Alpha 21164PC (.28µm) Microprocessor

Hardware Reference Manual

Order Number: EC–RADPA–TE

Revision/Update Information: This is a preliminary document.

Preliminary

Compaq Computer Corporation

Contents

Figures

Tables

Preface

This manual provides information about the architecture, internal design, external interface, and specifications of the Alpha 21164PC microprocessor (referred to as the 21164PC) and its associated software.

Audience

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

Manual Organization

This manual includes the following chapters and appendixes, and an index.

- Chapter 1, Introduction, introduces the 21164PC and provides an overview of the Alpha architecture.
- Chapter 2, Internal Architecture, describes the major hardware functions and the internal chip architecture. It describes performance measurement facilities, coding rules, and design examples.
- Chapter 3, Hardware Interface, lists and describes the external hardware interface signals.
- Chapter 4, Clocks, Cache, and External Interface, describes the external bus functions and transactions, lists bus commands, and describes the clock functions.
- Chapter 5, Internal Processor Registers, lists and describes the 21164PC internal processor register set.
- Chapter 6, Privileged Architecture Library Code, describes the privileged architecture library code (PALcode).

- Chapter 7, Initialization and Configuration, describes the initialization and configuration sequence.
- Chapter 8, Error Detection and Error Handling, describes error detection and error handling.
- Chapter 9, Electrical Data, provides electrical data and describes signal integrity issues.
- Chapter 10, Power Management, describes the processor states required for managing system power.
- Chapter 11, Thermal Management, provides information about thermal management.
- Chapter 12, Mechanical Packaging Information, provides mechanical data and packaging information, including signal pin lists.
- Chapter 13, Testability and Diagnostics, describes chip and system testability features.
- Appendix A, Alpha Instruction Set, summarizes the Alpha instruction set.
- Appendix B, 21164PC Microprocessor Specifications, summarizes the 21164PC specifications.
- Appendix C, Serial Icache Load Predecode Values, provides a C code example that calculates the predecode values of a serial Icache load.
- Appendix D, Support, Products, and Documentation, provides the Alpha OEM website URL for support and lists related Alpha and third-party publications with order information.
- The Glossary lists and defines terms associated with the 21164PC.

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

Conventions

This section defines product-specific terminology, abbreviations, and other conventions used throughout this manual.

Abbreviations

- Binary Multiples

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

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

  For example:

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

- Register Access

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

  IGN — Ignore

  Register bits specified as IGN are ignored on writes.

  MBZ — Must Be Zero

  Software must never place a nonzero value in bits and fields specified as MBZ. Reads return unpredictable values. Such fields are reserved for future use.

  RAO — Read As One

  Register bits specified as RAO return a 1 when read.

  RAZ — Read As Zero

  Register bits specified as RAZ return a 0 when read.

RES — Reserved

Bits and fields specified as RES are reserved by COMPAQ and should not be used; however, zeros can be written to reserved fields that cannot be masked.

RO — Read Only

Bits and fields specified as RO can be read and are ignored (not written) on writes.

RW — Read/Write

Bits and fields specified as RW can be read and written.

W0C — Write Zero to Clear

Bits and fields specified as W0C can be read. Writing a zero clears these bits for the duration of the write; writing a one has no effect.

W1C — Write One to Clear

Bits and fields specified as W1C can be read. Writing a one clears these bits for the duration of the write; writing a zero has no effect.

WO — Write Only

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

Addresses

Unless otherwise noted, all addresses and offsets are hexadecimal.

Aligned and Unaligned

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

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

Bit Notation

Multiple-bit fields can include contiguous and noncontiguous bits contained in square brackets ([]). Multiple contiguous bits are indicated by a pair of numbers separated by a colon (:). For example, [9:7,5,2:0] specifies bits 9,8,7,5,2,1, and 0. Similarly, single bits are frequently indicated with square brackets. For example, [27] specifies bit 27. See also Field Notation.

Caution

Cautions indicate potential damage to equipment or loss of data.

Data Units

The following data-unit terminology is used throughout this manual.

Term	Words	Bytes	Bits	Other
Byte	½	1	8	—
Word	1	2	16	—
Dword	2	4	32	Longword
Quadword	4	8	64	2 Dwords

External

Unless otherwise stated, external means not contained in the 21164PC.

Field Notation

The names of single-bit and multiple-bit fields can be used rather than the actual bit numbers (see Bit Notation). When the field name is used, it is contained in square brackets ([]). For example, **RegisterName[LowByte]** specifies **RegisterName[7:0]**.

Numbering

All numbers are decimal or hexadecimal unless otherwise indicated. The prefix 0x indicates a hexadecimal number. For example, 19 is decimal, but 0x19 and 0x19A are hexadecimal (also see Addresses). Otherwise, the base is indicated by a subscript; for example, 100_2 is a binary number.

Ranges and Extents

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

Extents are specified by a pair of numbers in square brackets ([]) separated by a colon (:) and are inclusive. Bit fields are often specified as extents. For example, bits

[7:3] specifies bits 7, 6, 5, 4, and 3.

Security Holes

Security holes exist when unprivileged software (that is, software that is 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

Signal Names

Signal names are printed in lowercase, boldface type. Low-asserted signals are indicated by the **_l** suffix, while high-asserted signals have the **_h** suffix. For example, **clk_in_h** is a high-asserted signal, and **clk_in_l** is a low-asserted signal.

Unpredictable and Undefined

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

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

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

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

Unpredictable

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

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

  Operations that produce UNPREDICTABLE results may also produce exceptions.

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

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

  Also, operations that may produce UNPREDICTABLE results must not:

  – Write or modify the contents of memory locations or registers to which the current process in the current access mode does not have access

  – Halt or hang the system or any of its components

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

Undefined

- Operations specified as UNDEFINED may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. The operation may vary in effect from nothing, to stopping system operation.

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

Revision History

Date	Revision	Description
December 9, 1998	EC–RADPA–TE	Preliminary version

1 Introduction

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

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

1.1 The Architecture

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

All registers are 64 bits long and all operations are performed between 64-bit registers. All instructions are 32 bits long. Memory operations are either load or store operations. All data manipulation is done between registers.

The Alpha architecture supports the following data types:

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

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

The Architecture

The 21164PC uses a set of subroutines, called privileged architecture library code (PALcode), that is specific to a particular Alpha operating system implementation and hardware platform. These subroutines provide operating system primitives for context switching, interrupts, exceptions, and memory management. These subroutines can be invoked by hardware or CALL_PAL instructions. CALL_PAL instructions use the function field of the instruction to vector to a specified subroutine. PALcode is written in standard machine code with some implementation-specific extensions to provide direct access to low-level hardware functions. PALcode supports optimizations for multiple operating systems, flexible memory-management implementations, and multi-instruction atomic sequences.

The Alpha architecture performs byte shifting and masking with normal 64-bit, register-to-register instructions, and also includes single-byte load and store instructions.

1.1.1 Addressing

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

Virtual addresses as seen by the program are translated into physical memory addresses by the memory-management mechanism. The 21164PC supports a 40-bit uncached and a 33-bit cached physical address space.

1.1.2 Integer Data Types

Alpha architecture supports four integer data types.

Data Type	Description
Byte	A byte is eight contiguous bits that start at an addressable byte boundary. A byte is an 8-bit value. A byte is supported in Alpha architecture by the EXTRACT, INSERT, LDBU, MASK, SEXTB, STB, ZAP, PACK, UNPACK, MIN, MAX, and PERR instructions.
Word	A word is two contiguous bytes that start at an arbitrary byte boundary. A word is a 16-bit value. A word is supported in Alpha architecture by the EXTRACT, INSERT, LDWU, MASK, SEXTW, STW, PACK, UNPACK, MIN, and MAX instructions.
Longword	A longword is four contiguous bytes that start at an arbitrary byte boundary. A longword is a 32-bit value. A longword is supported in Alpha architecture by sign-extended load and store instructions and by longword arithmetic instructions.
Quadword	A quadword is eight contiguous bytes that start at an arbitrary byte boundary. A quadword is supported in Alpha architecture by load and store instructions and quadword integer operate instructions.

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

1.1.3 Floating-Point Data Types

The 21164PC supports the following floating-point data types:

- Longword integer format in floating-point unit
- Quadword integer format in floating-point unit
- IEEE floating-point formats
  - S_floating
  - T_floating

- VAX floating-point formats
  - F_floating
  - G_floating
  - D_floating (limited support)

1.2 21164PC Microprocessor Features

The 21164PC is a superscalar pipelined processor manufactured using 0.28-µm CMOS technology. It is packaged in a 413-pin interstitial pin grid array (IPGA) carrier and has removable application-specific heat sinks. The 21164PC has been optimized for uniprocessor systems with very high cache and memory bandwidth. The 21164PC supports the new motion video instructions (MVI) added to the Alpha instruction set.

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

The 21164PC and associated PALcode implements IEEE single-precision and double-precision, VAX F_floating and G_floating data types, and supports longword (32-bit) and quadword (64-bit) integers. Byte (8-bit) and word (16-bit) support is provided by byte-manipulation instructions. Limited hardware support is provided for the VAX D_floating data type.

Other 21164PC features include:

- A peak instruction execution rate of four times the CPU clock frequency.
- The ability to issue up to four instructions during each clock cycle.
- An onchip, demand-paged memory-management unit with translation buffer, which, when used with PALcode, can implement a variety of page table structures and translation algorithms. The unit consists of a 64-entry data translation buffer (DTB) and a 48-entry instruction translation buffer (ITB), with each entry able to map a single 8KB page or a group of 8, 64, or 512 8KB pages. The size of each translation buffer entry's group is specified by hint bits stored in the entry. The DTB and ITB implement 7-bit address space numbers (ASN), (MAX_ASN=127).
- Two onchip, high-throughput pipelined floating-point units, capable of executing both COMPAQ and IEEE floating-point data types.

21164PC Microprocessor Features

- An onchip, 32KB, virtual, 2-way set-associative instruction cache with 7-bit ASNs (MAX_ASN=127).
- An onchip, dual-read-ported, 16KB direct-mapped, virtually-indexed, physically-tagged data cache.
- An onchip write buffer with eight 32-byte entries.
- A 128-bit data bus with onchip parity and offchip longword parity.
- Support for an external second-level cache. The size and access time of the external second-level cache is programmable.
- A 1× clocking mode, plus an onchip PLL to provide a high-speed CPU clock.
- Onchip performance counters to measure and analyze CPU and system performance.
- Chip and module level test support, including an instruction cache test interface to support chip and module level testing.
- A 2.5-V external interface and 2.0-V internal interface.

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

2 Internal Architecture

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

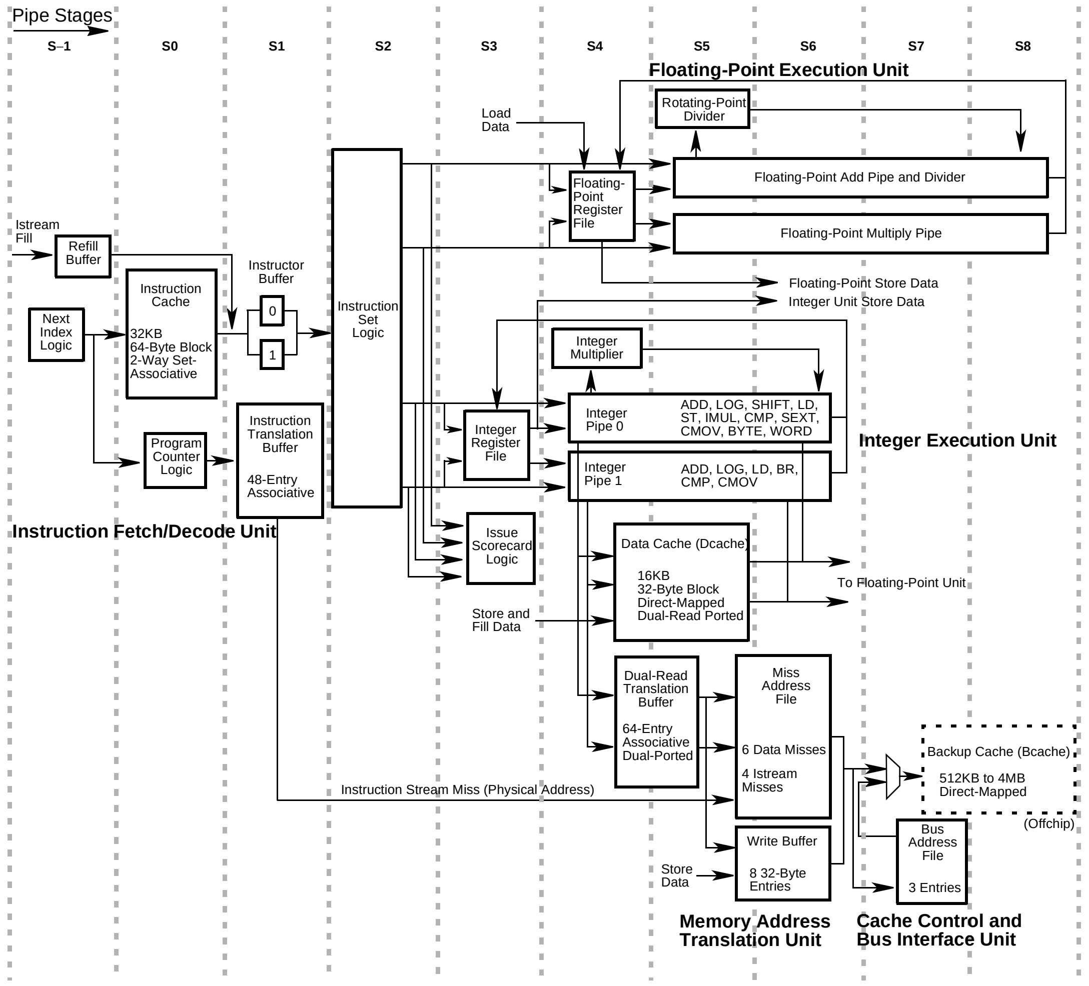
This chapter describes the major functional hardware units and is not intended to be a detailed hardware description of the chip. It is organized as follows:

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

2.1 21164PC Microarchitecture

The 21164PC microprocessor is a high-performance implementation of COMPAQ's Alpha architecture. Figure 2–1 is a block diagram of the 21164PC that shows the major functional blocks relative to pipeline stage flow. The following paragraphs provide an overview of the chip's architecture and major functional units.

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



The 21164PC microprocessor consists of the following internal sections:

- Clock generation logic (Section 4.2)
- Instruction fetch/decode unit and branch unit (IDU) (Section 2.1.1), which includes:
  - Instruction prefetcher and instruction decoder
  - Instruction translation buffer
  - Branch prediction
  - Instruction slotting/issue
  - Interrupt support
- Integer execution unit (IEU) (Section 2.1.2)
- Floating-point execution unit (FPU) (Section 2.1.3)
- Memory address translation unit (MTU) (Section 2.1.4), which includes:
  - Data translation buffer (DTB)
  - Miss address file (MAF)
  - Write buffer
  - Dcache control
- Cache control and bus interface unit (CBU) with interface to external cache (Section 2.1.5)
- Data cache (Dcache) (Section 2.1.6.1)
- Instruction cache (Icache) (Section 2.1.6.2)
- Serial read-only memory (SROM) interface (Section 2.1.7)

2.1.1 Instruction Fetch/Decode Unit and Branch Unit

The primary function of the instruction fetch/decode unit and branch unit (IDU) is to manage and issue instructions to the IEU, MTU, and FPU. It also manages the instruction cache. The IDU contains:

- Prefetcher and instruction buffer
- Instruction slot and issue logic
- Program counter (PC) and branch prediction logic

- 48-entry instruction translation buffers (ITBs)
- Abort logic
- Register conflict logic
- Interrupt and exception logic

2.1.1.1 Instruction Decode and Issue

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

In other words:

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

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

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

2.1.1.2 Instruction Prefetch

The IDU contains an instruction prefetcher and a four-entry, 32-byte-per-entry, prefetch buffer called the refill buffer. Each instruction fetch is looked up in the tags for both ways. One way is designated as primary, the other as secondary. Tag hit is calculated on both ways simultaneously, instruction data for the primary way is returned to the pipeline, and instruction data for the secondary way is siloed in the

Icache. If there is a hit in the primary way, fetching continues uninterrupted. However, if the primary way misses, and the secondary way hits, then the siloed data is sent to the pipeline the next cycle and a one cycle way-mispredict is taken.

A miss in both the primary and secondary ways is checked in the refill buffer. If the refill buffer contains the instruction data, it fills the Icache and instruction buffer simultaneously. If the refill buffer does not contain the necessary data, a fetch and a number of prefetches are sent to the MTU. One prefetch is sent per cycle until each of the four entries in the refill buffer is filled or has a pending fill. The refill buffer holds all returned fill data until the data is required by the IDU pipeline or until it is overwritten by a subsequent fetch/prefetch sequence caused by a future Icache miss.

Prefetching does not begin until there is a “true” miss. A true miss is a reference that misses in both Icache ways and also misses in the refill buffer. If an Icache miss results in a refill buffer hit, prefetching is not started until all the data has been moved from the refill buffer entry into the pipeline. Adjacent Icache fills are filled to the same way, in order to reduce the number of way-mispredicts.

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

Once an Icache miss occurs, the Icache enters fill mode. When the Icache is in fill mode, the refill buffer is checked for a hit while it awaits the arrival of the data from the Bcache or main memory. The IDU sends a read request to the CBU by means of the MTU. The CBU checks the Bcache, and if the request misses, the CBU drives a main memory request.

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

The Icache has a 64-byte block size, whereas the refill buffer is able to load the Icache with only one INT16 (16 bytes) per cycle. Therefore, each Icache block has four valid bits, one for each 16-byte subblock.

2.1.1.3 Branch Execution

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

The Icache records the outcome of branch instructions in a 4096-entry, 2-bit per entry branch history table. The table is indexed by the instruction's virtual address bits [13:03] and a 5-bit branch history vector. The output of the branch history table is used as the prediction for the next execution of the branch instruction. The 2-bit history state is a saturating counter that increments on taken branches and decrements on not-taken branches. The branch is predicted taken on the top two count values and is predicted not-taken on the bottom two count values. The branch history table is also overloaded for use as a way predictor for jump instructions as described below. The history status is not initialized on Icache fill, therefore it may "remember" a branch that was evicted from the Icache and subsequently reloaded.

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

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

The 21164PC provides a 12-entry subroutine return stack that is controlled by decoding the opcode (BSR, HW_REI, JMP/JSR/RET/JSR_COROUTINE), and DISP[15:0] in JMP/JSR/RET/JSR_COROUTINE. The stack stores an Icache index in each entry. The stack is implemented as a circular queue that wraps around in overflow and underflow cases.

Table 2–1 lists the effect each of these instructions has on the state of the branch-prediction stack.

Table 2–1 Effect of Branching Instructions on the Subroutine Return Stack

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

The 21164PC uses the Icache index hint in the JMP and JSR instructions to predict the target PC. The Icache index hint in the instruction's displacement field is used to access the Icache.

The RET, JSR_COROUTINE, and HW_REI instructions predict the next PC by using the index from the subroutine return stack. For these instructions, the Icache way is predicted using the upper bit of the branch history table: 1 means predict the target is in way 1 and 0 means predict way 0. The upper bits of the PC are formed from the data in the predicted way's Icache tag store at that index. Later in the pipeline, the PC prediction is checked against the actual PC generated by the IEU. A mismatch causes a PC mispredict trap, restart from the correct PC, and an update to the branch history table. This is similar to branch prediction.

The branch prediction stack never predicts a target address in PALmode. This prevents the possibility of nonprivileged code accessing privileged modes through incorrect stack predictions (for example, by underflow/overflow of the stack). This implies that PALcode libraries should avoid using instructions such as RET and JSR_COROUTINE for internal jumps with PALmode targets, as the 21164PC will always mispredict the target address.

2.1.1.4 Instruction Translation Buffer

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

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

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

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

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

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

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

2.1.1.5 Interrupts

The IDU exception logic supports three sources of interrupts:

- Hardware interrupts

There are 7 level-sensitive hardware interrupt sources supplied by the following signals:

irq_h[3:0]
mch_hlt_irq_h
pwr_fail_irq_h
sys_mch_chk_irq_h

- Software interrupts

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

- Asynchronous system traps (ASTs)

  There are 4 ASTs sourced by the asynchronous system trap request (ASTRR) register.

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

Each interrupt source, or group of sources, is assigned an interrupt priority level (IPL), as shown in Table 4–11. The current IPL is set using the IPLR register (see Section 5.1.18). Any interrupts that have an equal or lower IPL are masked. When an interrupt occurs that has an IPL greater than the value in the IPLR register, program control passes to the INTERRUPT PALcode entry point. PALcode processes the interrupt by reading the ISR (see Section 5.1.24) and the INTID register (see Section 5.1.19).

2.1.2 Integer Execution Unit

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

- Two adders
- Two logic boxes
- A barrel shifter
- Byte-manipulation logic
- An integer multiplier
- A motion video instruction unit

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

2.1.3 Floating-Point Execution Unit

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

- A 32-entry, 64-bit floating-point register file
- A user-accessible control register
- A floating-point multiply pipeline
- A floating-point add pipeline

  The floating-point divide unit is associated with the floating-point add pipeline but is not pipelined.

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

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

2.1.4 Memory Address Translation Unit

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

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

The MTU receives up to two virtual addresses every cycle from the IEU. The translation buffer generates the corresponding physical addresses and access control information for each virtual address. The 21164PC implements a 43-bit virtual address, a 40-bit noncacheable physical address, and a 33-bit cacheable physical address. Cacheable addresses consist of bits [32:0] when bit [39] = 0. Physical addresses that set bits [38:33] are not supported by the 21164PC. These addresses are not checked by the 21164PC and could result in erroneous data.

2.1.4.1 Data Translation Buffer

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

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

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

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

2.1.4.2 Load Instruction and the Miss Address File

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

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

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

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

2.1.4.3 Dcache Control and Store Instructions

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

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

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

2.1.4.4 Write Buffer

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

2.1.5 Cache Control and Bus Interface Unit

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

2.1.6 Cache Organization

The 21164PC has two onchip caches—a primary data cache (Dcache) and a primary instruction cache (Icache). All memory cells in the onchip caches are fully static, six-transistor, CMOS structures.

The 21164PC also provides control for the external cache (Bcache).

2.1.6.1 Data Cache

The data cache (Dcache) is a dual-read-ported, single-write-ported, 16KB cache. It is a write-through, read-allocate, direct-mapped, byte-accessible, virtually-indexed and physically-tagged cache with 32-byte blocks and data parity at the byte level.

2.1.6.2 Instruction Cache

The instruction cache (Icache) is a 32KB, virtual, 2-way set-associative cache with 64-byte blocks and 32-byte fills. Each block tag contains:

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

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

2.1.6.3 External Cache

The CBU implements control for an external, direct-mapped, physical, write-back, write-allocate cache with 64-byte blocks. The 21164PC supports board-level cache sizes of 512KB, 1MB, 2MB, and 4MB.

