

Serial read-only memory.

SROM

See serial ROM.

Stack

An area of memory set aside for temporary data storage or for procedure and interrupt service linkages. A stack uses the last-in/first-out concept. As items are added to (pushed on) the stack, the stack pointer decrements. As items are retrieved from (popped off) the stack, the stack pointer increments.

Static Stage

The 21064 integer pipeline divides instruction processing into four static and three dynamic stages of execution. The 21064 floating point pipeline implements the first four static stages and six dynamic stages of execution. The four static stages consist of:

- Instruction fetch
- Swap
- Decode
- Issue logic

Superpipelined

Describes a pipelined machine that has a larger number of pipe stages and more complex scheduling and control. *See also* pipeline.

Superscalar

Describes a machine that issues multiple independent instructions per clock cycle.

Tristate

Refers to a bused line that has three states: high, low, and high-impedance.

Unaligned

A datum of size 2**N stored at a byte address that is not a multiple of 2**N.

Unconditional Branch Instructions

Instructions that write a return address into a register.

Undefined

An operation that may halt the processor or cause it to lose information. Only privileged software (that is, software running in kernel mode) can trigger an undefined operation.

Unpredictable

Results or occurrences that do not disrupt the basic operation of the processor; the processor continues to execute instructions in its normal manner. Privileged or unprivileged software can trigger unpredictable results or occurrences.

Victim

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict.

Virtual Cache

A cache that is addressed with virtual addresses. The tag of the cache is a virtual address. This process allows direct addressing of the cache without having to go through the translation buffer making cache hit times faster.

Word

Two contiguous bytes (16 bits) starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 15.

Write Back

A cache management technique in which write operation data is written into cache but is not written into main memory in the same operation. This may result in temporary differences between cache data and main memory data. Some logic unit must maintain coherency between cache and main memory.

Write Data Wrapping

System feature that reduces apparent memory latency by allowing write data cycles to differ the usual low-to-high sequence. Requires cooperation between the 21064 and external hardware.

Write Through

A cache management technique in which a write operation to cache also causes the same data to be written in main memory.

WRITE_BLOCK

A transaction where the 21064 requests that an external logic unit process write data.

Index

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

W