2.1.7 Serial Read-Only Memory Interface

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

2.2 Pipeline Organization

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

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

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

Figure 2–2 Instruction Pipeline Stages

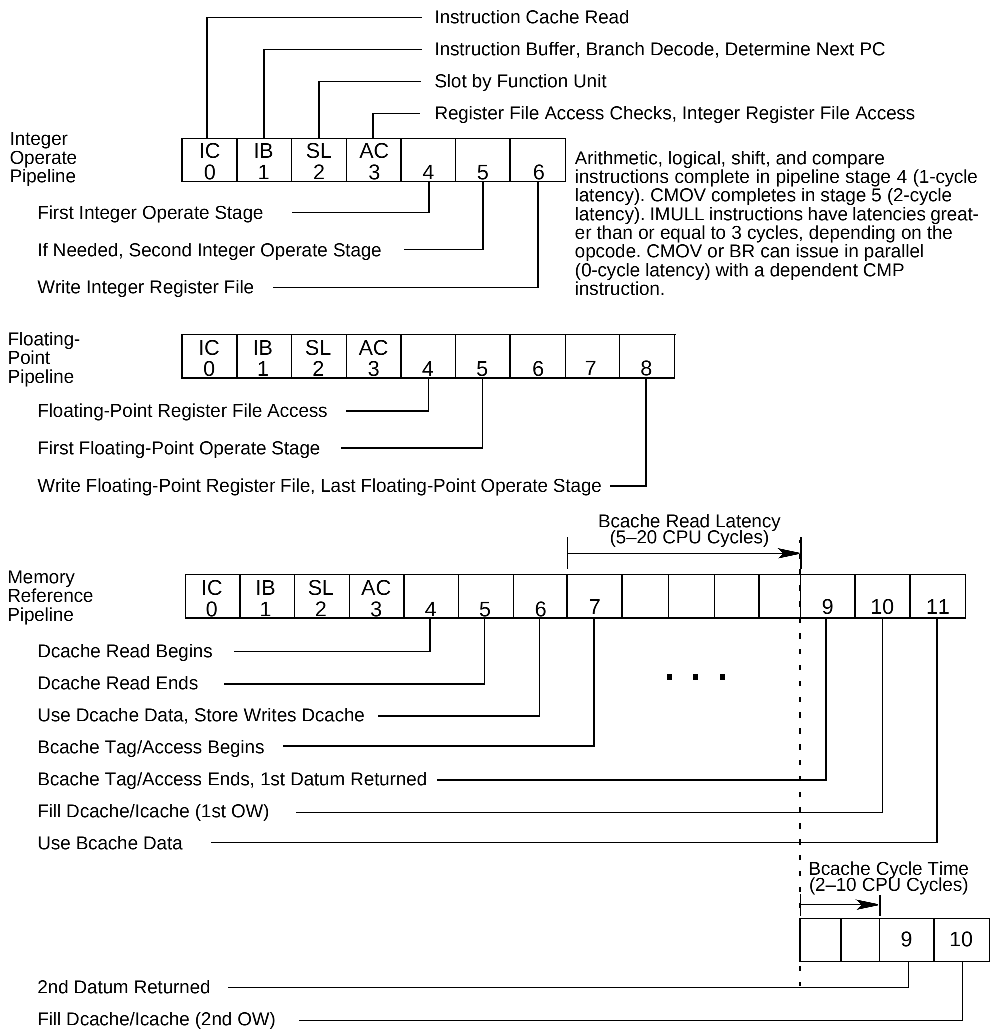

Instruction Cache Read
Instruction Buffer, Branch Decode, Determine Next PC
Slot by Function Unit
Register File Access Checks, Integer Register File Access
Integer Operate Pipeline
IC 0
IB 1
SL 2
AC 3
4
5
6
Arithmetic, logical, shift, and compare instructions complete in pipeline stage 4 (1-cycle latency). CMOV completes in stage 5 (2-cycle latency). IMULL instructions have latencies greater than or equal to 3 cycles, depending on the opcode. CMOV or BR can issue in parallel (0-cycle latency) with a dependent CMP instruction.
First Integer Operate Stage
If Needed, Second Integer Operate Stage
Write Integer Register File
Floating-Point Pipeline
IC 0
IB 1
SL 2
AC 3
4
5
6
7
8
Floating-Point Register File Access
First Floating-Point Operate Stage
Write Floating-Point Register File, Last Floating-Point Operate Stage
Bcache Read Latency (5–20 CPU Cycles)
Memory Reference Pipeline
IC 0
IB 1
SL 2
AC 3
4
5
6
7
9
10
11
. . .
Dcache Read Begins
Dcache Read Ends
Use Dcache Data, Store Writes Dcache
Bcache Tag/Access Begins
Bcache Tag/Access Ends, 1st Datum Returned
Fill Dcache/Icache (1st OW)
Use Bcache Data
Bcache Cycle Time (2–10 CPU Cycles)
9
10
2nd Datum Returned
Fill Dcache/Icache (2nd OW)

Table 2–2 Pipeline Examples—All Cases

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

Table 2–3 Pipeline Examples—Integer Add

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

Table 2–4 Pipeline Examples—Floating Add

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

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

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

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

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

Pipeline Stage[1]	Events
4	Calculate the effective address. Begin the Dcache data and tag store access.
5	Finish the Dcache data and tag store access. Detect Dcache miss. Bcache arbitration defaults to pipe E0 in anticipation of a possible miss. If there are load instructions in both E0 and E1, the load instruction in E1 would be delayed at least one more cycle because default arbitration speculatively assumes the load in E0 will miss.
6	Forward physical address to pins.
7	Begin Bcache access, cycle 1.
8	*N* more CPU cycles waiting for Bcache data.
9	Receive Bcache data at the pins, send data to the Dcache.
10	Begin Dcache fill. Format the data as required.
11	Finish the Dcache fill. Write the integer or floating-point register file. Data is available for use by an operate function in this cycle.

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

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

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

2.2.1 Pipeline Stages and Instruction Issue

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

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

2.2.2 Aborts and Exceptions

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

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

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

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

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

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

2.2.3 Nonissue Conditions

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

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

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

2.3 Scheduling and Issuing Rules

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

2.3.1 Instruction Class Definition and Instruction Slotting

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

Table 2–8 Instruction Classes and Slotting *(Sheet 1 of 3)*

Class Name	Pipeline	Instruction List
LD	E0[1] or E1[2]	All loads except LD*x*_L
ST	E0	All stores except ST*x*_C
MBX	E0	LD*x*_L, MB, WMB, ST*x*_C, HW_LD-lock, HW_ST-cond, FETCH
RX	E0	RS, RC

Table 2–8 Instruction Classes and Slotting *(Sheet 2 of 3)*

Class Name	Pipeline	Instruction List
MXPR	E0 or E1 (depends on the IPR)	HW_MFPR, HW_MTPR
IBR	E1	Integer conditional branches
FBR	FA[3]	Floating-point conditional branches
JSR	E1	Jump-to-subroutine instructions: JMP, JSR, RET, or JSR_COROUTINE, BSR, BR, HW_REI, CALLPAL
IADD	E0 or E1	ADDL, ADDL/V, ADDQ, ADDQ/V, SUBL, SUBL/V, SUBQ, SUBQ/V, S4ADDL, S4ADDQ, S8ADDL, S8ADDQ, S4SUBL, S4SUBQ, S8SUBL, S8SUBQ, LDA, LDAH
ILOG	E0 or E1	AND, BIS, XOR, BIC, ORNOT, EQV
SEXT	E0	SEXTB, SEXTW
SHIFT	E0	SLL, SRL, SRA, EXTQL, EXTLL, EXTWL, EXTBL, EXTQH, EXTLH, EXTWH, MSKQL, MSKLL, MSKWL, MSKBL, MSKQH, MSKLH, MSKWH, INSQL, INSLL, INSWL, INSBL, INSQH, INSLH, INSWH, ZAP, ZAPNOT
CMOV	E0 or E1	CMOVEQ, CMOVNE, CMOVLT, CMOVLE, CMOVGT, CMOVGE, CMOVLBS, CMOVLBC
ICMP	E0 or E1	CMPEQ, CMPLT, CMPLE, CMPULT, CMPULE, CMPBGE
IMULL	E0	MULL, MULL/V
IMULQ	E0	MULQ, MULQ/V
IMULH	E0	UMULH
MVI	E0	PERR, UNPKBW, UNPKBL, PKWB, PKLB, MINSB8, MINSB4, MINUB8, MINUW4, MAXUB8, MAXUW4, MAXSB8, MAXSW4
FADD	FA	Floating-point operates, including CPYSN and CPYSE, except multiply, divide, and CPYS
FDIV	FA	Floating-point divide
FMUL	FM[4]	Floating-point multiply

Table 2–8 Instruction Classes and Slotting *(Sheet 3 of 3)*

Class Name	Pipeline	Instruction List
FCPYS	FM or FA	CPYS, not including CPYSN or CPYSE
MISC	E0	RPCC, TRAPB
UNOP	None	UNOP[5]

[1] IEU pipeline 0.
[2] IEU pipeline 1.
[3] FPU add pipeline.
[4] FPU multiply pipeline.
[5] UNOP is LDQ_U R31,0(Rx).

Slotting

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

Starting from the first (lowest addressed) valid instruction in the INT16 in stage 2 of the 21164PC IDU pipeline, attempt to assign that instruction to one of the four pipelines (E0, E1, FA, FM). If it is an instruction that can issue in either E0 or E1, assign it to E0. However, if one of the following is true, assign it to E1:

- E0 is not free and E1 is free.
- The next integer instruction[1] in this INT16 can issue only in E0.

If the current instruction is one that can issue in either FA or FM, assign it to FA unless FA is not free. If it is an FA-only instruction, it must be assigned to FA. If it is an FM-only instruction, it must be assigned to FM. Mark the pipeline selected by this process as taken and resume with the next sequential instruction. Stop when an instruction cannot be allocated in an execution pipeline because any pipeline it can use is already taken.

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

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

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

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

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

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

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

2.3.2 Coding Guidelines

Code should be scheduled according to latency and function unit availability. This is good practice in most RISC architectures. Code alignment and the effects of split-issue[2] should be considered.

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

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

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

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

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

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

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

2.3.3 Instruction Latencies

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

The latency to produce a valid result for most instructions is fixed. The exceptions are loads that miss and integer multiplies. Table 2–9 gives the latencies for each instruction class. A latency of 1 means that the result may be used by an instruction issued one cycle after the producing instruction. There are no variations in latency due to which a particular unit produces a given result relative to the particular unit that consumes it.

Table 2–9 Instruction Latencies *(Sheet 1 of 2)*

Class	Latency
LD	Dcache hits, latency=2. Dcache miss/Bcache hit, latency=10 or longer.[1]
ST	Store operations produce no result.
MBX	LD*x*_L Dcache hits, latency=2. LD*x*_L Dcache miss/Bcache hit, latency=10 or longer.[1] LD*x*_L Dcache miss/Bcache miss, latency depends on memory subsystem state. ST*x*_C, latency depends on memory subsystem state. MB, WMB, and FETCH produce no result.
RX	RS, RC, latency=1.
MXPR	HW_MFPR, latency=1, 2, or longer, depending on the IPR. HW_MTPR, produces no result.
IBR	Produces no result. (Taken branch issue latency minimum=1 cycle, branch mispredict penalty=5 cycles.)
FBR	Produces no result. (Taken branch issue latency minimum=1 cycle, branch mispredict penalty=5 cycles.)
JSR	All but HW_REI, latency=1. HW_REI produces no result. (Issue latency—minimum 1 cycle.)
SEXT	Latency=1.
IADD	Latency=1.
ILOG	Latency=1.[2]
SHIFT	Latency=1.
CMOV	Latency=2.
ICMP	Latency=1.[2]
IMULL	Latency=3.
IMULL/V	Latency=4. Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 2 cycles.
IMULQ	Latency=4. Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 2 cycles.
IMULQ/V	Latency=7. Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 5 cycles.

Table 2–9 Instruction Latencies *(Sheet 2 of 2)*

Class	Latency
IMULH	Latency=7. Latency until next IMULL, IMULQ, or IMULH instruction can issue (if there are no data dependencies) is 5 cycles.
MVI	Latency=2.
FADD	Latency=4.
FDIV	Double-precision latency=13. Latency until next FDIV instruction can issue is 10 cycles. Single-precision latency=8. Latency until next FDIV instruction can issue is 5 cycles.
FMUL	Latency=4.
FCYPS	Latency=4.
MISC	RPCC, latency=2. TRAPB produces no result.
UNOP	UNOP produces no result.

[1] When idle, Bcache arbitration predicts a load miss in E0. If a load actually does miss in E0, it is sent to the Bcache immediately. If it hits in the Bcache, and no other event in the CBU affects the operation, the requested data is available for use in 10 or more cycles. Otherwise, the request takes longer (possibly much longer, depending on the state of the CBU and memory). It should be possible to schedule some unrolled code loops for Bcache by prefetching data into the Dcache using LDQ R31, x(Rx).

[2] A special bypass provides an effective latency of 0 (zero) cycles for an ICMP or ILOG instruction producing the test operand of an IBR or CMOV instruction. This is true only when the IBR or CMOV instruction issues in the same cycle as the ICMP or ILOG instruction that produced the test operand of the IBR or CMOV instruction. In all other cases, the effective latency of ICMP and ILOG instructions is 1 cycle.

2.3.3.1 Producer–Producer Latency

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

Conditions that involve an intervening producer–consumer conflict can occur commonly in a multiple-issue situation when a register is reused. In these cases, producer–consumer latencies are equal to or greater than the required producer–producer latency as determined by write ordering and therefore dictate the overall latency.

An example of this case is shown in the following code:

```
LDQ   R2,0(R0)   ;R2 destination
ADDQ  R2,R3,R4   ;wr-rd conflict stalls execution waiting for R2
LDQ   R2,D(R1)   ;wr-wr conflict may dual issue when ADDQ issues
```

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

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

2.3.4 Issue Rules

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

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

  Technically, load-miss-and-use replay traps are an exception to this rule. The consumer of the load’s result issues, and is aborted, because a load was predicted to hit and was discovered to miss just as the consumer instruction issued. In practice, the only difference is that the latency of the consumer may be longer than it would have been had the issue logic “known” the load would miss in time to prevent issue.

- An instruction of class LD cannot be issued in the second cycle after an instruction of class ST is issued.
- No LD, ST, MXPR (to an MTU register), or MBX class instructions can be issued after an MB instruction has been issued until the MB instruction has been acknowledged by the CBU.

Scheduling and Issuing Rules

- No LD, ST, MXPR (to an MTU register), or MBX class instructions can be issued after a ST*x*_C (or HW_ST-cond) instruction has been issued until the MTU writes the success/failure result of the ST*x*_C (HW_ST-cond) in its destination register.
- No floating-point divide instructions can be issued if the floating-point divider is busy.
- No instruction can be issued to pipe E0 exactly two cycles before an integer multiplication completes.
- No instruction can be issued to pipe FA exactly five cycles before a floating-point divide completes.
- No Store instruction can be issued exactly three cycles before a fill. The data store write operation, if the store hits, will conflict with the fill operation.
- No instruction can be issued to pipe E0 or E1 exactly two cycles before an integer register fill is requested (speculatively) by the CBU, except IMULL/V, IMULQ, IMULQ/V, and IMULH instructions and instructions that do not produce any result.
- No LD, ST, or MBX class instructions can be issued to pipe E0 or E1 exactly one cycle before an integer register fill is requested (speculatively) by the CBU.
- No instruction issues after a TRAPB instruction until all previously issued instructions are guaranteed to finish without generating a trap other than a machine check.

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

2.4 Replay Traps

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

- The write buffer is full when a store instruction is executed and there are already eight write buffer entries allocated. The trap occurs even if the entry would have merged in the write buffer.
- A load instruction is issued in pipe E0 when all six MAF entries are valid (not available), or a load instruction issued in pipe E1 when five of the six MAF entries are valid. The trap occurs even if the load instruction would have hit in the Dcache or merged with an MAF entry.
- Alpha shared memory model order trap (Litmus test 1 trap): If a load instruction issues that address matches with any miss in the MAF (down to the quadword boundary), the load instruction is aborted through a replay trap regardless of whether the newly issued load instruction hits or misses in the Dcache. This ensures that the two loads execute in issue order.
- Load-after-store trap: A replay trap occurs if a load instruction is issued in the cycle immediately following a store instruction that hits in the Dcache, and both access the same location. The address match is exact for address bits [12:2] (longword granularity), but ignores address bits [42:13].
- When a load instruction is followed, within one cycle, by any instruction that uses the result of that load, and the load misses in the Dcache, the consumer instruction traps and is restarted from the beginning of the pipeline. This occurs because the consumer instruction is issued speculatively while the Dcache hit is being evaluated. If the load misses in the Dcache, the speculative issue of the consumer instruction was incorrect. The replay trap generally brings the consumer instruction to the issue point before or simultaneously with the availability of fill data.

2.5 Miss Address File and Load-Merging Rules

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

2.5.1 Merging Rules

When a load miss occurs, each MAF entry is checked to see if it contains a load miss that addresses the same 32-byte Dcache block. If it does, and certain merging rules[3] are satisfied, then the new load miss is merged with an existing MAF entry. This allows the MTU to service two or more load misses with one data fill from the CBU. The merging rules for an individual MAF entry are different for cacheable and non-cacheable space.

2.5.1.1 Cacheable Space Load-Merge Rules

The merging rules for cacheable space loads (physical address bit [39]=0) are as follows:

- Merging only occurs if the new load miss addresses a different INT8 from all loads previously entered or merged to that MAF entry. If it addresses the same INT8, the machine traps and replays the instruction. This continues until the MAF entry is retired, at which time the trapping load hits in the Dcache.
- Bytes, words, longwords, and quadwords *can* merge with each other, provided that they are not in the same INT8.
- Merging is prevented for the MAF entry after the first data fill (to that MAF entry) from the Bcache, regardless of whether the Bcache access hits or not.
- Load misses that match any MAF address down to the INT32 boundary, but could not merge (for any reason), are replay trapped. Once the Dcache is filled, this load instruction executes and hits in the Dcache.

All DREAD load-merging is prevented when MAF_MODE[00]=1 (see Section 5.2.16).

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

2.5.1.2 Noncacheable Space Load-Merge Rules

The merging rules for noncacheable space loads (physical address bit [39]=1) are as follows:

- Merging only occurs if the new load miss addresses a different INT8 from all loads previously entered or merged to that MAF entry. If it addresses the same INT8, the machine traps and replays the instruction. This continues until the MAF entry is retired, at which time the trapping load hits in the Dcache.
- Only quadwords can merge with other quadwords, provided they are not in the same INT8. Bytes, words, and longwords *cannot* merge.
- Merging stops for a load instruction to noncacheable space as soon as the CBU accepts the reference. This permits the system environment to access only those INT8s that are actually requested by load instructions.
- All accesses that could not merge (except those to the same INT8) are allocated new MAF entries.

Noncacheable space load-merging is prevented when MAF_MODE[03]=1. All DREAD load-merging is prevented when MAF_MODE[00]=1 (see Section 5.2.16).

At the external interface, noncacheable read instructions indicate to the system environment which INT32 is addressed and which of the INT8s within the INT32 are actually accessed. Each load for longword, word, or byte data results in a separate request to the CBU.

2.5.2 Read Requests to the CBU

Merging is done for two load instructions that issued simultaneously, and both miss; in effect, as if they were issued sequentially with the load from IEU pipe E0 first. The MTU sends a read request to the CBU for each MAF entry allocated.

A bypass is provided so that if the load instruction issues in IEU pipe E0, and no MAF requests are pending, the load instruction’s read request is sent to the CBU immediately, provided the CBU is ready for such an access. Similarly, if a load instruction from IEU pipe E1 misses, and there was no load instruction in pipe E0 to begin with, the E1 load miss is sent to the CBU immediately. In either case, the bypassed read request is aborted if the load hits in the Dcache, merges in the MAF, or is replay trapped by the MTU.

2.5.3 MAF Entries and MAF Full Conditions

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

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

2.5.4 Fill Operation

Eventually, the CBU provides the data requested for a given MAF entry (a fill). The CBU requests that the IDU allocate up to three consecutive “bubble” cycles in the IEU pipelines. The first bubble prevents any store instruction from issuing. The second bubble prevents any instructions from issuing. The third bubble prevents only MTU instructions (particularly load and store instructions) from issuing. The first bubble prevents store data from colliding with the fill in the data cache. The fill uses the second bubble cycle as it progresses down the IEU/MTU pipelines to format the data and load the register file. It uses the third bubble cycle to fill the Dcache.

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

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

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

LDL_L and LDQ_L instructions always allocate a new MAF entry if they miss the Dcache. LDL_L and LDQ_L instructions that hit in the Dcache are retired by the MTU immediately. No load instructions that follow an LDL_L or LDQ_L instruction are allowed to merge with it. After an LDL_L or LDQ_L instruction is issued (and misses in the Dcache), the IDU does not issue any more MTU instructions until the MTU has successfully sent the LDL_L or LDQ_L instruction to the CBU. This guarantees correct ordering between an LDL_L or LDQ_L instruction and a subsequent STL_C or STQ_C instruction even if they access different addresses.

2.6 MTU Store Instruction Execution

Store instructions execute in the MTU by:

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

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

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

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

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

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

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

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

2.7 Write Buffer and the WMB Instruction

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

2.7.1 Write Buffer

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

In addition to HW_ST and other store instructions, the STQ_C and STL_C instructions are also written into the write buffer and sent to the CBU. However, unlike store instructions, these write buffer-directed instructions are never merged into a write buffer entry with other instructions.

2.7.2 Write Memory Barrier (WMB) Instruction

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

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

2.7.3 Entry-Pointer Queues

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

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

2.7.4 Write Buffer Entry Processing

When the number of entries in the pending-request queue reaches the number programmed in MAF_MODE[WB_SET_LO_THRESH][4], the MTU begins arbitration with the other MTU queue requests. Once the request is granted, the MTU sends the entry at the head of the pending-request queue to the CBU. The MTU then removes the entry from the pending-request queue without placing it in the free-entry queue. When the CBU has completely processed the write buffer entry, it notifies the MTU, and the now invalid write buffer entry is placed in the free-entry queue. The MTU may request that up to five additional write buffer entries be processed while waiting for the CBU to finish the first. The write buffer entries are invalidated and placed in the free-entry queue in the order that the requests complete. This order may be different from the order in which the requests were made.

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

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

- One buffer contains an STQ_C or STL_C instruction.
- One buffer is marked by a WMB flag.
- An MB instruction is being executed by the MTU.
- The number of entries in the write buffer exceeds the number programmed in MAF_MODE[WB_CLR_LO_THRESH].

This ensures that these instructions complete as quickly as possible.

The MTU requests that a write buffer entry be processed every 256 cycles (provided there is a valid entry in the write buffer), even if the write buffer is not arbitrating. This ensures that write instructions do not wait forever to be written to memory. (This is triggered by a free-running timer that is reset each time a write operation is completed.)

[4] The following actions can also cause the WB to begin arbitration: (1) an MB or WMB instruction is issued, or (2) 264 cycles have elapsed without completing a write operation while there were pending write operations in the WB (triggered by the WB write counter).

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

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

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

2.7.5 Ordering of Noncacheable Space Write Instructions

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

2.8 Performance Measurement Support–Performance Counters

The 21164PC contains a performance-recording feature. The implementation of this feature provides a mechanism to count various hardware events and causes an interrupt upon counter overflow. Interrupts are triggered six cycles after the event and, therefore, the exception PC might not reflect the exact instruction causing counter overflow. Three counters are provided to allow accurate comparison of two variables under a potentially nonrepeatable experimental condition. The three counters are designated counter 0 (16 bits), counter 1 (16 bits), and counter 2 (14 bits).

Counter inputs include:

- Issues
- Nonissues

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

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

- Hardware interrupt clear (HWINT_CLR) register (see Section 5.1.23)
- Interrupt summary register (ISR) (see Section 5.1.24)
- Performance counter (PMCTR) register (see Section 5.1.27)
- CBU configuration control (CBOX_CONFIG2) register bits [13:08] (see Section 5.3.4)

2.8.1 CBU Performance Counters

The counters in the CBU (counters 0 and 1) are used to count Bcache and system bus events. There are request events from the MTU to the CBU (three types), requests from the CBU to the system (three types), and requests from the system to the CBU (four types).

MTU-to-CBU Requests

The MTU can issue the following requests:

- Istream read request (32 bytes of instruction data), due to an Icache miss
- Dstream read request (32 bytes of noninstruction data), due to a Dcache miss
- Write request (32 bytes)

Read and write requests can be to either cacheable or I/O space addresses, but the CBU performance counters only count requests to cacheable address space. The total number of read requests is equal to the sum of the Dstream read requests and the Istream read requests.

CBU-to-System Requests

The CBU can issue the following requests to the system:

- READ MISS commands
- BCACHE VICTIM commands

- WRITE BLOCK commands

READ MISS commands to I/O space and WRITE BLOCK commands (which are always to I/O space on the 21164PC) are not counted by the performance counters. BCACHE VICTIM commands are always to cacheable space and, therefore, are always counted. READ MISS commands to cacheable space are generated when the 21164PC detects either a read miss or write miss in the Bcache. A BCACHE VICTIM command is also generated along with the READ MISS command if the block the request misses on is valid and dirty in the cache. In this case, the 64-byte Bcache block is read from the Bcache and sent to the system.

System-to-CBU Requests

The system can issue the following requests to the 21164PC:

- FILL commands
- READ commands
- FLUSH commands
- INVAL commands

Cacheable FILL commands are in response to READ MISS commands and write 64 bytes of data into the Bcache. I/O space FILL commands are not counted by the CBU performance counters. Depending on whether the miss was for a read or write request, the 21164PC will either forward the data to the onchip caches or write data from the write buffer into the newly filled block. The total number of FILL commands is the same as the total number of READ MISS commands.

The other three system commands are external probes of the Bcache. INVAL commands are not counted by the CBU performance counter.

Misses in the onchip caches can merge in the MTU before being issued to the CBU. Therefore, MTU read or write requests are not the same as onchip cache misses. Also, two Bcache misses can merge in the CBU and appear on the system bus as a single READ MISS request. Requests only merge with other requests of the same type (that is, Istream and Dstream requests do not merge, nor does a write request merge with a read request).

Using the Counters

The two counters work in parallel, so they can be used to determine simple ratios like Bcache miss rate or more complex statistics like Dstream read merging in the CBU (by running several tests and normalizing the results).

For example:

Bcache miss rate	=	1 – (Bcache read hits/Total read requests) Counter 0 selects *0x0* and counter 1 selects *0x1*.
Dstream read merge rate in the CBOX	=	1 – (Bcache Dstream read hits/Bcache Dstream read requests) – (Bcache Dstream read fills/Bcache Dstream read requests) Counter 0 selects *0x1* and counter 1 selects *0x0* on the first pass, then counter 0 selects *0x2* and counter 1 selects *0x0* on the second pass.

2.9 Floating-Point Control Register

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

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

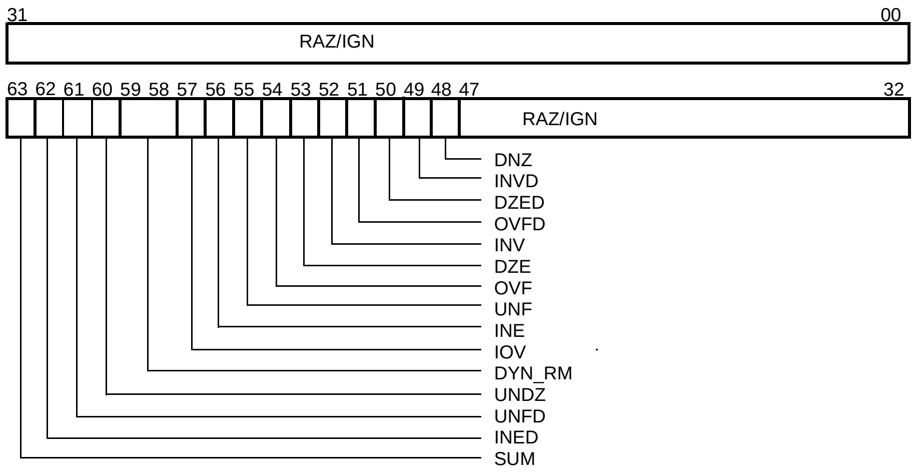


Table 2–10 Floating-Point Control Register Bit Descriptions *(Sheet 1 of 2)*

Name	Extent	Description (Meaning When Set)
SUM	[63]	Summary bit. Records bitwise OR of FPCR exception bits. Equal to FPCR[57 | 56 | 55 | 54 | 53 | 52]
INED	[62]	Inexact disable. Suppress INE trap and place correct IEEE nontrapping result in the destination register if the 21164PC is capable of producing correct IEEE nontrapping result.
UNFD	[61]	Underflow disable. Subset support: Suppress UNF trap if UNDZ is also set and the /S qualifier is set on the instruction.
UNDZ	[60]	Underflow to zero. When set together with UNFD, on underflow, the hardware places a true zero (all 64 bits zero) in the destination register rather than the denormal number specified by the IEEE standard.

Table 2–10 Floating-Point Control Register Bit Descriptions *(Sheet 2 of 2)*

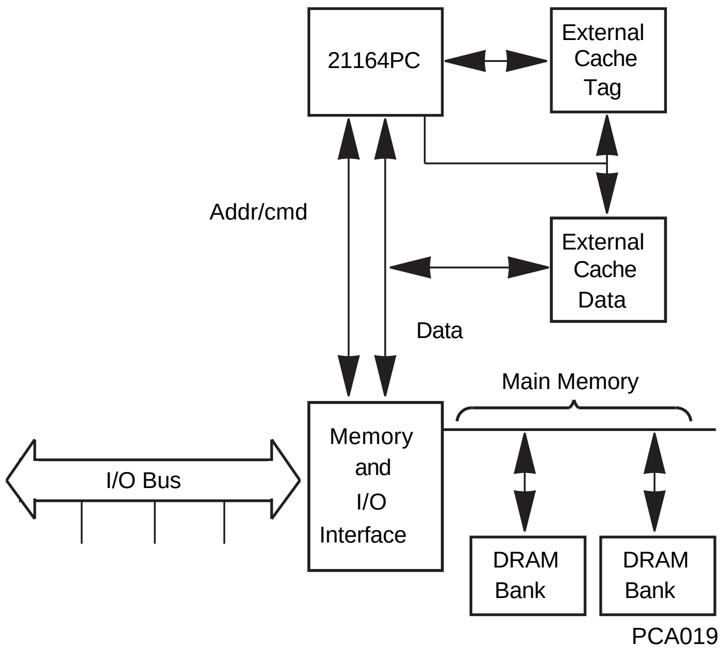
Name	Extent	Description (Meaning When Set)
DYN_RM	[59:58]	Dynamic routing mode. Indicates the rounding mode to be used by an IEEE floating-point operate instruction when the instruction's function field specifies dynamic mode (/D). The assignments are: **DYN** **IEEE Rounding Mode Selected** 00 Chopped rounding mode 01 Minus infinity 10 Normal rounding 11 Plus infinity
IOV	[57]	Integer overflow. An integer arithmetic operation or a conversion from floating to integer overflowed the destination precision.
INE	[56]	Inexact result. A floating arithmetic or conversion operation gave a result that differed from the mathematically exact result.
UNF	[55]	Underflow. A floating arithmetic or conversion operation underflowed the destination exponent.
OVF	[54]	Overflow. A floating arithmetic or conversion operation overflowed the destination exponent.
DZE	[53]	Division by zero. An attempt was made to perform a floating divide operation with a divisor of zero.
INV	[52]	Invalid operation. An attempt was made to perform a floating arithmetic, conversion, or comparison operation, and one or more of the operand values were illegal.
OVFD	[51]	Overflow disable. Not supported.
DZED	[50]	Division by zero disable.
INVD	[49]	Invalid operation disable.
DNZ	[48]	Division by signed zero. When set, if the instruction has /SU or /SUI qualifier, all denormal inputs for IEEE instructions are treated as signed zeros and the operation is completed in hardware without invoking INV trap. If the instruction has no /S qualifier, denormal inputs take INV trap.
Reserved	[47:0]	Reserved. Read as zero; ignored when written.

2.10 Design Examples

The 21164PC can be designed into many different uniprocessor system configurations. Figure 2–4 illustrates one possible configuration. This configuration employs additional system/memory controller chipsets.

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

Figure 2–4 Typical Uniprocessor Configuration



3 Hardware Interface

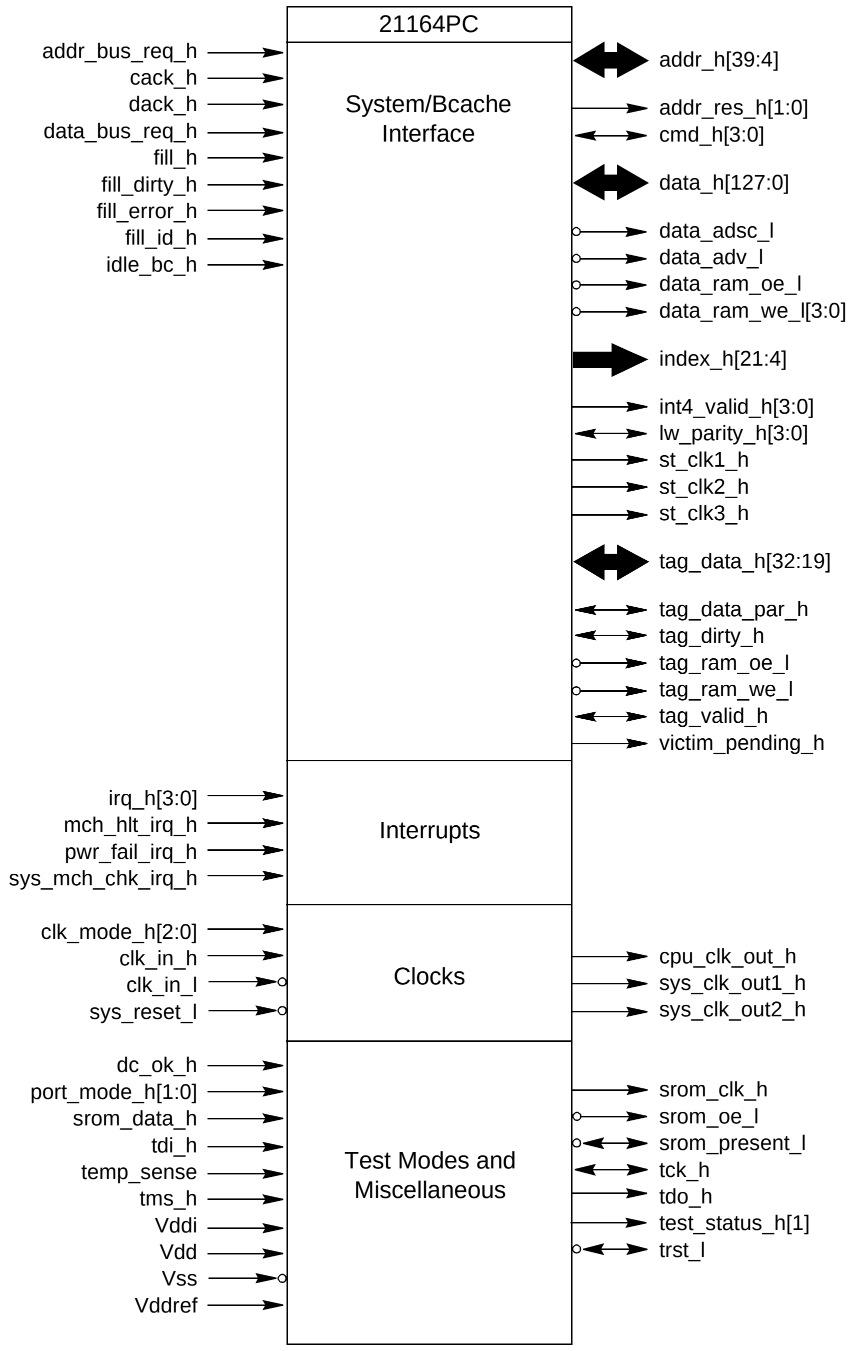
This chapter shows the 21164PC microprocessor logic symbol and provides a list of signal names and their functions.

3.1 21164PC Microprocessor Logic Symbol

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

21164PC Microprocessor Logic Symbol

Figure 3–1 21164PC Microprocessor Logic Symbol



3.2 21164PC Signal Names and Functions

The 21164PC is contained in a 413-pin interstitial pin grid array (IPGA) package. There are 265 functional signal pins, 26 spare signal pins (unused), 4 voltage reference pins, 23 external power (**Vdd**) pins, 34 internal power (**Vddi**) pins, 1 **Vdd_PLL** pin, and 60 ground (**Vss**) pins.

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

Signal Type	Definition
B	Bidirectional
I	Input only
O	Output only

21164PC Signal Names and Functions

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

Table 3–1 21164PC Signal Descriptions *(Sheet 1 of 10)*

Signal	Type	Count	Description
addr_h[39:4]	B	36	Address bus. These bidirectional signals provide the address of the requested data or operation between the 21164PC and the system. If **addr_h[39]** is asserted, then the reference is to noncached, I/O memory space. When the byte/word instructions are used and **addr_h[39]** is asserted, six additional bits of information are communicated over the pin bus. Two of the new bits are driven over **addr_h[38:37]**, becoming **transfer_size[1:0]**, with the following values: 00 Size = 8 bytes 01 Size = 4 bytes 10 Size = 2 bytes 11 Size = 1 byte
addr_bus_req_h	I	1	Address bus request. The system interface uses this signal to gain control of the **addr_h[39:4]** and **cmd_h[3:0]** pins (see Figure 4–26).
addr_res_h[1:0]	O	2	Address response bits [1] and [0]. For system commands, the 21164PC uses these pins to indicate the state of the block in the Bcache: **Bits / Command / Meaning** 00 / NOP / Nothing. 01 / NOACK / Data not found or clean. 10 / — / Reserved. 11 / ACK/Bcache / Data from Bcache.
cack_h	I	1	Command acknowledge. The system interface uses this signal to acknowledge any one of the commands driven by the 21164PC.

Table 3–1 21164PC Signal Descriptions *(Sheet 2 of 10)*

Signal	Type	Count	Description
clk_in_h **clk_in_l**	I I	1 1	Clock inputs. These signals provide the differential clock input that is the fundamental timing of the 21164PC. These signals are driven at the same frequency as the internal clock frequency (**clk_mode_h[2:0]** = 100). See Section 9.4.2.
clk_mode_h[2:0]	I	3	Clock test mode. These signals specify a relationship between **clk_in_h** and **clk_in_l**, the CPU cycle time, and the duty-cycle equalizer. These signals should be deasserted in normal operation mode.

Bits	Description
100	CPU clock frequency is equal to the input clock frequency.
101	CPU clock frequency is equal to the input clock frequency, with the onchip duty-cycle equalizer enabled.
x10	Initialize the CPU clock, allowing the system clock to be synchronized to a stable reference clock.
x11	Initialize the CPU clock, allowing the system clock to be synchronized to a stable reference clock, with the onchip duty-cycle equalizer enabled.
00x	CPU clock frequency is driven at the system bus frequency.

Table 3–1 21164PC Signal Descriptions *(Sheet 3 of 10)*

Signal	Type	Count	Description
cmd_h[3:0]	B	4	Command bus. These signals drive and receive the commands from the command bus. The following tables define the commands that can be driven on the **cmd_h[3:0]** bus by the 21164PC or the system. For additional information, refer to Section 4.1.1.1.

21164PC Commands to System:

cmd_h [3:0]	Command	Meaning
0000	NOP	Nothing.
0001	—	Reserved.
0010	—	Reserved.
0011	—	Reserved.
0100	—	Reserved.
0101	—	Reserved.
0110	WRITE BLOCK	Request to write a block.
0111	—	Reserved.
1000	READ MISS0	Request for data.
1001	READ MISS1	Request for data.
1010	—	Reserved.
1011	—	Reserved.
1100	BCACHE VICTIM	Bcache victim should be removed.
1101	—	Reserved.
1110	—	Reserved.
1111	—	Reserved.

Table 3–1 21164PC Signal Descriptions *(Sheet 4 of 10)*

Signal	Type	Count	Description

System Commands to 21164PC:

cmd_h [3:0]	Command	Meaning
0000	NOP	Nothing.
0001	FLUSH	Removes block from caches; return dirty data.
0010	INVALIDATE	Invalidates the block from caches.
0011	—	Reserved.
0100	READ	Read a block.
0101	—	Reserved.
0111	—	Reserved.
1xxx	—	Reserved.

Signal	Type	Count	Description
cpu_clk_out_h	O	1	CPU clock output. This signal is used for test purposes.
dack_h	I	1	Data acknowledge. The system interface uses this signal to control data transfer between the 21164PC and the system.
data_h[127:0]	B	128	Data bus. These signals are used to move data between the 21164PC, the system, and the Bcache.
data_adsc_l	O	1	Load a new address into the Bcache SSRAM.
data_adv_l	O	1	Advances the Bcache index to the next address.
data_bus_req_h	I	1	Data bus request. If the 21164PC samples this signal asserted on the rising edge of sysclk *n*, then the 21164PC does not drive the data bus on the rising edge of sysclk *n*+1. Before asserting this signal, the system should assert **idle_bc_h** for the correct number of cycles. If the 21164PC samples this signal deasserted on the rising edge of sysclk *n*, then the 21164PC drives the data bus on the rising edge of sysclk *n*+1. For timing details, refer to Section 4.10.4.
data_ram_oe_l	O	1	Data RAM output enable. This signal is asserted for Bcache read operations.

Table 3–1 21164PC Signal Descriptions *(Sheet 5 of 10)*

Signal	Type	Count	Description
data_ram_we_l[3:0]	O	4	Data RAM write-enable. These signals are asserted for any Bcache write operation. Refer to Section 5.3.1 for timing details.
dc_ok_h	I	1	dc voltage OK. Must be deasserted until dc voltage reaches proper operating level. After that, **dc_ok_h** is asserted.
fill_h	I	1	Fill warning. If the 21164PC samples this signal asserted on the rising edge of sysclk *n*, then the 21164PC provides the address indicated by **fill_id_h** to the Bcache on the rising edge of sysclk *n*+1. The Bcache begins to write in that sysclk. At the end of sysclk *n*+1, the 21164PC waits for the next sysclk and then begins the write operation again if **dack_h** is not asserted. Refer to Section 4.10.3 for timing details.
fill_dirty_h	I	1	Fill dirty. If the block being filled is dirty, this pin should be asserted.
fill_error_h	I	1	Fill error. If this signal is asserted during a fill from memory, it indicates to the 21164PC that the system has detected an invalid address or hard error. The system still provides an apparently normal read sequence with correct ECC/parity though the data is not valid. The 21164PC traps to the machine check (MCHK) PALcode entry point and indicates a serious hardware error. **fill_error_h** should be asserted when the data is returned. Each assertion produces a MCHK trap.
fill_id_h	I	1	Fill identification. Asserted with **fill_h** to indicate which register is used. The 21164PC supports two outstanding load instructions. If this signal is asserted when the 21164PC samples **fill_h** asserted, then the 21164PC provides the address from miss register 1. If it is deasserted, then the address in miss register 0 is used for the read operation.
idle_bc_h	I	1	Idle Bcache. When asserted, the 21164PC finishes the current Bcache read or write operation but does not start a new read or write operation until the signal is deasserted. The system interface must assert this signal in time to idle the Bcache before fill data arrives.
index_h[21:4]	O	18	Index. These signals index the Bcache.

Table 3–1 21164PC Signal Descriptions *(Sheet 6 of 10)*

Signal	Type	Count	Description
int4_valid_h[3:0]	O	4	INT4 data valid. During write operations to noncached space, these signals are used to indicate which INT4 bytes of data are valid. This is useful for noncached write operations that have been merged in the write buffer.

int4_valid_h[3:0]	Write Meaning
xxx1	**data_h[31:0]** valid
xx1x	**data_h[63:32]** valid
x1xx	**data_h[95:64]** valid
1xxx	**data_h[127:96]** valid

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

int4_valid_h[3:0]	Read Meaning
xxx1	**data_h[63:0]** valid
xx1x	**data_h[127:64]** valid
x1xx	**data_h[191:128]** valid
1xxx	**data_h[255:192]** valid

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

Table 3–1 21164PC Signal Descriptions *(Sheet 7 of 10)*

Signal	Type	Count	Description
			When **addr_h[39]** is asserted, the **int4_valid_h[3:0]** signals are considered the **addr_h[3:0]** bits required for byte/word transactions. The functionality of these bits is tied to the value stored in **addr_h[38:37]**.

For read transactions:

addr_h [38:37]	int4_valid_h[3:0] Value
00	Valid INT8 mask
01	**addr_h[3:2]** valid on **int4_valid_h[3:2]**; **int4_valid[1:0]** undefined
10	**addr_h[3:1]** valid on **int4_valid_h[3:1]**; **int4_valid[0]** undefined
11	**addr_h[3:0]** valid on **int4_valid_h[3:0]**

For write transactions:

addr_h [38:37]	int4_valid_h[3:0] Value
00	Valid INT4 mask
01	Valid INT4 mask
10	**addr_h[3:1]** valid on **int4_valid_h[3:1]**; **int4_valid[0]** undefined
11	**addr_h[3:0]** valid on **int4_valid_h[3:0]**

Table 3–1 21164PC Signal Descriptions *(Sheet 8 of 10)*

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

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

Signal	Type	Count	Description
lw_parity_h[3:0]	B	4	Longword parity. These signals set even INT4 parity for the current data cycle. Refer to Section 4.13.1 for information on the purpose of each **lw_parity_h** bit.
mch_hlt_irq_h	I	1	Machine halt interrupt request. This signal has multiple modes of operation. During initialization, this signal is used to set up **sys_clk_out2_ h** delay (see Table 4–3). During normal operation, it is used to signal a halt request.
port_mode_h[1:0]	I	2	Select test port interface modes (normal, manufacturing, and debug). For normal operation, both signals must be deasserted.
pwr_fail_irq_h	I	1	Power failure interrupt request. This signal has multiple modes of operation. During initialization, this signal is used to set up **sys_clk_out2_ h** delay (see Table 4–3). During normal operation, this signal is used to signal a power failure.

Table 3–1 21164PC Signal Descriptions *(Sheet 9 of 10)*

Signal	Type	Count	Description
srom_clk_h	O	1	Serial ROM clock. Supplies the clock that causes the SROM to advance to the next bit. The cycle time of this clock is 128 times the cycle time of the CPU clock.
srom_data_h	I	1	Serial ROM data. Input for the SROM.
srom_oe_l	O	1	Serial ROM output enable. Supplies the output enable to the SROM.
srom_present_l[1]	B	1	Serial ROM present. Indicates that SROM is present and ready to load the Icache.
st_clk1_h	O	1	STRAM clock. Clock for synchronously timed RAMs (STRAMs). For Bcache, this signal is synchronous with **index_h[21:4]** during private read and write operations, and with **sys_clk_out1_h** during read and fill operations.
st_clk2_h	O	1	This signal is a duplicate of **st_clk1_h**, to increase the fanout capability of the signal.
st_clk3_h	O	1	This signal is another duplicate of **st_clk1_h**, to increase the fanout capability of the signal.
sys_clk_out1_h	O	1	System clock output. Programmable system clock (**cpu_clk_out_h** divided by a value of 3 to 15) is used for board-level cache and system logic.
sys_clk_out2_h	O	1	System clock output. A version of **sys_clk_out1_h** delayed by a programmable amount from 0 to 7 CPU cycles.
sys_mch_chk_irq_h	I	1	System machine check interrupt request. This signal has multiple modes of operation. During initialization, it is used to set up **sys_clk_out2_h** delay (see Table 4–3). During normal operation, it is used to signal a machine interrupt check request.
sys_reset_l	I	1	System reset. This signal protects the 21164PC from damage during initial power-up. It must be asserted until **dc_ok_h** is asserted. After that, it is deasserted and the 21164PC begins its reset sequence.
tag_data_h[32:19]	B	14	Bcache tag data bits. This bit range supports .5MB to 4MB Bcaches.
tag_data_par_h	B	1	Tag data parity bit. This signal indicates odd parity for **tag_data_h[32:19]**.
tag_dirty_h	B	1	Tag dirty state bit. This bit is private to the 21164PC.

Table 3–1 21164PC Signal Descriptions *(Sheet 10 of 10)*

Signal	Type	Count	Description
tag_ram_oe_l	O	1	Tag RAM output enable. This signal is asserted during any Bcache read operation.
tag_ram_we_l	O	1	Tag RAM write-enable. This signal is asserted during any tag write operation.
tag_valid_h	B	1	Tag valid bit. During fills, this signal is asserted to indicate that the block has valid data. See Table 4–5 for information about Bcache protocol.
tck_h	B	1	JTAG boundary-scan clock.
tdi_h	I	1	JTAG serial boundary-scan data-in signal.
tdo_h	O	1	JTAG serial boundary-scan data-out signal.
temp_sense	I	1	Temperature sense. This signal is used to measure the die temperature and is for manufacturing use only. For normal operation, this signal must be left disconnected.
test_status_h[1]	O	1	Icache test status or timeout reset. This signal is used for manufacturing test purposes only to extract Icache test status information from the chip.
tms_h	I	1	JTAG test mode select signal.
trst_l[1]	B	1	JTAG test access port (TAP) reset signal.
victim_pending_h	O	1	Victim pending. When asserted, this signal indicates that the current read miss has generated a victim.

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

21164PC Signal Names and Functions

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

Table 3–2 21164PC Signal Descriptions by Function *(Sheet 1 of 3)*

Signal	Type	Count	Description
Clocks			
clk_in_h	I	1	Differential clock input.
clk_in_l	I	1	Differential clock input.
clk_mode_h[2:0]	I	3	Clock test mode.
cpu_clk_out_h	O	1	CPU clock output.
st_clk1_h	O	1	Bcache STRAM clock output.
st_clk2_h	O	1	Bcache STRAM clock output.
st_clk3_h	O	1	Bcache STRAM clock output.
sys_clk_out1_h	O	1	System clock output.
sys_clk_out2_h	O	1	System clock output.
sys_reset_l	I	1	System reset.
Bcache			
data_h[127:0]	B	128	Data bus.
data_adsc_l	O	1	Data RAM address load enable.
data_adv_l	O	1	Data RAM address advance enable.
data_ram_oe_l	O	1	Data RAM output enable.
data_ram_we_l[3:0]	O	4	Data RAM write-enable bits.
index_h[21:4]	O	18	Index.
lw_parity_h[3:0]	B	4	Data check.
tag_data_h[32:19]	B	14	Bcache tag data bits.
tag_data_par_h	B	1	Tag data parity bit.
tag_dirty_h	B	1	Tag dirty state bit.
tag_ram_oe_l	O	1	Tag RAM output enable.
tag_ram_we_l	O	1	Tag RAM write-enable.
tag_valid_h	B	1	Tag valid bit.

Table 3–2 21164PC Signal Descriptions by Function *(Sheet 2 of 3)*

Signal	Type	Count	Description
System Interface			
addr_h[39:4]	B	36	Address bus.
addr_bus_req_h	I	1	Address bus request.
addr_res_h[1:0]	O	2	Address response.
cack_h	I	1	Command acknowledge.
cmd_h[3:0]	B	4	Command bus.
dack_h	I	1	Data acknowledge.
data_bus_req_h	I	1	Data bus request.
fill_h	I	1	Fill warning.
fill_dirty_h	I	1	Fill dirty.
fill_error_h	I	1	Fill error.
fill_id_h	I	1	Fill identification.
idle_bc_h	I	1	Idle Bcache.
int4_valid_h[3:0]	O	4	INT4 data valid.
victim_pending_h	O	1	Victim pending.
Interrupts			
irq_h[3:0]	I	4	System interrupt requests.
mch_hlt_irq_h	I	1	Machine halt interrupt request.
pwr_fail_irq_h	I	1	Power failure interrupt request.
sys_mch_chk_irq_h	I	1	System machine check interrupt request.
Test Modes and Miscellaneous			
dc_ok_h	I	1	dc voltage OK.
port_mode_h[1:0]	I	2	Selects the test port interface mode (normal, manufacturing, and debug).
srom_clk_h	O	1	Serial ROM clock.
srom_data_h	I	1	Serial ROM data.

21164PC Signal Names and Functions

Table 3–2 21164PC Signal Descriptions by Function *(Sheet 3 of 3)*

Signal	Type	Count	Description
srom_oe_l	O	1	Serial ROM output enable.
srom_present_l[1]	B	1	Serial ROM present.
tck_h	B	1	JTAG boundary-scan clock.
tdi_h	I	1	JTAG serial boundary-scan data in.
tdo_h	O	1	JTAG serial boundary-scan data out.
temp_sense	I	1	Temperature sense.
test_status_h[1]	O	1	Icache test status or timeout reset.
tms_h	I	1	JTAG test mode select.
trst_l[1]	B	1	JTAG test access port (TAP) reset.

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

4 Clocks, Cache, and External Interface

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

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

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

4.1 Introduction to the External Interface

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

- 21164PC microprocessor
- External Bcache
- System interface logic

The 21164PC external interface is optimized for uniprocessor-based systems and mandates few design rules. The interface includes a 128-bit bidirectional data bus, a 36-bit bidirectional address bus, and several control signals.

Read latencies and data repetition rates of the external Bcache can be programmed by means of register bits. The Bcache clock frequency for private read and write operations is independent of the system interface clock frequency and makes for a more flexible design.

The cache system supports a 64-byte block size to the external Bcache.

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

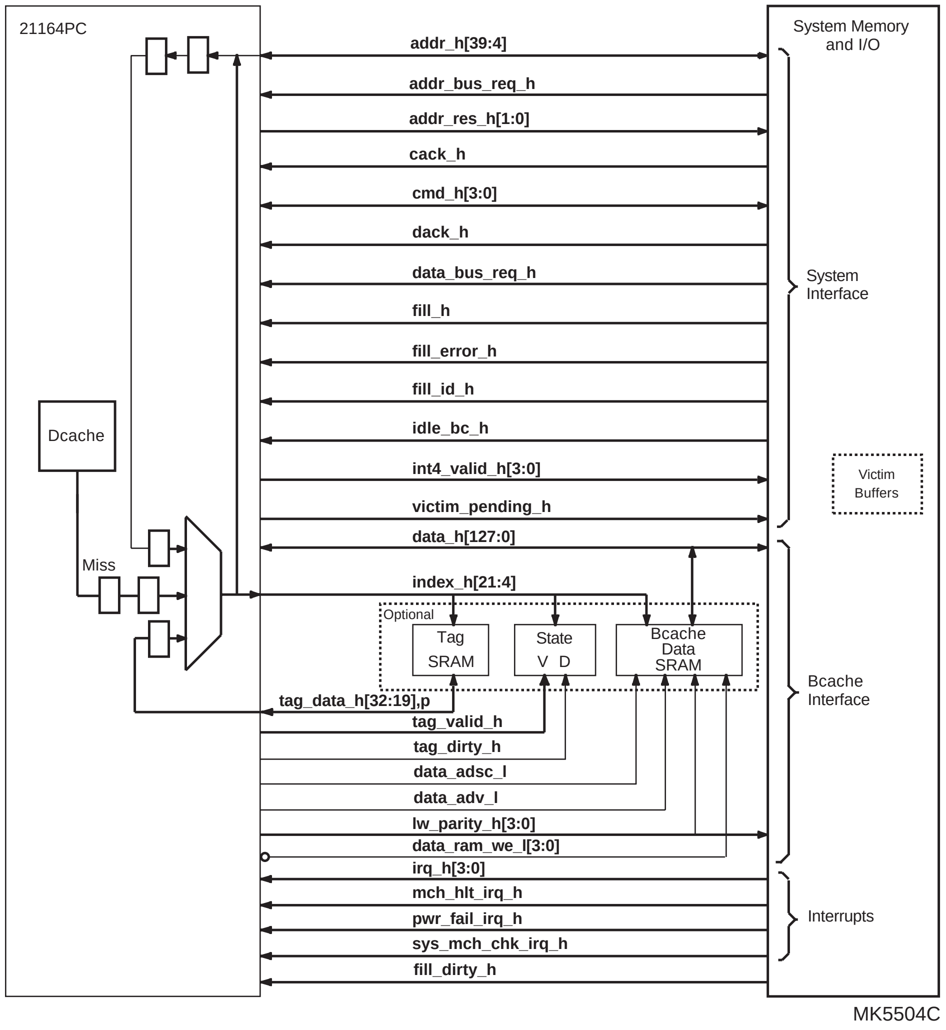
4.1.1 System Interface

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

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

The cycle time of the system interface is programmable to speeds of 4 to 15 times the CPU cycle time (**sysclk** ratio). All system interface signals are driven or sampled by the 21164PC on the rising edge of signal **sys_clk_out1_h**. In this chapter, this edge is sometimes referred to as "**sysclk**." Precisely when interface signals rise and fall does not matter as long as they meet the setup and hold times specified in Chapter 9.

Figure 4–1 21164PC System/Bcache Interface



4.1.1.1 Commands and Addresses

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

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

The BIU contains a three-entry BIU command/address buffer (BAF) capable of queueing up to three Bcache misses or I/O references. These buffers are capable of merging both read and write miss references, to reduce external system bus traffic.

4.1.2 Bcache Interface

The 21164PC includes an interface and control for a required backup cache (Bcache). The Bcache interface features the following:

- Support for pipelined and flow-through synchronous burst SRAMs (SSRAMs)
- Nonblocking, pipelined Bcache (up to three probes in flight)
- Fully interleaved writes to saturate write-hit traffic
- Flexible Bcache sizes (512KB – 4MB)
- Direct-mapped organization with 64-byte block size
- Read/write-allocate replacement policy
- Write-back cache policy
- A 128-bit data bus (shared with the system interface)
- 4.8-GB/s peak data transfer rate
- Programmable Bcache clock rate up to 300-MHz operation

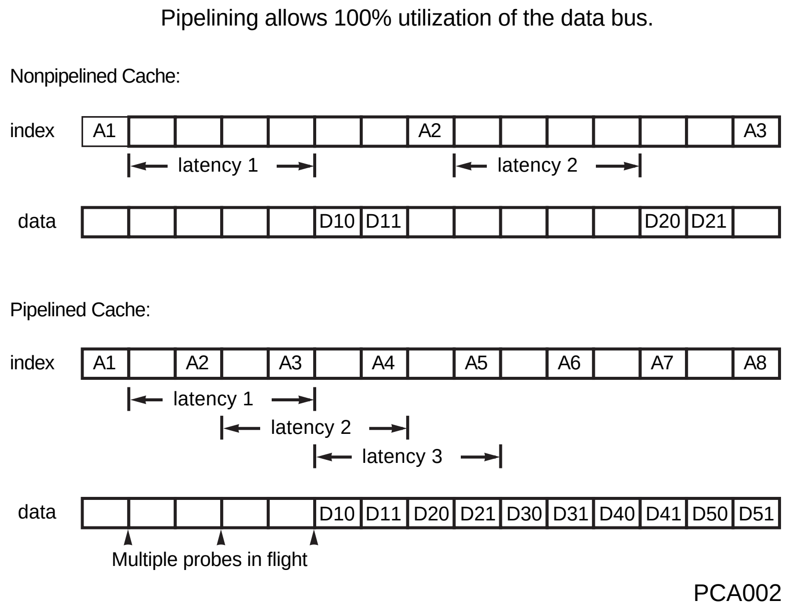
4.1.2.1 Bcache Interface Enhancements

With the advent of commodity SSRAMs, offchip high-speed caches can now be built at low cost to take advantage of the same performance techniques that until now had been restricted to onchip caches. The SSRAMs contain an address register, a self-incrementing address mechanism, and optionally, a data output register (pipelined). The 21164PC uses these additional control features to deliver a high-performance nonblocking, interleaved, fully pipelined Bcache interface.

4.1.2.2 Pipelined Bcache

A pipelined cache allows the processor to issue multiple cache operations that are overlapped in time to increase throughput. The 21164PC supports pipelining of up to three outstanding read or write probes at any given time to attain 100% data bus utilization. The outstanding Bcache probes are tracked by the BIU's “Bcache in flight” (or BIF) buffer. Figure 4–2 shows the benefits of having multiple probes in flight for a pipelined cache.

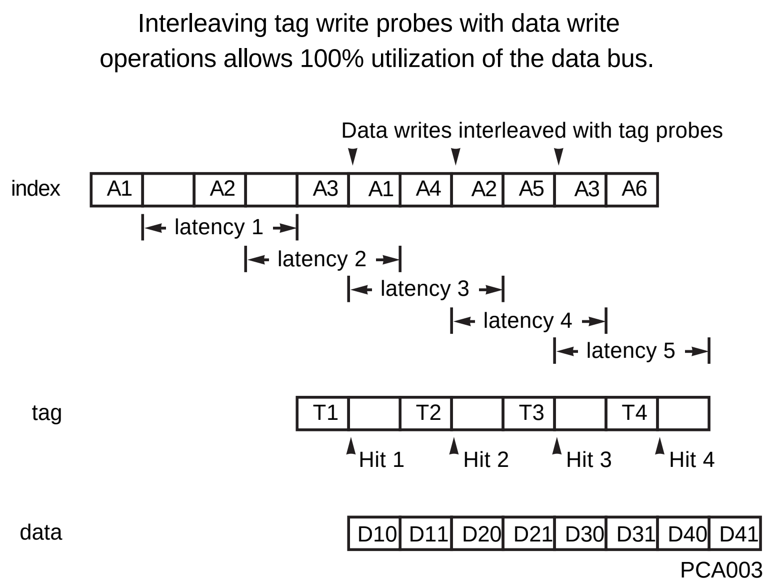
Figure 4–2 Merits of a Multiprobes In Flight – Pipelined Cache



4.1.2.3 Write Interleaving

The 21164PC Bcache interface takes advantage of the SSRAM address input register to employ interleaving techniques to maximize write-hit dirty bandwidth. The Bcache interface decouples the tag and data store control to allow tag write probes to be interleaved with data writes. Figure 4–3 shows an example of write interleaving and its ability to keep the data bus at 100% utilization.

Figure 4–3 Tag/Data Store Interleaving


Interleaving tag write probes with data write operations allows 100% utilization of the data bus.
Data writes interleaved with tag probes
index
A1
A2
A3
A1
A4
A2
A5
A3
A6
latency 1
latency 2
latency 3
latency 4
latency 5
tag
T1
T2
T3
T4
Hit 1
Hit 2
Hit 3
Hit 4
data
D10
D11
D20
D21
D30
D31
D40
D41
PCA003

Tag probes for writes that hit clean (valid, not dirty) in the Bcache must schedule a tag store write to update the dirty bit.

4.2 Clocks

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

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

The behavior of the programmable clocks during the reset sequence is described in Section 7.1.

4.2.1 CPU Clock

The 21164PC uses the differential input clock lines **clk_ in_h** and **clk_ in_l** as a source to generate its CPU clock. The input signals **clk_mode_h[1:0]** control generation of the CPU clock, as listed in Table 4–1 and as shown in Figure 4–4.

The 21164PC uses **clk_mode_h[0]** to provide onchip capability to equalize the duty cycle of the input clock (eliminating the need for a 2× oscillator). When **clk_mode_h[0]** is asserted, the equalizing circuitry, called a **symmetrator**, is enabled.

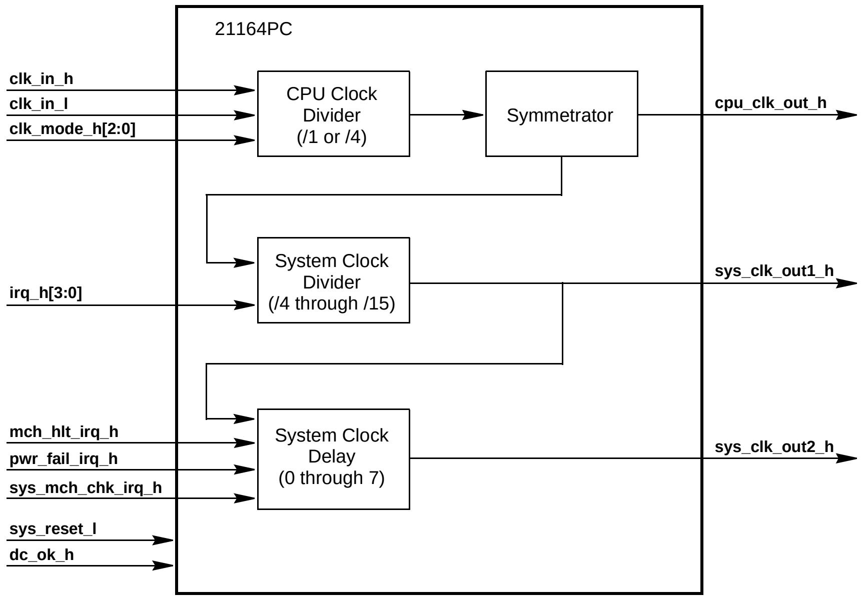
The 21164PC uses **clk_mode_h[1]** to reset the CPU clock. When **clk_mode_h[1]** is set, the internal CPU clock is reset to a known state. When it is clear, the CPU clock is driven at the same frequency as the **clk_in_h** and **clk_ in_l** differential input.

Table 4–1 CPU Clock Generation Control

Mode	clk_mode_h[1:0]	Description
Normal	0 0	CPU clock frequency is the same as the input clock frequency; symmetrator is disabled.
Normal	0 1	CPU clock frequency is the same as the input clock frequency; symmetrator is enabled. Also used to accommodate chip testers.
Reset	1 0	Initializes CPU clock, allowing system clock to be synchronized to a stable reference clock; symmetrator is disabled.
Reset	1 1	Initializes CPU clock, allowing system clock to be synchronized to a stable reference clock; symmetrator is enabled.

Caution: A clock source should always be provided on **clk_ in_h** and **clk_ in_l** when signal **dc_ok_h** is asserted.

Figure 4–4 Clock Signals and Functions



4.2.2 System Clock

The CPU clock is the source clock used to generate the system clock **sys_clk_out1_h**. The system clock divider controls the frequency of **sys_clk_out1_h**. The divisor, 4 to 15, is obtained from the four interrupt lines **irq_h[3:0]** at power-up as listed in Table 4–2. The system clock frequency is determined by dividing the ratio into the CPU clock frequency. Refer to Section 7.2 for information on **sysclk** behavior during reset. The value is also latched into the SYS_CLK_RATIO[3:0] field of the CBOX_STATUS IPR (bits [7:4]) for read-only purposes.

Table 4–2 System Clock Divisor *(Sheet 1 of 2)*

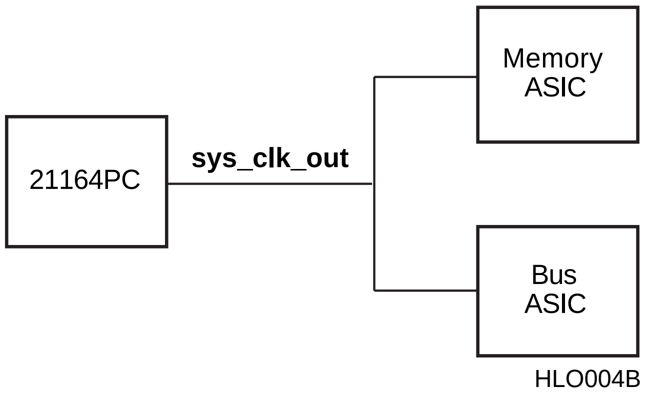
irq_h[3]	irq_h[2]	irq_h[1]	irq_h[0]	Ratio
Low	High	Low	Low	4
Low	High	Low	High	5
Low	High	High	Low	6
Low	High	High	High	7
High	Low	Low	Low	8

Table 4–2 System Clock Divisor *(Sheet 2 of 2)*

irq_h[3]	irq_h[2]	irq_h[1]	irq_h[0]	Ratio
High	Low	Low	High	9
High	Low	High	Low	10
High	Low	High	High	11
High	High	Low	Low	12
High	High	Low	High	13
High	High	High	Low	14
High	High	High	High	15

Figure 4–5 shows the 21164PC driving the system clock on a uniprocessor system.

Figure 4–5 21164PC Uniprocessor Clock



4.2.3 Delayed System Clock

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

Table 4–3 System Clock Delay

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

4.3 Physical Address Considerations

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

4.3.1 Physical Address Regions

Physical memory of the 21164PC is divided into three regions:

1. The first region is the first half of the physical address space. It is treated by the 21164PC as memory-like.
2. The second region is the second half of the physical address space except for a 1MB region reserved for CBU IPRs. It is treated by the 21164PC as noncacheable.
3. The third region is the 1MB region reserved for CBU IPRs.

In the first region, write merging and load merging are permitted. All 21164PC accesses in this region are 64-byte, the Bcache block size. This memory-like region is limited to 8GB (maximum).

The 21164PC does not cache data accessed in the second and third region of the physical address space; 21164PC read accesses in these regions are always INT32 requests. Load merging is permitted, but the request includes a mask to inform the system environment as to which INT8s are accessed. Write merging is permitted. Write accesses are INT32 requests with a mask indicating which INT4s are actually modified.

The 21164PC never writes more than 32 bytes at a time in noncached space.

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

Table 4–4 shows the 21164PC physical memory regions.

Table 4–4 Physical Memory Regions

Region	Address Range	Description
Memory-like	00 0000 0000 – 01 FFFF FFFF $_{16}$	Write invalidate cached, load, and store merging allowed.
Noncacheable	80 0000 0000 – FF FFEF FFFF $_{16}$	Not cached, load merging limited.
IPR region	FF FFF0 0000 – FF FFFF FFFF $_{16}$	Accesses do not appear on the interface unless an undefined location is accessed (which produces UNDEFINED results).

4.3.2 Data Wrapping

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

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

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

BCACHE VICTIM commands provide the data with the same wrap order as the read miss that produced them.

4.3.3 Noncached Read Operations

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

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

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

4.3.4 Noncached Write Operations

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

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

4.4 Bcache Structure

The 21164PC supports a .5, 1, 2, and 4MB Bcache. The size is under program control and is specified by CBOX_CONFIG[13:12] (BC_SIZE[1:0]). The Bcache block size is 64-byte blocks.

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

The 21164PC partitions physical address (**addr_h[32:4]**) into an index field and a tag field. The 21164PC presents **index_ h[21:4]** and **tag_data_h[32:19]** to the Bcache interface. The tag size required is Bcache_size/block_size.

The system designer uses the signal lines needed for a particular size Bcache. For example, the 1MB Bcache needs **index_h[19:4]** to address the cache block while the tag field would be **tag_data_h[32:20]**.

The 21164PC uses only the relevant tag address bits during the tag compare for the selected Bcache size. A larger Bcache has more index bits and fewer unused tag address bits, while a smaller Bcache has fewer index bits and more unused tag address bits. Unused index bits are driven to 0.

All private Bcache transactions operate on 32-byte subblocks. All system Bcache transactions (memory fill, Bcache victim, system commands that require data movement) operate on the 64-byte Bcache block size. The CPU data bus is 16 bytes wide (128 bits), thus private Bcache transactions require two data cycles and system Bcache transactions require four data cycles.

Longword write enables are provided to the data store for Bcache write operations. To support byte and word write transactions, the 21164PC performs a read-modify-write sequence at the Bcache interface.

4.4.1 Bcache Victim Buffers

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

External logic is required to maintain at least one victim buffer that acts as temporary storage that can be written faster and with lower latency than system memory. The victim buffer(s) hold Bcache victims and enable the Bcache location to be filled with data from the desired address. Data in the victim buffer(s) will be written to memory at a later time. This action reduces the time that the 21164PC is waiting for data.

4.5 Cache Coherency

Cache coherency rules must be followed when designing 21164PC-based uniprocessor systems as there are two levels of caches on a processor module that may be snooped for data by I/O devices.

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

The 21164PC requires the system to allow only one change to a block at a time. This means that if the 21164PC gains the bus to read or write a block, I/O devices on the system bus should not be allowed to access that block until the data has been moved.

Flush Cache Coherency Protocol

The 21164PC provides hardware mechanisms to support a flush-based cache coherence protocol. This protocol is best suited for low-cost uniprocessor systems. It is typically used by an I/O subsystem to ensure that data coherence is maintained when DMA transactions are performed. Flush protocol does not allow shared data in the cache. Table 4–5 shows the Bcache states for the cache coherency protocols.

Table 4–5 Bcache States for Cache Coherency Protocols

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

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

The Bcache is probed for each transaction to determine if the block is present. If the block is present, the requested action is taken. If the block is not present, the command is still acknowledged, but no other action is taken. The Flush protocol for the 21164PC does not support a duplicate tag store.

Section 4.5.1 provides a more detailed description of flush cache coherency protocol. The system commands that are used to maintain cache coherency are described in more detail in Section 4.8.2.

4.5.1 Flush Cache Coherency Protocol

System logic notifies the 21164PC of all DMA read operations that occur on the system bus by using the interface READ command. The 21164PC returns data if the block is dirty.

System logic notifies the 21164PC of all DMA write operations that occur on the system bus by using the interface FLUSH command. If the block is dirty, the 21164PC provides the data to the system and invalidates the block in the Bcache. If the block is not dirty (clean), data is not returned, and the block is invalidated.

System logic may choose to notify the 21164PC of full cache line DMA write operations that occur on the system bus by using the interface INVALIDATE system command. The 21164PC invalidates the Bcache block if the block was found.

Figure 4–6 shows the 21164PC cache state transitions that can occur as a result of transactions with the system. Figure 4–7 shows the 21164PC cache state transitions maintained by the 21164PC as a result of transactions by other nodes on the system bus. These two figures both represent the same state machine. They show transitions caused by the 21164PC, and by the system, separately for clarity.

Note : The abbreviations “I”, “M” and “E” indicate the /VALID, VALID and DIRTY, and the VALID and /DIRTY states, respectively.

Figure 4–6 Flush-Based Protocol 21164PC States

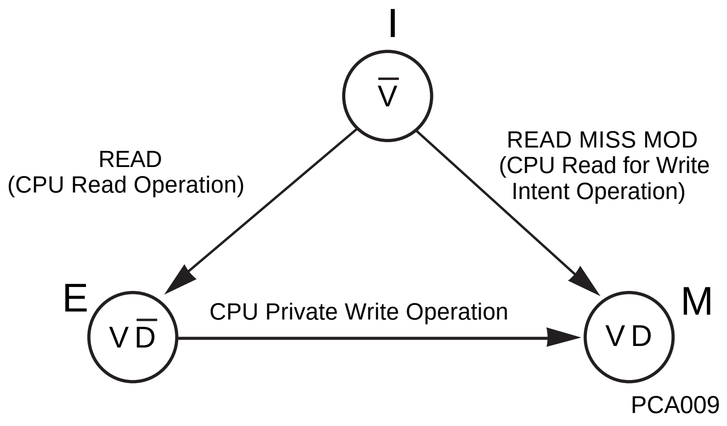
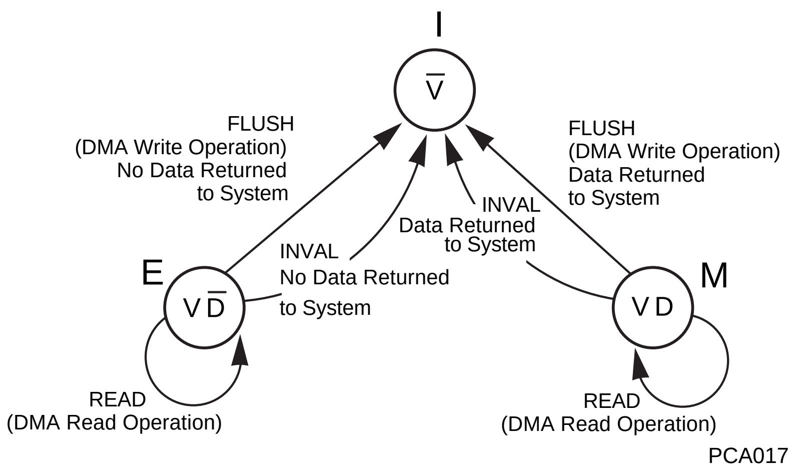


Figure 4–7 Flush-Based Protocol System/Bus States



4.6 21164PC-to-Bcache Transactions

When initiating an Istream or Dstream data transaction, the 21164PC first probes the Icache or Dcache, respectively. If the probe misses in the onchip caches, then the Bcache is probed.

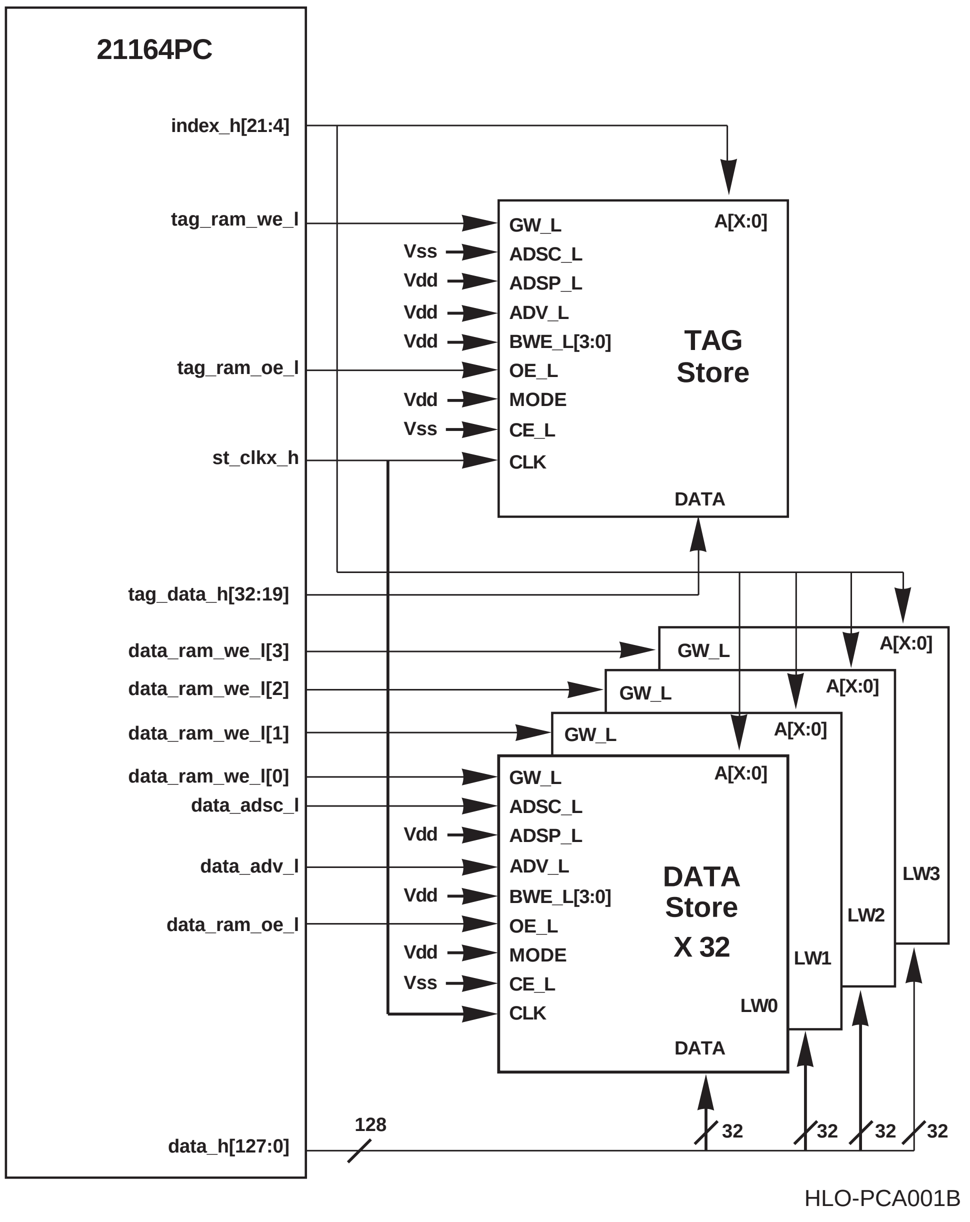
The 21164PC interface to the system and Bcache is controlled by the CBU. The CBU provides address and control signals for transactions to and from the Bcache and the system interface logic. The CBU also transfers data across the 128-bit bidirectional data bus.

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

4.6.1 Synchronous Burst-Mode Cache Support

The 21164PC supports both pipelined and flow-through SSRAMs. These SSRAMs provide several new control functions that are capitalized on to deliver a high-performance Bcache interface. All control pins driven from the 21164PC to the SSRAMs are synchronous (except the output enables) and are sampled relative to the SSRAM clock (**st_clk**). Figure 4–8 shows the SSRAM/Bcache interface.

Figure 4–8 SSRAM/Bcache Interface



For every Bcache access, the 21164PC drives the index, address strobe (**data_adsc_l**), and the SSRAM clock (**st_clk**) to the SSRAMs to load the initial address. The **st_clk** may be delayed a programmable number of CPU cycles to facilitate better control over module timing. For additional data reads or writes, the data advance (**data_adv_l**) is driven to the SSRAMs and is used to autoadvance the address in interleaved burst mode to facilitate the data wrap order described in Section 4.3.2.

For Bcache read and write probes, the tag store and data store have separate asynchronous output enables (**tag_ram_oe_l** and **data_ram_oe_l**) that control the SSRAM output drivers. A unique tag RAM output enable is required to facilitate the interleaved writes as described in Section 4.1.2.3. When switching from a Bcache read to a write transaction, the 21164PC provides a programmable read-to-write spacing to avoid data contention on the bidirectional tag and data buses (refer to Section 4.10.2 for more details).

For Bcache data writes, the 21164PC supports the **early write** protocol using the ADSC# pin (the ADSP# late-write is *not* supported). During data transfers, the 21164PC drives longword write enables (**data_ram_we_l[3:0]**) to the SSRAMs that correspond to the appropriate longword lanes within the 128-bit data bus. For byte and word granularity data writes, the 21164PC performs a read-modify-write operation to the Bcache. Tag store updates are necessary for memory fills and private writes that hit clean and are facilitated using the tag write enable (**tag_ram_we_l**).

There are subtle but important differences between the *pipelined* and *flow-through* SSRAMs that must be accounted for when interfacing to the 21164PC. The pipelined SSRAM (PBSRAM) includes an additional data output register that drives the data one **st_clk** later than the flow-through SSRAM. These differences and their effect on the 21164PC are explained in greater detail in Section 4.7.3.

4.6.2 Bcache Timing

The 21164PC provides a flexible Bcache interface with many programmable features, including the following:

- Programmable read latency
- Programmable data repetition rate
- Programmable **st_clk** delay
- Programmable read-to-write spacing

Bcache timing is configured using the CBOX_CONFIG and CBOX_CONFIG2 IPRs. Figures 5–48 and 5–51 show the layout of these registers. These registers are normally configured by 21164PC initialization code.

Both the 21164PC and system require access to the Bcache through a shared 128-bit data bus. When the 21164PC requires access to the Bcache (private mode), the **st_clk** is switched to the **bc_clk** regime where the clock is based on CBOX_CONFIG[BC_CLK_RATIO[3:0]. When the system requires access to the Bcache (system mode), the **st_clk** is switched to the **sysclk** regime where the clock is based on the **sysclk** ratio.

Table 4–6 describes the clocking regime and access type to the tag and data stores for each Bcache transaction.

Table 4–6 Bcache Transactions

		Bcache Access	
Transaction	**Clock Regime**	**Tag Store**	**Data Store**
Memory Fill	System	Write	Write
Bcache Victim	System	—	Read
System Data Movement	System	Write[1]	Read
CPU Read Probe	Private	Read	Read
CPU Write Probe	Private	Read	—
System Probe	Private	Read	—
CPU Data Write-Dirty	Private	—	Write
CPU Data Write-Clean	Private	Write	Write
Invalidate-Hit	Private	Write	—

[1] A tag store write during a system data movement is conditional.

System transactions include memory fills, Bcache victims, and system commands that require data movement. System transactions read and write the Bcache in the **sys_clk** regime (see Section 4.2.2). System Bcache read or write operations start relative to a **sysclk** edge. It is the responsibility of the system to control the rate of Bcache transactions by using the **dack_h** signal.

Private transactions include CPU-initiated read and write probes, CPU-initiated data writes, system probes, and system invalidates. Private transactions read and write the Bcache in the **bc_clk** regime (see Section 5.3.1). For private Bcache reads, both the latency and repetition rate are programmable using the CBOX_CONFIG register fields [11:08] (BC_LATENCY_OFF[3:0]) and [07:04] (BC_CLK_RATIO[3:0]). For private Bcache writes, the 21164PC uses the early write SSRAM protocol controlled by the ADSC# pin. The repetition rate for data writes is programmable through the (BC_CLK_RATIO[3:0]). Read and write operations that are private to the 21164PC and Bcache may start on any CPU clock.

There is no relation between **sysclk** and private Bcache latency. The **bc_clk** ratio is required to be less than or equal to the **sysclk** ratio for proper operation.

Note: Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.10.

When the 21164PC is operating in PLL mode (that is, **clk_mode_h[2:0]** = 00x), the onchip PLL accumulates 50 ps of phase error for each successive CPU cycle. The analysis of the Bcache loop timing must account for the accumulated phase error, since the Bcache timing is based on CPU cycle granularity.

4.6.3 Bcache Private Read Transaction

CPU-initiated reads must probe the Bcache to determine hit or miss status. CPU-initiated reads always perform the tag store lookup in parallel with a speculative 32-byte data store read.

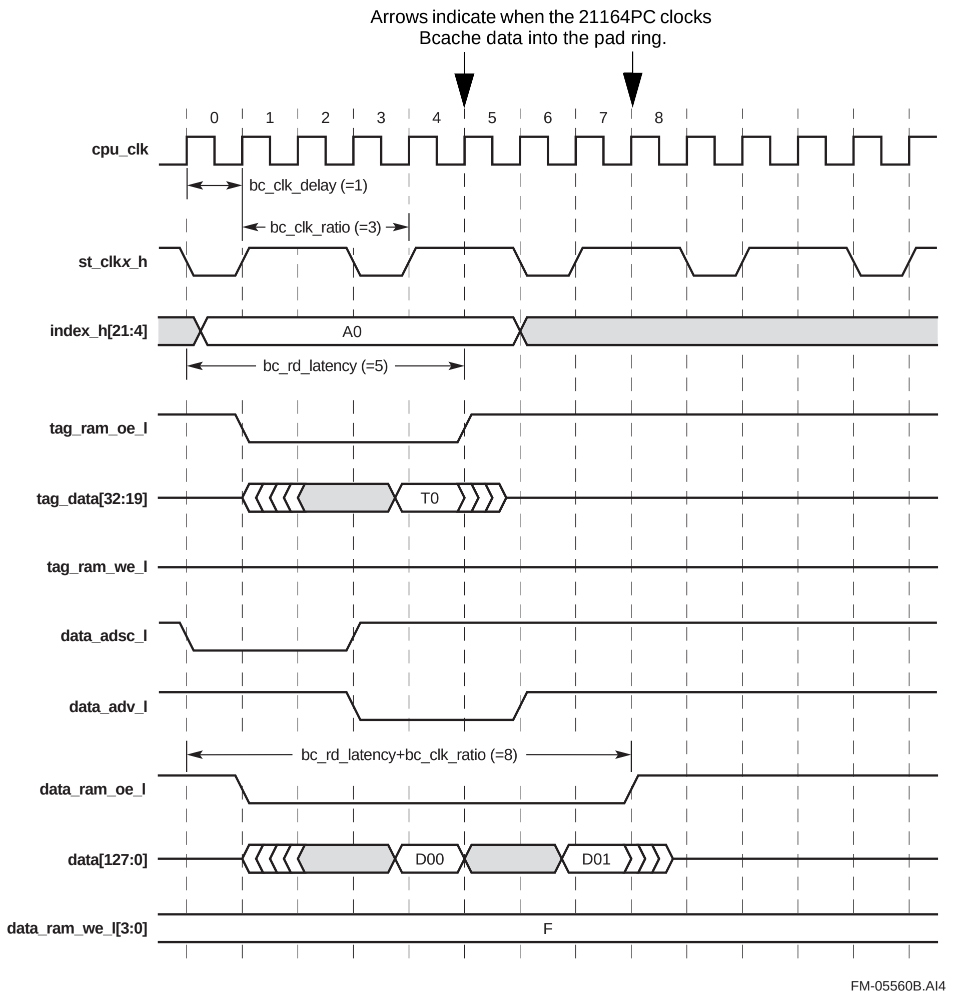
Figure 4–9 shows an example of the timing for a private read operation to Bcache by the 21164PC. CBOX_CONFIG[BC_LATENCY_OFF] = 0, which represents a minimum read latency value of five **cpu_clk** cycles.

The index is launched from an arbitrary internal **cpu_clk** edge (t=0). The data store address strobe, **data_adsc_l**, is also asserted at this time and is deasserted one **bc_clk** cycle later. The tag store address strobe is tied at the module level to always be asserted so that a new address is latched every **bc_clk** cycle. The data store address is autoadvanced for the next 16-byte data read with the assertion of **data_adv_l** one **bc_clk** cycle after the launch of the index. It is deasserted in the following **bc_clk** cycle.

The asynchronous tag RAM output enable, **tag_ram_oe_l**, is asserted at index launch plus one **cpu_clk** cycle and is deasserted *bc_rd_latency* **cpu_clk** cycles after the index launch.

The asynchronous data RAM output enable, **data_ram_oe_l,** is also asserted at index launch plus one **cpu_clk** cycle and is deasserted (*bc_rd_latency* + *bc_clk_ratio*) **cpu_clk** cycles after the index launch. All private Bcache operations return a 32-byte subblock (two data cycles).

Figure 4–9 Bcache Private Read Transaction



4.6.4 Bcache st_clk Timing

The SSRAM clock (**st_clk**) is asserted *bc_clk_delay* **cpu_clk** cycles from the index launch. The minimum pulse width for **st_clk** is dependent upon the CBOX_CONFIG[BC_CLK_RATIO] programmable parameter. For a *bc_clk_ratio* greater than 5, the **st_clk** remains asserted for three **cpu_clk** cycles. For a *bc_clk_ratio* from 3 to 5, the **st_clk** remains asserted for two **cpu_clk** cycles. For a *bc_clk_ratio* of 2, the **st_clk** remains asserted for one **cpu_clk** cycle.

An additional ½ cycle (or phase) delay can be realized if CBOX_CONFIG2[ST_CLK_PH_GR] = 1. This bit allows the system designer to fine tune the placement of the **st_clk** within the data valid window down to ½ CPU cycle granularity.

4.6.5 Bcache Private Write Transactions

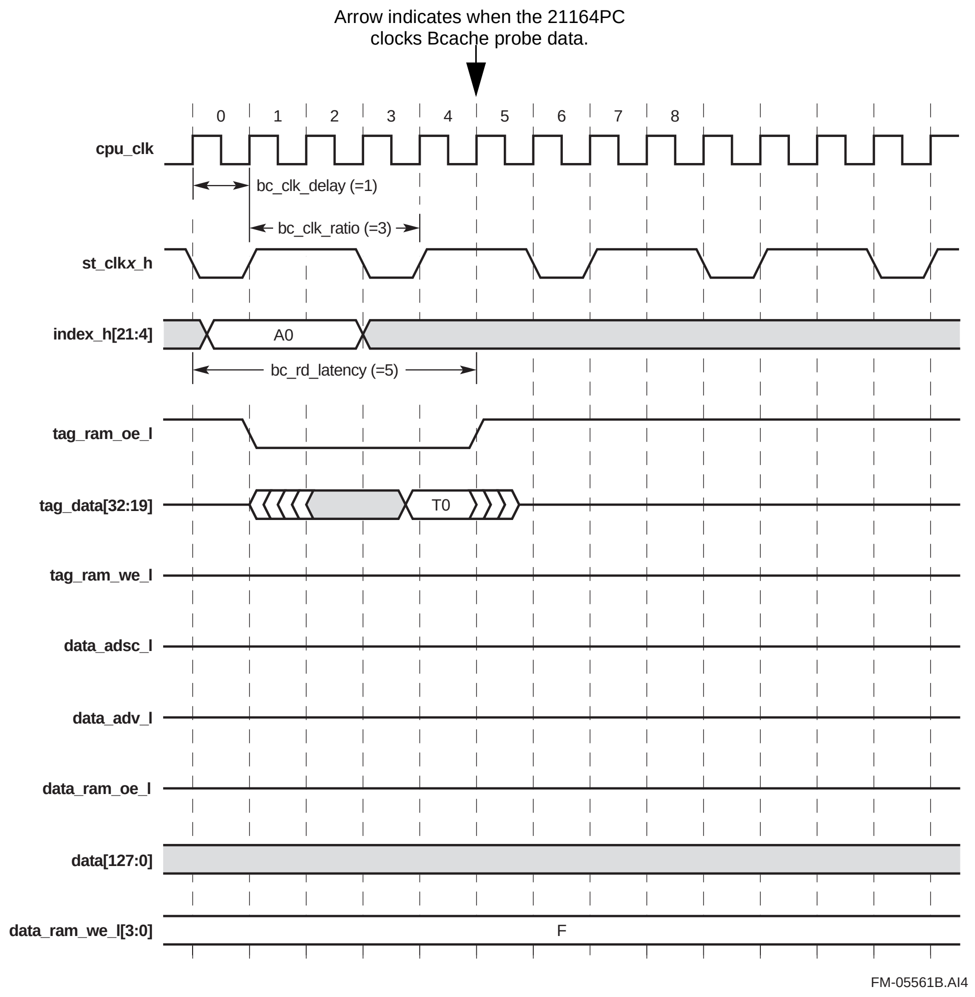
CPU-initiated write operations are broken into two suboperations, namely a write-probe operation and a subsequent data-write operation. The write-probe operation performs the tag store lookup to determine hit or miss status as well as to determine the tag state, clean (V */D) or dirty (V*D). Once a write hit has been detected, a data-write operation is performed to update the data store. Additionally, if the write-probe operation hits clean, the tag store must be updated to reflect dirty (or modified) status. If the write-probe operation hits dirty, then the tag store will not be updated. The two suboperations that make up a CPU-initiated write operation are not atomic and are pipelined with other read and write operations. Up to three Bcache probe operations can be in flight at any given time to increase overall Bcache performance.

4.6.5.1 Bcache Private Write-Probe Operation

Figure 4–10 shows an example of the timing for a write-probe operation to the Bcache by the 21164PC. CBOX_CONFIG[BC_CLK_RATIO] is set to three **cpu_clk** cycles.

The write-probe operation is identical to the Bcache read at the tag store interface. The index is launched from an arbitrary internal **cpu_clk** clock edge (t=0). The asynchronous tag RAM output enable, **tag_ram_oe_l**, is asserted at index launch plus one **cpu_clk** cycle and is deasserted *bc_rd_latency* **cpu_clk** cycles after the index launch.

Figure 4–10 Bcache Private Write Probe



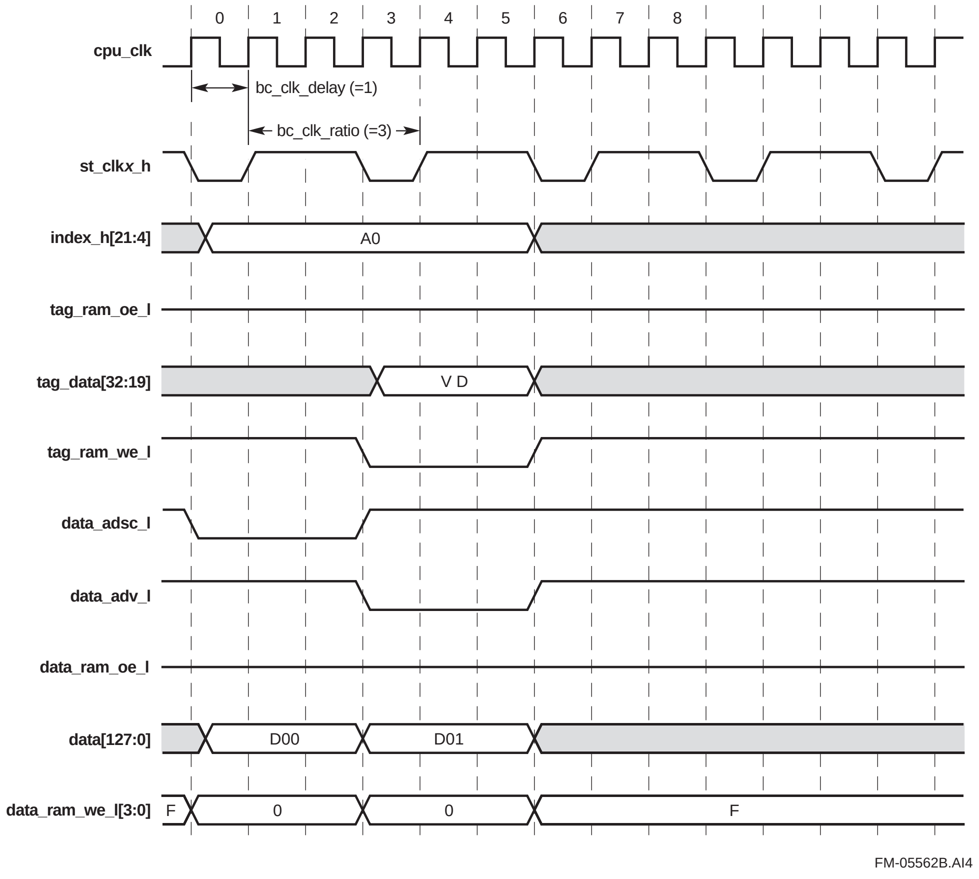
4.6.5.2 Bcache Private Data-Write Operation

If a CPU-initiated write command hits in the Bcache, the data-write operation is scheduled immediately. If the write hits clean, then the data-write operation must update the tag state to dirty. If the write hits dirty, then the data-write operation does not update the tag store.

If the CPU-initiated write command misses in the Bcache, the data-write is scheduled after the fill data from memory has returned. During the fill operation, the Bcache tag store is updated to reflect the new tag and control state (modified). Therefore, the data-write operation after the fill operation completes does not update the tag store. The Bcache is nonblocking and allows other transactions to use the Bcache while waiting for outstanding Bcache misses.

Figure 4–11 shows an example of the timing for a data-write operation that hits clean to the Bcache during the write probe. CBOX_CONFIG[BC_CLK_RATIO] is set to three **cpu_clk** cycles.

Figure 4–11 Bcache Private Data – Write Hit Clean

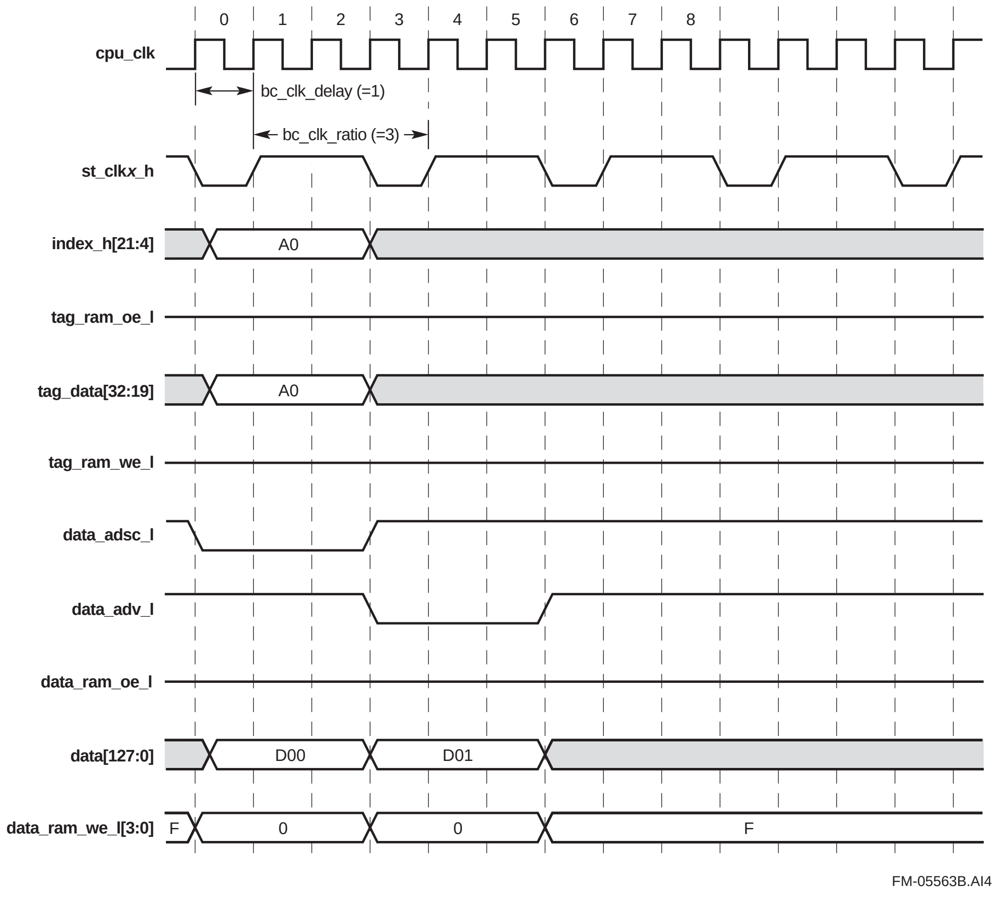


The index is launched from an arbitrary internal **cpu_clk** edge (t=0). The data store address strobe, **data_adsc_l**, is also asserted at this time and is deasserted one **bc_clk** cycle later. The tag store address strobe is tied at the module level to always be asserted so that a new address is latched every **bc_clk** cycle. The data store address is autoadvanced for the next 16-byte data write with the assertion of

data_adv_l one **bc_clk** cycle after the launch of the index. It is deasserted in the following **bc_clk** cycle. The longword write enables, **data_ram_we_l[3:0]**, are driven for each 16-byte of write data at index launch time and at the subsequent **bc_clk** cycle.

Figure 4–12 shows an example of the timing for a data-write operation that hits dirty to the Bcache during the write probe. CBOX_CONFIG[BC_CLK_RATIO] is set to three **cpu_clk** cycles. Note that the tag update is not required for a write hit to a dirty block.

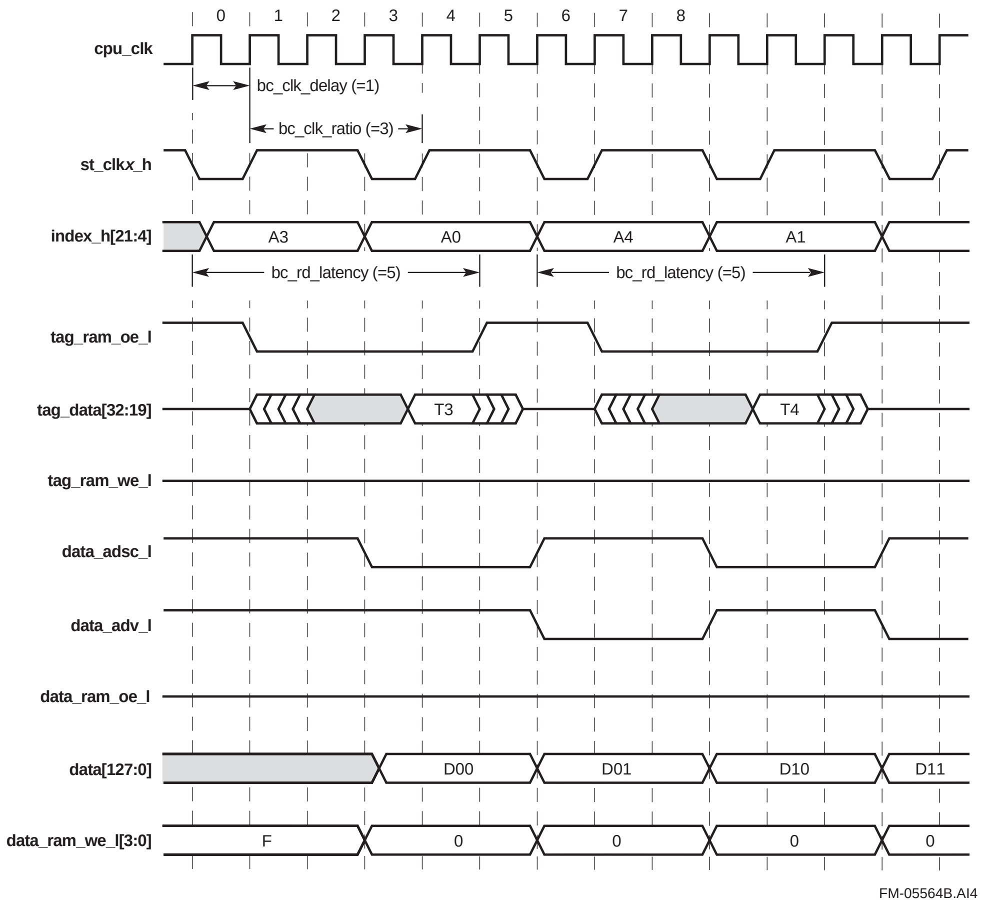
Figure 4–12 Bcache Private Data – Write Hit Dirty



4.6.5.3 Interleaving Write-Probes

The 21164PC is able to interleave data-write operations that hit dirty with write-probe operations, since both operations access different stores (tag and data). This technique is used to fully saturate the data bus during write-hit streams as is shown in Figure 4–13.

Figure 4–13 Bcache Interleaving



4.6.6 Selecting Bcache Options

Table 4–7 lists the variables to consider when designing and implementing a Bcache.

Table 4–7 Bcache Options

Parameter	Selection
sysclk ratio (4–15)	____ CPU cycles
Cache protocol, flush or flush invalidate	____
Longword parity or no parity	____
Bcache size (.5MB to 4MB)	____ MB
Bcache read latency (5–20)	____ CPU cycles
Bcache cycle time (2–10)	____ CPU cycles
Bcache victim buffer	Must be present
Bcache read-to-write spacing (1–8)	____
Bcache fill offset (1–8)	____
SSRAM type, pipelined or flow-through	____
st_clk delay (0–3)	____

4.7 21164PC-Initiated System Transactions

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

Note: Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.10.

The 21164PC starts an external transaction when:

- It encounters a “miss.”
- The CPU addresses a noncached region of memory.

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

- At the start of a Bcache transaction, the 21164PC checks the tag and tag control status of the target block.

- If there is a tag mismatch or the valid bit is clear, a Bcache miss has been detected. If the block to be replaced is clean, the Bcache continues operation while the READ MISS request is sent to the system. If the block to be replaced is dirty, the 21164PC waits for all outstanding probes in flight to complete, and then starts an external READ MISS with VICTIM PENDING transaction that instructs the system logic to access and return data.
- System logic acknowledges acceptance of the command from the 21164PC by asserting **cack_h**.
- Because the transaction is a read operation, requiring a fill operation, the transaction is broken (pended) while system logic obtains the fill data. The Bcache is nonblocking and allows other transactions to use the Bcache while a miss is being serviced.
- Prior to the fill data arriving, the system asserts **idle_bc_h** back to the 21164PC to arbitrate for the shared 128-bit data bus. Any private read or write operations in progress are allowed to complete before the fill data arrives from the system.
- At a later time, the system asserts **fill_h**.
- The 21164PC asserts the tag and tag control bits, and controls the write action during the fill operation.
- The system logic provides the data. As each of the two (or four) data cycles becomes valid, the system logic asserts **dack_h** to cause the 21164PC to sample the data and write it into the Bcache.

Interface commands from the 21164PC to the system are driven on the **cmd_h[3:0]** signals. Table 4–8 lists and describes the set of interface commands.

Table 4–8 21164PC-Initiated Interface Commands *(Sheet 1 of 2)*

Command	cmd_h [3:0]	Description
NOP	0000	The NOP command is driven by the owner of the **cmd_h** bus when it has no tasks queued.
—	0001	Reserved.
—	0010	Reserved.
—	0011	Reserved.
—	0100	Reserved.
—	0101	Reserved.

Table 4–8 21164PC-Initiated Interface Commands *(Sheet 2 of 2)*

Command	cmd_h [3:0]	Description
WRITE BLOCK	0110	Request to write a block. When the 21164PC wants to write a 32-byte block of data to noncached memory, it drives the command, address, and first INT16 of data on a **sysclk** edge. The 21164PC outputs the next INT16 of data when **dack_h** is received. When the system asserts **cack_h**, the 21164PC removes the command and address from the bus and begins the write of the Bcache. Signal **cack_h** can be asserted before all the data is removed.
—	0111	Reserved.
READ MISS0	1000	Request for data. This command indicates that the 21164PC has probed its caches and that the addressed block is not present.
READ MISS1	1001	Request for data. This command indicates that the 21164PC has probed its caches and that the addressed block is not present.
—	1010	Reserved.
—	1011	Reserved.
BCACHE VICTIM	1100	Bcache victim should be removed. If there is a victim buffer in the system, this command is used to pass the address of the victim to the system. The READ MISS command that produced the victim precedes the BCACHE VICTIM command. Signal **victim_pending_h** is asserted during the READ MISS command to indicate that a BCACHE VICTIM command is waiting, and that the Bcache is starting the read of the victim data.
—	1101	Reserved.
—	1110	Reserved.
—	1111	Reserved.

4.7.1 READ MISS Clean - No Victim

A READ MISS command is launched to the system interface when:

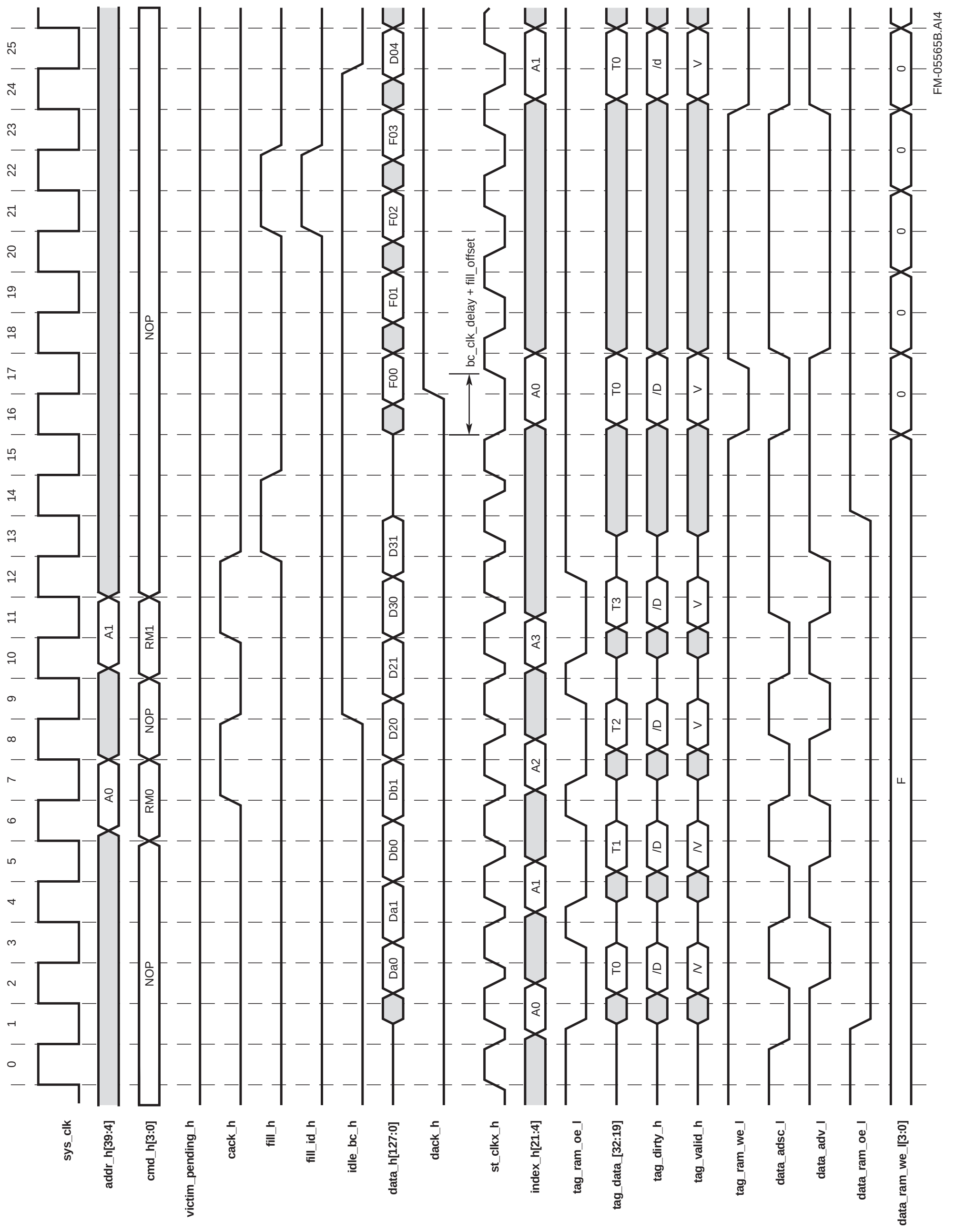
1. The Bcache probe for a CPU-initiated READ command detects a miss.
2. The Bcache probe for a CPU-initiated WRITE command detects a miss.
3. A CPU-initiated READ command to noncached memory space is detected.

The 21164PC starts a Bcache probe operation on any CPU clock. If a miss is detected to a clean or invalid block, then a victim does not need to be processed, and the address and command can be immediately issued to the system interface on the next **sysclk** edge. However, if a miss is detected to a dirty block, the CBU will require the data bus to process victims, and must wait for any in-flight probes to complete.

Figure 4–14 shows the timing of several Bcache reads and the resulting READ MISS Clean request. The system immediately asserts **cack_h** to acknowledge the command. This allows the 21164PC to make additional READ MISS requests. It is also possible for the system to defer assertion of **cack_h** until the fill data is returned. The assertion of **cack_h** should arrive no later than the last fill **dack_h**.

Note: A READ MISS command with **int4_valid_h[3:0]** of zero is a request for Istream data while **int4_valid_h[3:0]** of nonzero is a request for Dstream data.

Figure 4–14 READ MISS Clean – Bcache Timing Diagram



4.7.2 FILL

The 21164PC provides an **st_clk*x*_h** pulse a certain number of cycles after the rising edge of the system clock, determined by the sum of the BC_CLK_DELAY[1:0] and the FILL_OFFSET[2:0] values in the CBOX_CONFIG register (see Section 5.3.1). The value must be from 1 to 7 and cannot be greater than the **sysclk** ratio. This allows the SSRAM write operation to take place later in the **sysclk** cycle, allowing more time for the data to get to the 21164PC.

Signals **fill_h**, **fill_id_h**, and **fill_error_h** are used to control the return of fill data to the 21164PC and the Bcache. Signal **idle_bc_h** must be used to stop CPU requests in the Bcache in such a way that the Bcache will be idle when the fill data arrives (but not the FILL command). Signal **fill_h** must be asserted so that it is sampled by the CPU at least one **sysclk** period before the fill data is driven by the system. Signal **fill_id_h** should be asserted at the same time to indicate whether the fill operation is for a READ MISS0 or READ MISS1 operation. The 21164PC uses this information to select the correct fill address. Figure 4–14 shows the timing of a FILL command. Refer also to Section 4.10.3 for more information on using signals **idle_bc_h** and **fill_h**.

If **fill_h** is asserted at the rising edge of **sysclk** N, the 21164PC samples **fill_id_h**, then ensures that **data_h[127:0]** are tristated at the rising edge of **sysclk** N+1. Also at **sysclk** N+1, the 21164PC asserts the Bcache index, and begins a Bcache write operation. The system should drive the data onto the data bus and assert **dack_h** before the end of the **sysclk** cycle. If **dack_h** has not been asserted, the Bcache write operation starts again at the same index. If **dack_h** is asserted, the Bcache data-write operation starts again at the same index. If **dack_h** is asserted, the advance pin, **data_ram_adv_l**, is asserted, which advances the index to the next part of the fill, and the data-write operation begins again.

For all cacheable memory fill operations, the 21164PC updates the tag store in the same cycle that the Bcache index is driven, to reflect the new tag and control. Fill operations for READ commands update the tag store to the clean (V*/D) state, and fill operations for WRITE commands update the tag store to the dirty (V*D) state.

For system logic that returns fill data directly from its victim buffer without updating memory (Victim Buffer Fill Hit), the **fill_dirty_h** signal is used to remark the tag store to the dirty (V*D) state. This maintains data coherency. In systems that do not support Victim Buffer Fill Hits, it is recommended to tie the **fill_dirty_h** signal deasserted.

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

4.7.3 READ MISS with Victim

The 21164PC requires that the system contain a victim buffer to displace dirty blocks from the Bcache. The 21164PC requests the new block from memory while it starts to read the victim from the Bcache. The VICTIM command and address follow the miss request. This technique, known as “victims under fills,” allows the victim to be processed in parallel with the memory read.

When a miss is detected to a dirty block, the 21164PC waits for all outstanding Bcache probes in flight to complete. Then on the next **sysclk** edge, the 21164PC asserts a READ MISS command, the read miss address, the **victim_pending_h** signal, and indexes the Bcache to begin the read operation of the victim. When the system asserts **cack_h**, the 21164PC sends out a NOP command. In the following cycle the BCACHE VICTIM command is driven along with the victim address. Each assertion of **dack_h** causes the Bcache index to advance to the next part of the block. Figures 4–15 and 4–16 show the timing of a READ MISS command with a victim.

The 21164PC and system *must* treat a READ MISS/BCACHE VICTIM as an atomic transaction pair. Once the system has acknowledged the READ MISS command, it must guarantee ***not*** to start a system command or a FILL command until the READ MISS/BCACHE VICTIM atomic pair have completed. However, if the READ MISS command has not yet been acknowledged, the system is allowed to start a system command or a FILL command (by the assertion of **addr_bus_req_h** or **idle_bc_h**). The unacknowledged READ MISS command is retracted from the pin bus, and the system command or fill transaction is serviced at a higher priority. After the system or FILL command has completed, the READ MISS command is then replayed to the pin bus. For example, if the 21164PC sends the READ MISS command to the system, victim data can be removed from the Bcache without the READ MISS command being acknowledged. If the system sends an INVALIDATE system command to the same address before the READ MISS command is acknowledged, the 21164PC processes the INVALIDATE request and then restarts the READ MISS command from the beginning. The second time, the READ MISS command is issued to the pin bus without **victim_pending_h** asserted, because the data had been invalidated.

The use of **dack_h** for a system Bcache read command (Bcache victim or system command with data movement) is very dependent on the SSRAM style, either pipelined or flow-through. The assertion of **dack_h** is responsible for the assertion of the **data_adv_l** pin, and is not to be confused with the sampling of data. When using the pipelined SSRAMs, the data output register delays the data an additional **sysclk** cycle. When the CBOX_CONFIG[BC_REG_REG] bit is set, the **data_ram_oe_l** deassertion is delayed an additional **sysclk** cycle to allow the system ample time to sample the delayed Bcache read data.

System designers must also maintain the proper read-to-write spacing when going from BCACHE VICTIM commands to FILL commands. When using the pipelined SSRAMs, this delayed data on the victims impacts the earliest **fill_h** assertion (see Section 4.10.5.2 for more details).

Figure 4–15 READ MISS with Victim Timing Diagram, Pipelined Mode

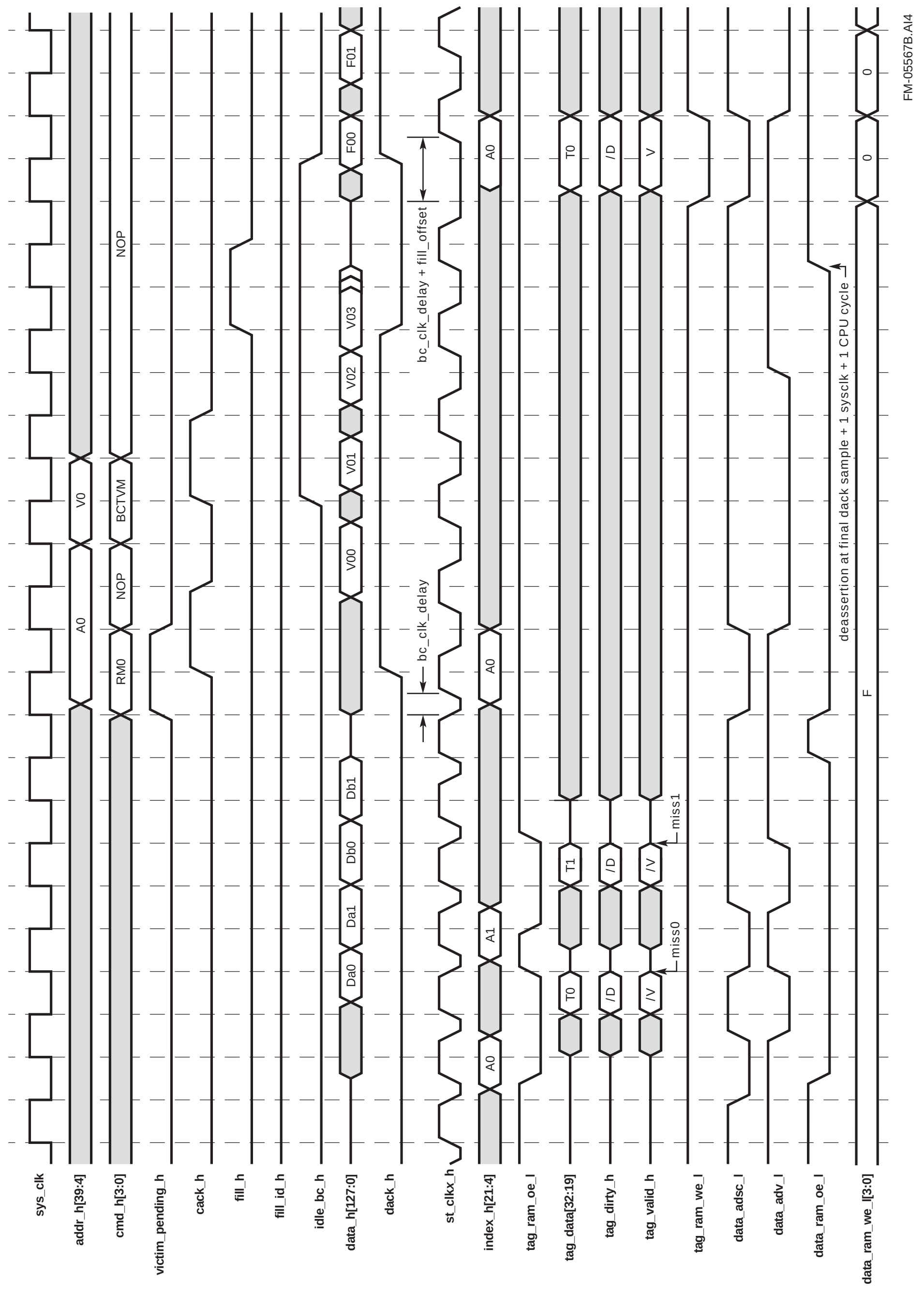
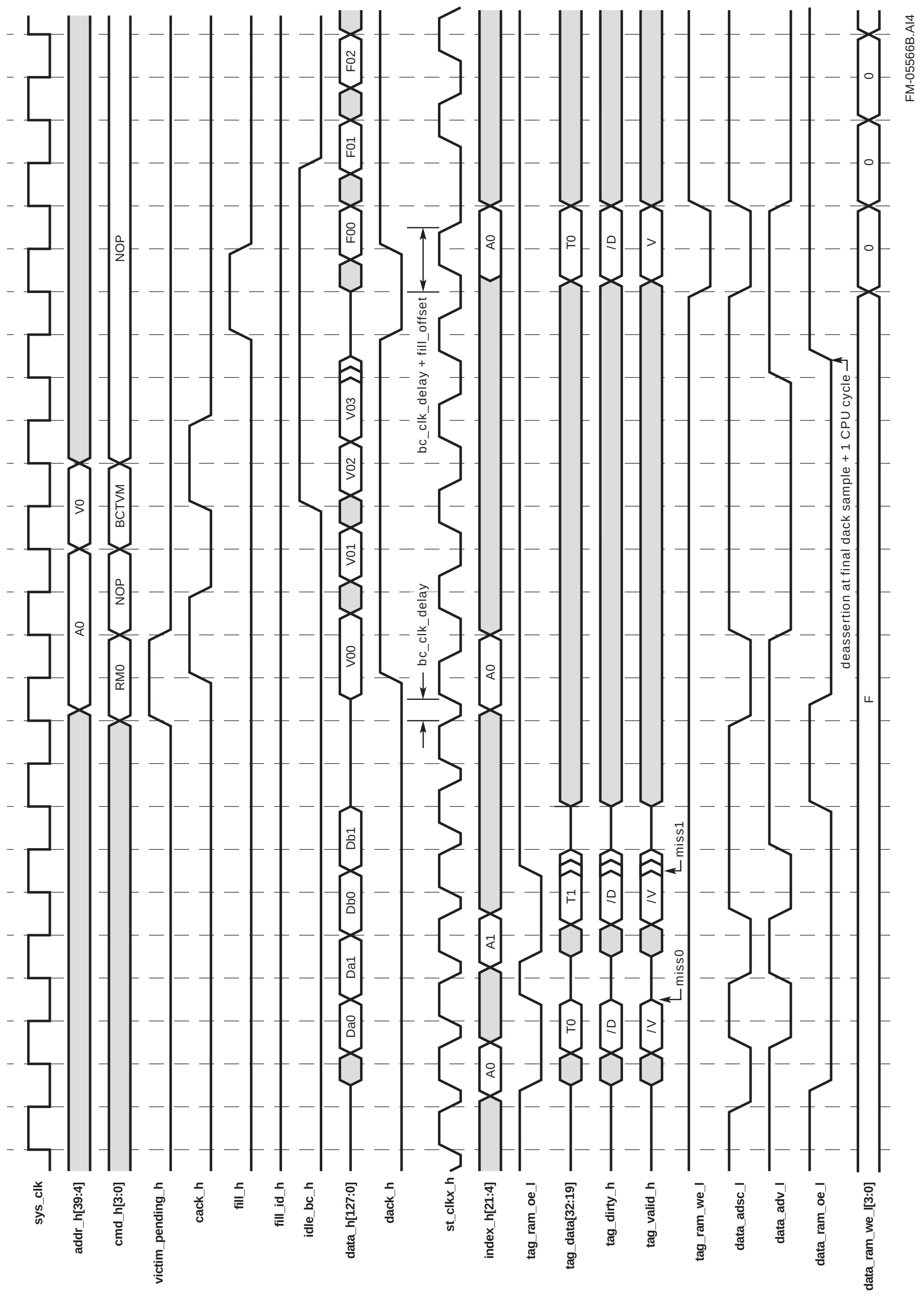


Figure 4–16 READ MISS with Victim Timing Diagram, Flow-Through Mode



4.7.4 WRITE BLOCK

The WRITE BLOCK command is used to complete write operations to noncached memory. The 21164PC asserts the WRITE BLOCK command, along with the address at the start of a **sysclk** cycle. The first 16 bytes of data and the **int4_valid** signals are driven one **cpu_clk** cycle later, so that system interface can be assured a one **cpu_clk** cycle minimum hold time when sampling data on the next **sysclk** edge. If the system removes ownership of the **cmd_h[3:0]** bus, the 21164PC retains the WRITE command and waits for bus ownership to be returned.

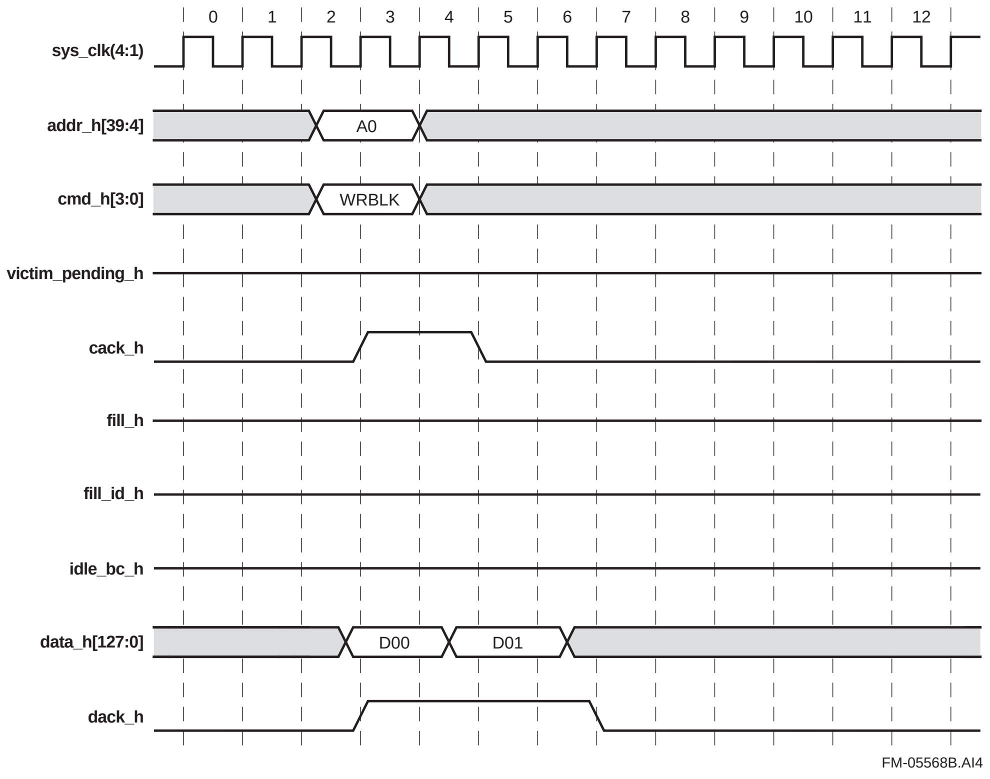
When the system takes the first part of the data, it asserts **dack_h**. This causes the 21164PC to drive the next 16 bytes of data on the same **sysclk** edge plus one **cpu_clk** cycle delay.

If the system asserts **cack_h**, the 21164PC outputs the next command in the next **sysclk**. Receipt of **cack_h** indicates to the 21164PC that the write operation will be taken.

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

Figure 4–17 shows the timing of a WRITE BLOCK command.

Figure 4–17 WRITE BLOCK Timing Diagram



4.8 System-Initiated Transactions

System commands to the 21164PC are driven on the **cmd_h[3:0]** signal lines. Before driving these signals, the system must gain control of the command and address buses by using **addr_bus_req_h**, as described in Section 4.10.1. The algorithm used by the 21164PC for accepting system commands to be processed in parallel by the 21164PC is presented in Section 4.8.1.

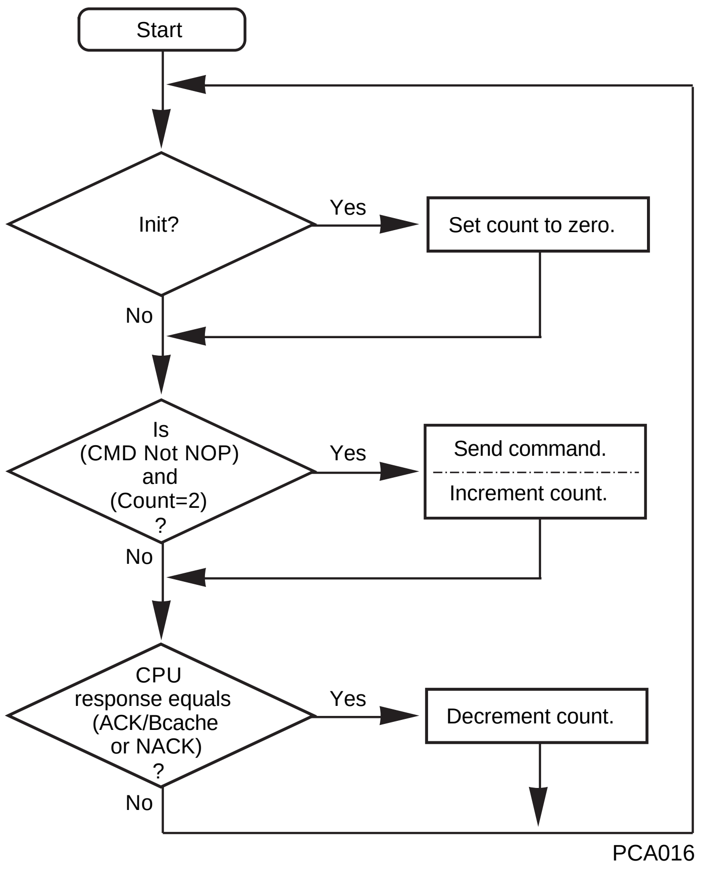
Note: Timing diagrams do not explicitly show tristated buses. For examples of tristate timing, refer to Section 4.10.

4.8.1 Sending Commands to the 21164PC

The rules used by the CBU BIU to process commands sent by the system to the 21164PC are listed in Section 4.12.1.

The 21164PC can hold two outstanding commands from the system at any time. The algorithm used by the system to send commands to the 21164PC without overflowing the two CBU BIU command buffers is shown in Figure 4–18.

Figure 4–18 Algorithm for System Sending Commands to the 21164PC



4.8.2 Write Invalidate Protocol Commands

All 21164PC-based systems that use the write invalidate protocol are expected to use the READ, FLUSH, and INVALIDATE commands to maintain cache coherency. These commands are defined in Table 4–9.

Table 4–9 System-Initiated Interface Commands (Write Invalidate Protocol)

Command	cmd_h [3:0]	Description
NOP	0000	The NOP command is driven by the owner of the **cmd_h[3:0]** bus when it has no tasks queued.
FLUSH	0001	Remove block from caches; return dirty data. The FLUSH command causes a block to be removed from the 21164PC cache system. If the block is not found, the 21164PC responds with NOACK. If the block is found and is clean, the 21164PC responds with NOACK. The block is invalidated in the Dcache and Bcache. If the block is found and is dirty, the 21164PC responds with ACK/Bcache and the Bcache read operation begins in the same **sysclk** cycle as the ACK. The block is invalidated in the Dcache and Bcache.
INVALIDATE	0010	Remove the block. When the system issues the INVALIDATE command, the 21164PC probes its Bcache. If the block is found, the 21164PC responds with ACK/Bcache and invalidates the block. If the block is not found, the 21164PC responds with a NOACK.
READ	0100	Read a block. The READ command probes the Bcache to see if the requested block is present. If the block is present and is dirty, the 21164PC responds with ACK/Bcache and the Bcache read operation begins in the same **sysclk** cycle as the ACK. If the block is not present or is present and clean, the 21164PC responds with a NOACK on **addr_res_h[1:0]**.

4.8.2.1 21164PC Responses to Flush-Based Protocol Commands

The system responds to flush-based protocol commands on **addr_res_h[1:0]**, as shown in Table 4–10.

Table 4–10 21164PC Responses to Flush-Based Protocol Commands

READ and FLUSH Commands	
Bcache	**21164PC Response**
Bcache Miss	NOACK
Bcache Hit, Not Dirty	NOACK
Bcache Hit, Dirty	ACK/Bcache

4.8.2.2 FLUSH

The FLUSH command is used to remove blocks from the 21164PC cache system. Figure 4–19 shows the timing of a FLUSH transaction.

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

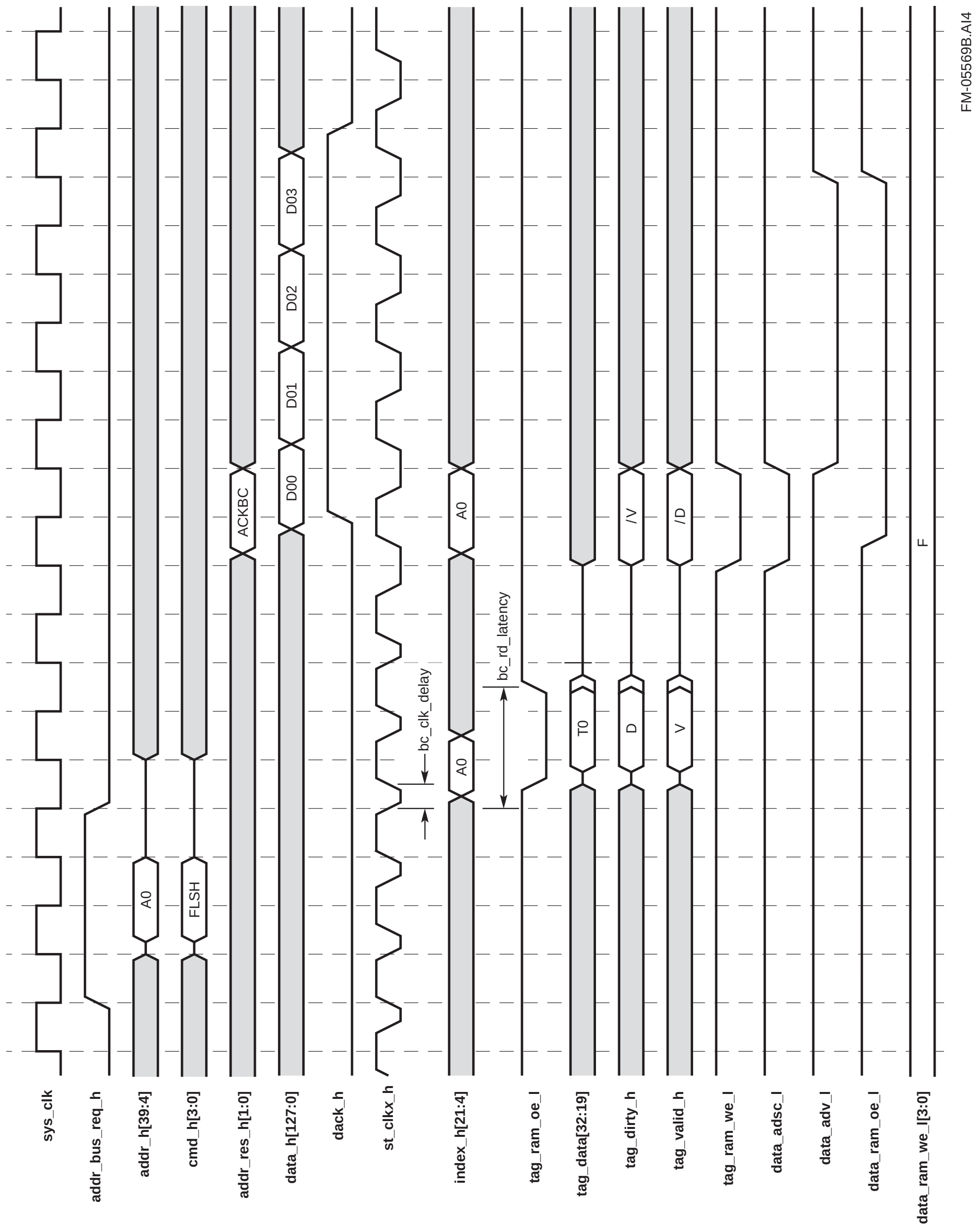
In the timing diagram shown in Figure 4–19, the cache block state changes from DIRTY, VALID to $\overline{\text{DIRTY}}$, $\overline{\text{VALID}}$. When the block state changes to $\overline{\text{VALID}}$, the state of DIRTY does not matter.

If the block is clean, the 21164PC invalidates both the Dcache and Bcache and responds to the system with a NOACK. If the block is not found, the 21164PC responds to the system with a NOACK.

The system probe is performed in private mode, and if data is found dirty in the Bcache, the subsequent tag invalidate and data movement are performed in system mode.

When using the pipelined SSRAMs, the data output register delays the data an additional **sysclk** cycle. When the CBOX_CONFIG[BC_REG_REG] bit is set, the **data_ram_oe_l** deassertion is delayed an additional **sysclk** cycle to allow the system ample time to sample the delayed Bcache read data.

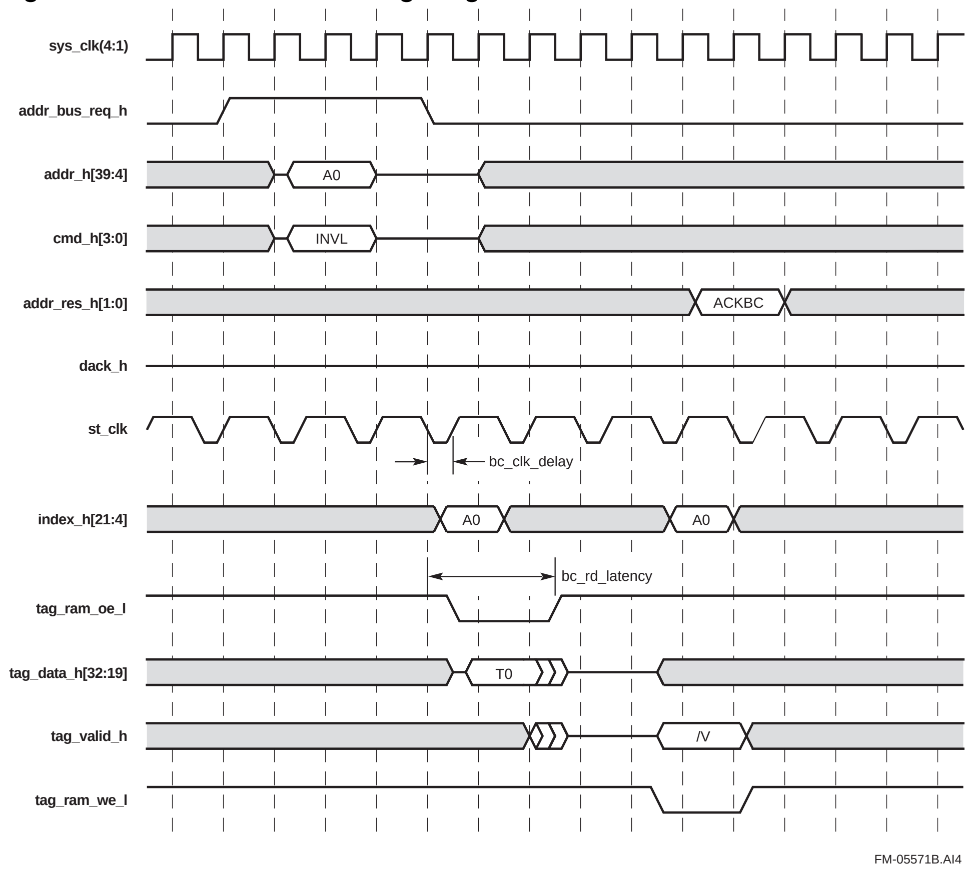
Figure 4–19 FLUSH Timing Diagram (Bcache Hit) Flow-Through SSRAM



4.8.2.3 INVALIDATE

The INVALIDATE command can be used to remove a block from the cache system. Unlike the FLUSH command, any modified data will not be read. The Bcache is probed and invalidated if the block is found. Figure 4–20 shows the timing of an INVALIDATE transaction. Both the system probe and the invalidate are performed in private mode to reduce overall latency.

Figure 4–20 INVALIDATE Timing Diagram – Bcache Hit

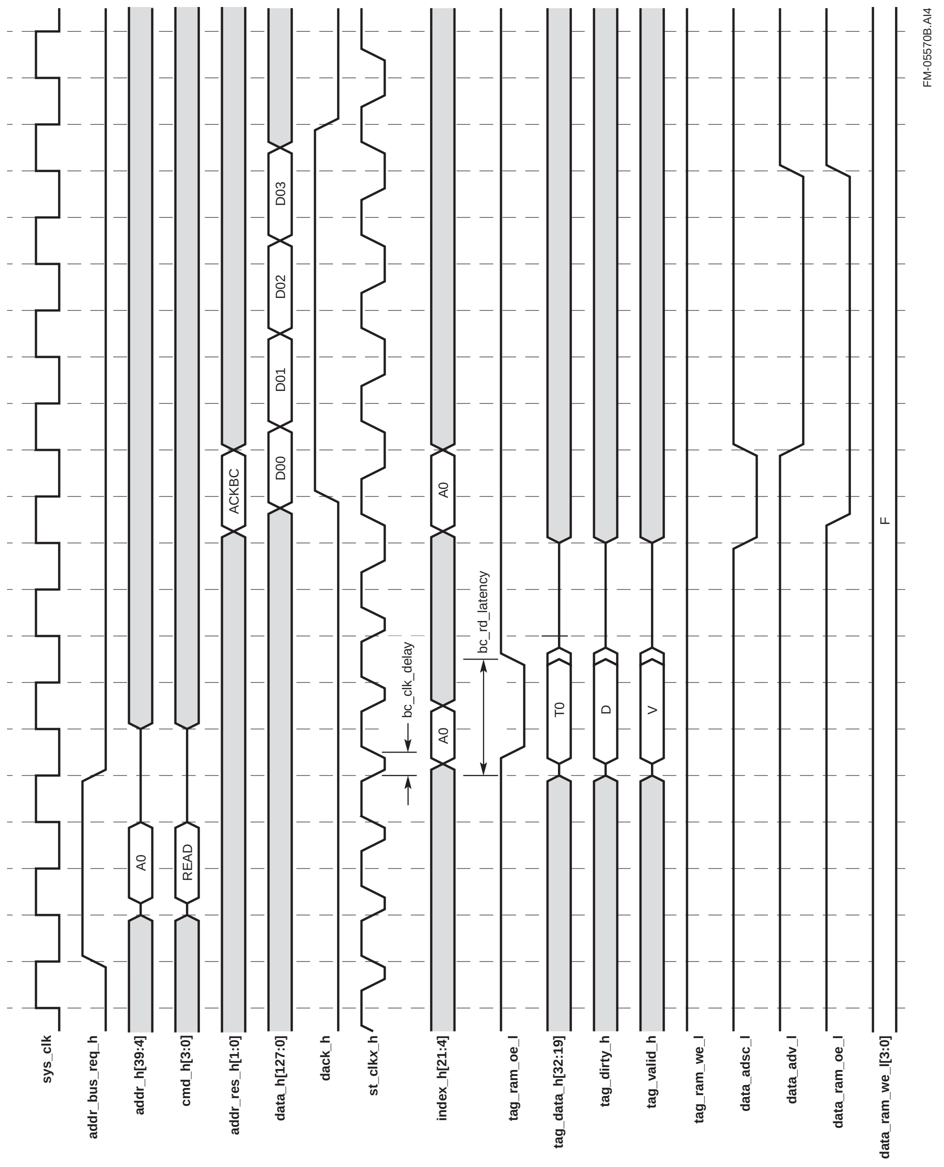


4.8.2.4 READ

The READ command is used by the system to read dirty data from the 21164PC. The tag control status does not change. Figure 4–21 shows the timing and tag control status of a read transaction.

When using the pipelined SSRAMs, the data output register delays the data an additional **sysclk** cycle. When the CBOX_CONFIG[BC_REG_REG] bit is set, the **data_ram_oe_l** deassertion is delayed an additional **sysclk** cycle to allow the system ample time to sample the delayed Bcache read data.

Figure 4–21 READ Timing Diagram (Bcache Hit) Flow-Through SSRAM



4.9 Memory Performance Enhancements

The 21164PC microprocessor includes two new, memory performance-enhancement features that, when used together, are capable of fully saturating the memory data bus by taking advantage of the burst SSRAM capability. The internal burst address latch of the PBSRAM at the data store allows the index bus to be time multiplexed, which allows tag store probes to run coincident with data store writes.

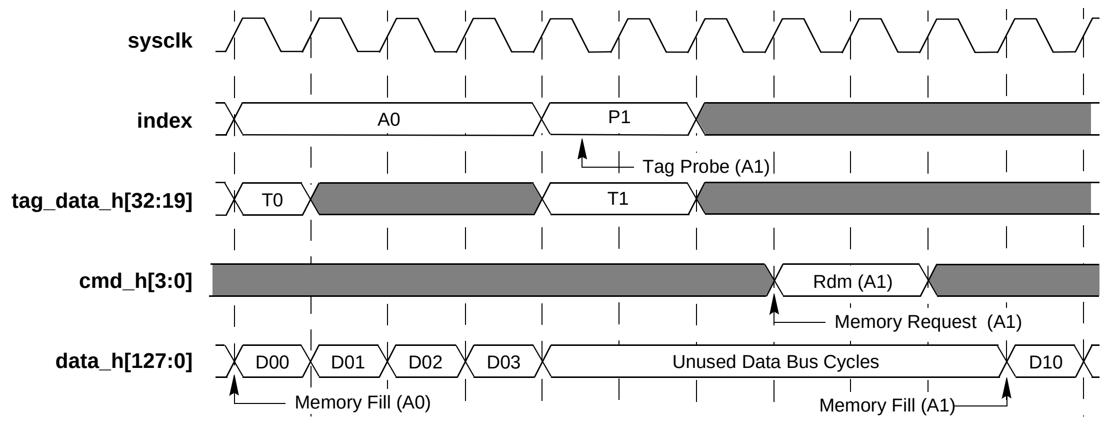
4.9.1 Three Read Miss

When the CBOX_CONFIG2[BC_THREE_MISS] bit is enabled, a third outstanding read-miss reference is allowed to the system. To use this feature, the system must guarantee that memory fills are returned in the order in which they were requested. This feature allows three outstanding memory read-miss requests to be in flight to the memory subsystem. When used in conjunction with the BC_PROBE_UNDER_FILL feature (described in Section 4.9.2), the 21164PC is capable of 100% data bus utilization during streaming read misses.

4.9.2 Probe Under Fill

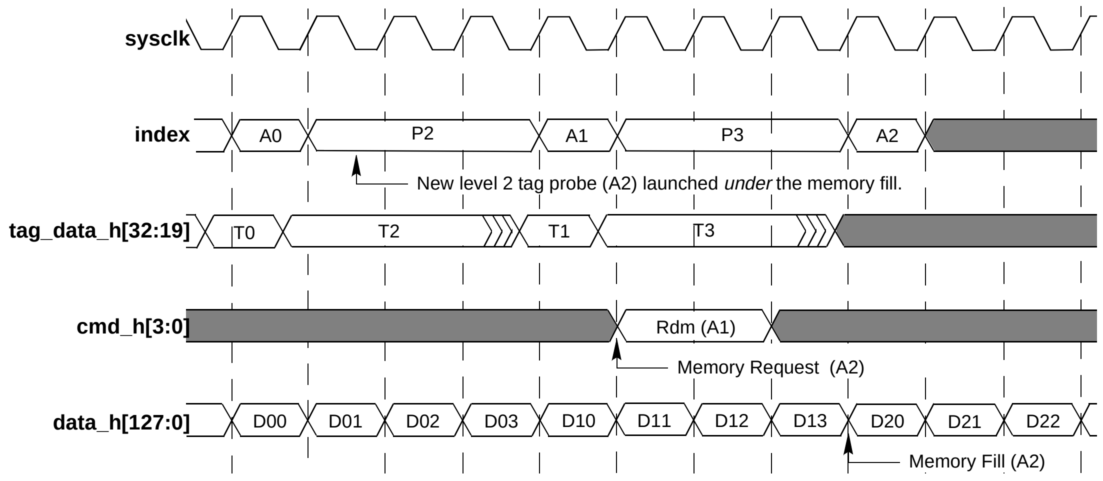
When CBOX_CONFIG[BC_PROBE_UNDER_FILL] is enabled, new tag probes can be launched to the tag store in parallel with (or under) memory fills to the data store. This feature allows more read misses to merge to the onchip miss buffer (BAF) and allows read-miss commands to be launched to the system earlier, having a net effect of increasing read-miss bandwidth. Figures 4–22 and 4–23 show a comparison of memory fills with and without probe under fill.

Figure 4–22 Typical Memory Fill Without BC_PROBE_UNDER_FILL



Note the unused data-bus cycles when BC_PROBE_UNDER_FILL is disabled. Figure 4–23 shows the data-bus usage with the feature enabled.

Figure 4–23 Memory Fill With BC_PROBE_UNDER_FILL



To use the probe under fill performance-enhancement feature, the system designer must ensure the following rules are enforced:

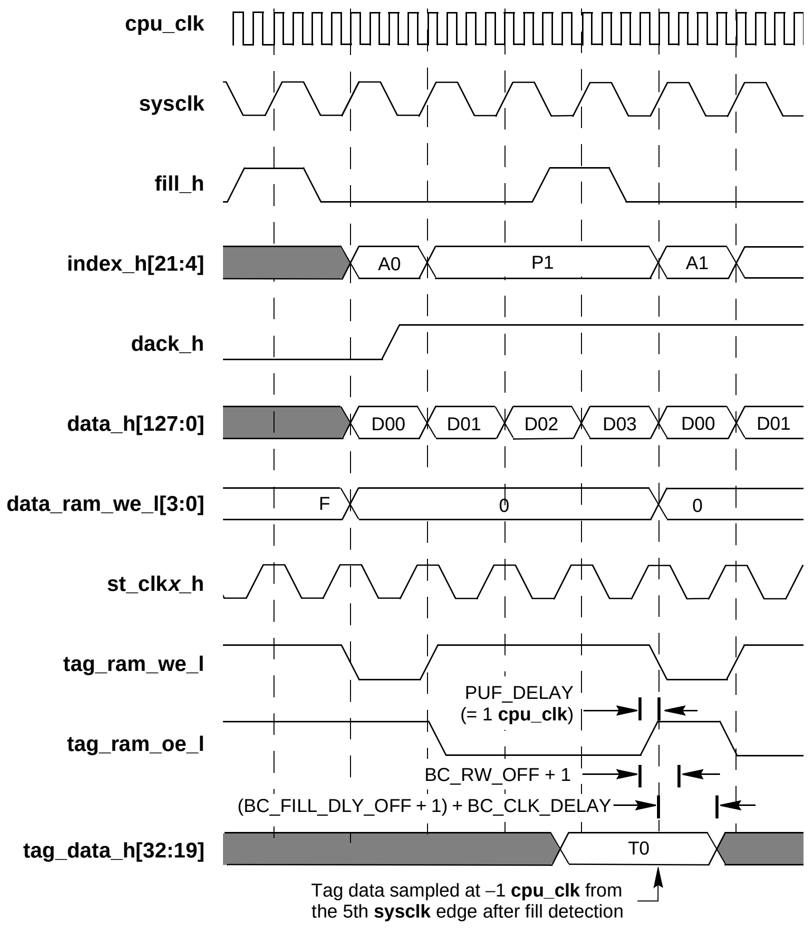
1. BC_CLK_DELAY + BC_FILL_DLY_OFF + PUF_DELAY ≥ BC_RW_OFF

   This rule must be satisfied to avoid an R⇒W tag store collision for the back-to-back fill case, when the probe under fill tag-store read is followed directly by the next fill's tag store update (see Figure 4–24).

   To help alleviate potential tag store R⇒W collisions, the **tag_ram_oe_l** can be deasserted either 0 or 1 CPU cycles before the 5th **sysclk** edge from the fill detection, based on the value in CBOX_CONFIG2[PUF_DELAY]. Once **tag_ram_oe_l** is deasserted for the probe under fill, the 21164PC always waits (BC_RW_OFF + 1) CPU cycles before driving the tag-data bus for the next fill's tag update. The system designer must ensure that there is adequate tag-data setup to the tag store to account for the deferred drive.

   Figure 4–24 is an example of this.

Figure 4–24 PUF_DELAY = 1



2. tKQ(SSRAM) ≤ 3(SYSCLK_RATIO) – PUF_DELAY – (SYSCLK_RATIO) × REG_REG – (BC_CLK_DELAY + BC_FILL_DLY_OFF + 1) – tflt

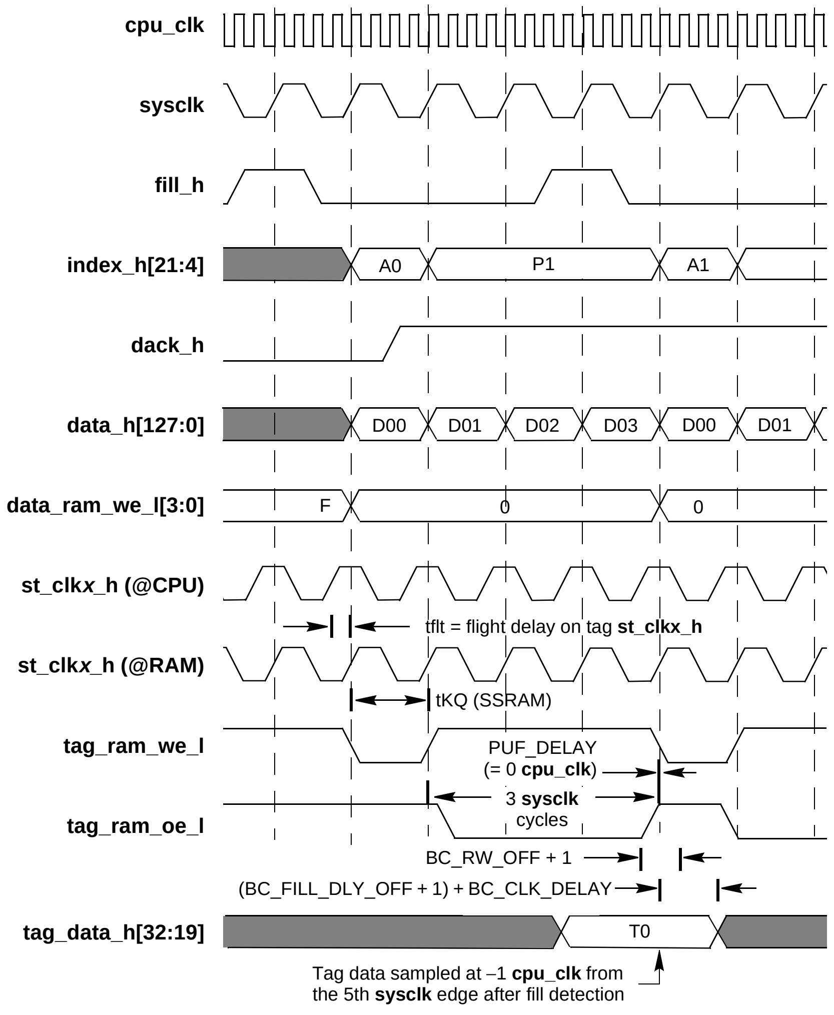
   The system designer must guarantee that the tag data roundtrip delay during the probe under fill is sampled properly at 21164PC. This is dependent upon the SSRAM type, either reg-reg (REG_REG = 1) or reg-flow style (REG_REG = 0) SSRAMs.

   Figure 4–25 is an example of this.

Figure 4–25 PUF_DELAY = 0 Using Reg-Reg SSRAMs

SYSCLK_RATIO = 4 **cpu_clk**
BC_CLK_DELAY = 1 **cpu_clk**
BC_FILL_DLY_OFF = 1 **cpu_clk**
tflt = 1 **cpu_clk**

tKQ(SSRAM) ≤ [3(4) – 0 – (4)×1 – (3) – 1] **cpu_clk**
tKQ(SSRAM) ≤ 4 **cpu_clk**



3. SYS_CLK_RATIO = 4, PUF_DELAY must be 1.

   This rule is imposed by an internal design restriction and must be enforced to ensure tag data integrity.

4.10 Data Bus and Command/Address Bus Contention

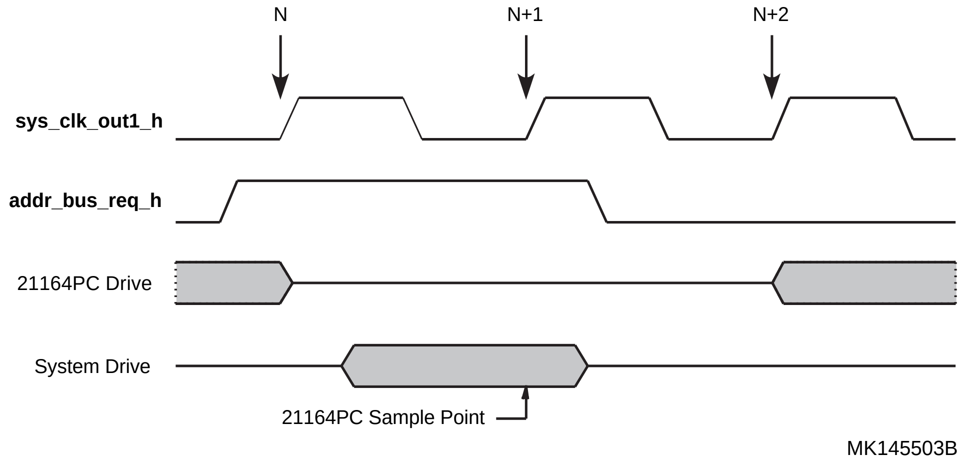
The data bus is composed of **data_h[127:0]** and **lw_parity_h[3:0]**. The command/address bus is composed of **cmd_h[3:0]** and **addr_h[39:4]**.

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

4.10.1 Command/Address Bus

Figure 4–26 shows the 21164PC and the system alternately driving the command/address bus. If signal **addr_bus_req_h** is asserted at the rising edge of **sysclk** N, the next cycle on the command/address bus belongs to the system. The 21164PC turns off its drivers at the rising edge of **sysclk** N. While the system must turn on its drivers between **sysclk** N and **sysclk** N+1, it must ensure that the drivers do not turn on before the 21164PC drivers turn off. The 21164PC samples the state of the command/address bus at the end of **sysclk** N+1. If **addr_bus_req_h** remains asserted, the system should continue to drive the command/address bus.

Figure 4–26 Driving the Command/Address Bus



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

4.10.2 Read/Write Spacing—Data Bus Contention

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

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

For private Bcache read operations followed by private Bcache write operations, the 21164PC inserts a programmable number of **cpu_clk** cycles between the read and the write operation (controlled by CBOX_CONFIG[BC_RW_OFF]). This allows time for the Bcache drivers to turn off before the 21164PC data drivers are turned on.

4.10.3 Using idle_bc_h and fill_h

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

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

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

Calculating Time to Assert idle_bc_h

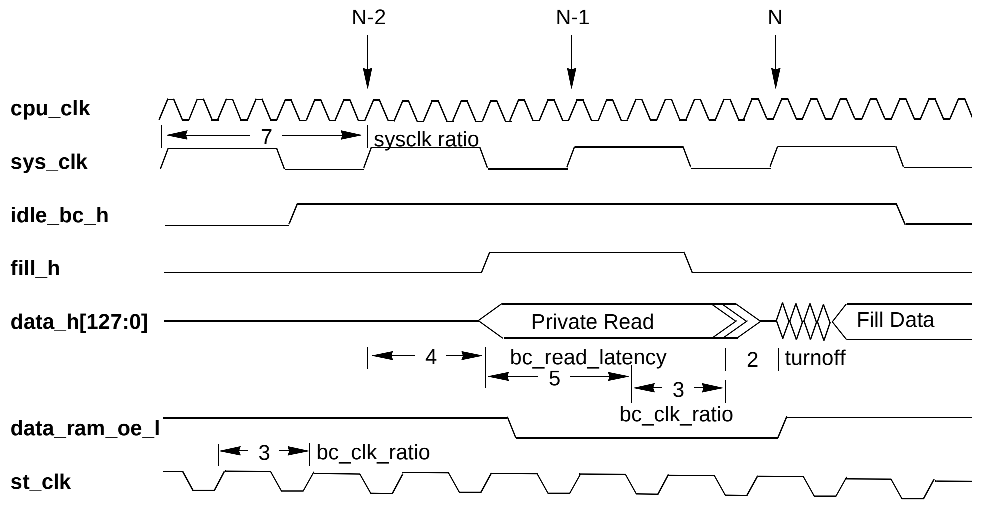
The idle_bc_time equation, for calculating the number of **sysclk** cycles that **idle_bc_h** must sample prior to fill data being driven, can be expressed as:

```
idle_cpu_cycles = (4 + BC_RD_LATENCY + BC_CLK_RATIO + tristate_ram_turn_off);
                  All values expressed as # of cpu cycles

idle_bc_time    = ROUNDUP(idle_cpu_cycles / sysclk_ratio);
                  All values expressed as # of sysclk cycles
```

When determining the tristate turnoff times, if the system will not turn on its drivers for some number of nanoseconds after the 21164PC starts driving Bcache **index_h[21:4]**, this time can be used to reduce the tristate_turn_off time.

For example, if the **sysclk** ratio is 7, the Bcache read latency is 5, the **bc_clk** ratio is 3, and two cycles are necessary for tristate turnoff, then the equations would work out to:



```
idle_cpu_cycles     = 4 + bc_rd_latency +  bc_clk_ratio +
                      tristate_ram_turn_off

                    = 4 + 5 + 3 + 2

                    = 14 CPU cycles

idle_bc_time(sysclk)= ROUNDUP(14/7)

                    = 2 sysclk cycles

This requires idle_bc_h to be sampled two sysclk cycles before the fill data
is driven.
```

4.10.4 Using data_bus_req_h

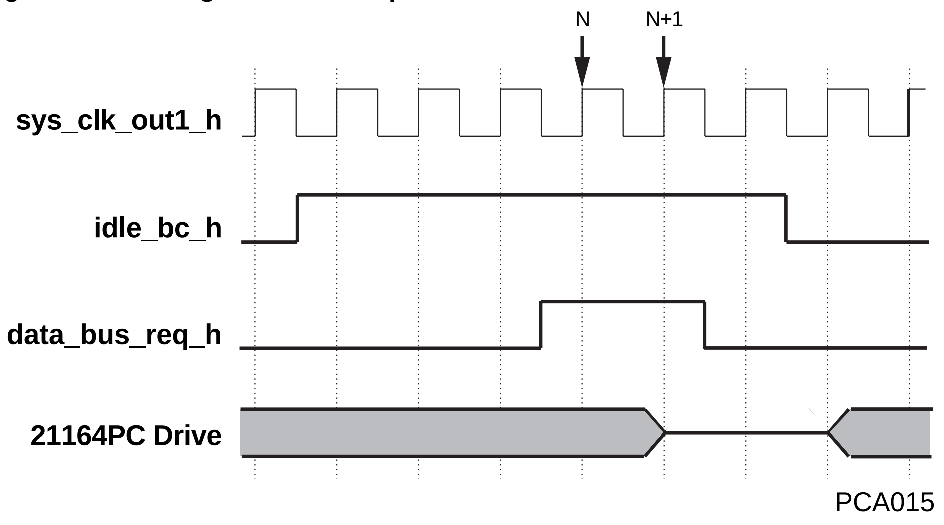
The signal **data_bus_req_h** can be used along with the **idle_bc_h** signal to prevent the 21164PC and the Bcache from driving the data bus. In general, the system should not need to use this feature but it may be useful if the system places other devices on the data bus. Figure 4–27 shows an example of this timing.

Data Bus and Command/Address Bus Contention

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

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

Figure 4–27 Using data_bus_req_h



4.10.5 Tristate Overlap

The **addr_h[39:4]**, **cmd_h[3:0]**, **data_h[127:0]**, and **tag_data_h[32:19]** buses must be operated in such a way that no more than one driver may drive the bus at a time. This section describes particular cases where tristate overlap may be a problem that needs to be corrected using features described in previous sections.

The “owner” of each bus must drive the bus to some value for each cycle. Tristate drivers in the 21164PC turn on and off very fast (in the 0.5-ns to 1.0-ns range). At the other end of the range, SSRAM memory devices turn on and off slowly (in the 4.0-ns to 7.0-ns range). Generally, system drivers fall somewhere in the middle.

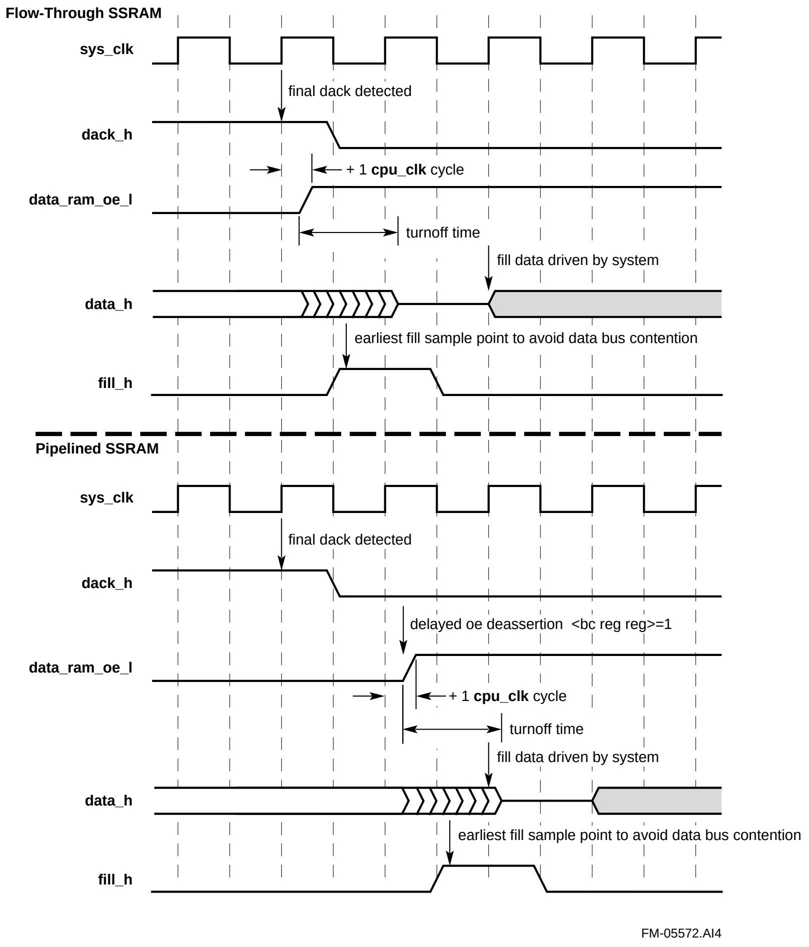
4.10.5.1 Private READ or WRITE to FILL

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

4.10.5.2 System READ to FILL (System WRITE) Spacing

The time to turn off the Bcache drivers at the end of a system READ (Bcache victim or system command with data movement) is fixed by the 21164PC design (refer to Figure 4–28). The CBOX_CONFIG[BC_REG_REG] bit is set when using pipelined SSRAMs, which delays the deassertion of **data_ram_oe_l** by one **sysclk** cycle after the detection of the final **dack_h**. When the bit is clear (for use with flow-through SSRAMs), **data_ram_oe_l** is deasserted one **cpu_clk** cycle after the detection of the final **dack_h**. The system must allow time for **data_ram_oe_l** to turn off and the RAMs to stop driving the bus, before the system drives fill data to avoid data bus contention.

Figure 4–28 System READ to FILL Spacing

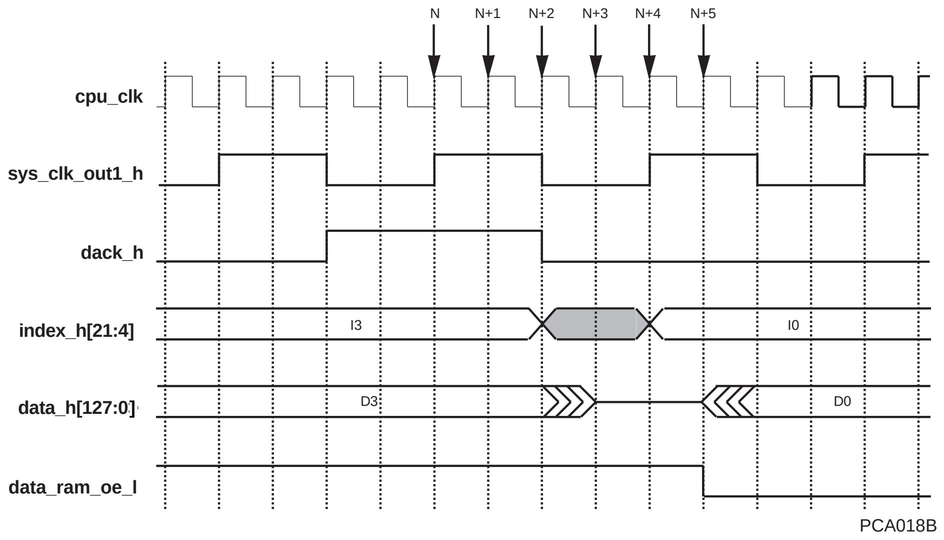


4.10.5.3 FILL to Private READ or WRITE Operation

At the end of the fill, the 21164PC does not begin to drive the data bus until the fifth **cpu_clk** cycle after the **sysclk** that loads the last **dack_h** (refer to Figure 4–29). The 21164PC does not assert **data_ram_oe_l** until the fifth cycle after the **sysclk** that loads the last **dack_h**.

Systems requiring more time to turn off their drivers must not send any more requests and must use **idle_bc_h** and **data_bus_req_h** at the end of the fill to provide adequate write-to-read spacing to avoid data bus contention.

Figure 4–29 FILL to Private READ or WRITE Operation



4.11 21164PC Interface Restrictions

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

4.11.1 Fill Operations After Other Transactions

For a system Bcache read operation (Bcache victim or a system-initiated data movement) followed by a fill operation, the earliest assertion of **fill_h** by the system is dependent upon the CBOX_CONFIG[BC_REG_REG] bit to avoid a read-to-write data bus contention. When the BC_REG_REG bit is set, the 21164PC will deassert **data_ram_oe_l** a full **sysclk** cycle after the final **dack_h** is detected to allow the system adequate time to sample the Bcache read data. See Section 4.10.5.2 for timing diagrams and assumptions that *must* be met by the system.

For a WRITE BLOCK operation followed by a fill operation, the earliest point the system can assert the **fill_h** signal is at the **sysclk** after the last assertion of **dack_h**.

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

4.11.2 Command Acknowledge for WRITE BLOCK Commands

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

4.12 21164PC/System Race Conditions

When certain sequences of transactions occur on the interface between the 21164PC, the Bcache, and the system, race conditions may occur. The rules for use of the interface by the 21164PC and the system are listed in Section 4.12.1.

Examples of race conditions to be avoided are described and illustrated in Section 4.12.2 through Section 4.12.6.

4.12.1 Rules for 21164PC and System Use of External Interface

This section lists the rules for determining the order in which 21164PC and system requests are allowed by the CBU BIU. In general, the order allowed is determined by use of **cmd_h[3:0]**, **idle_bc_h**, and **fill_h**.

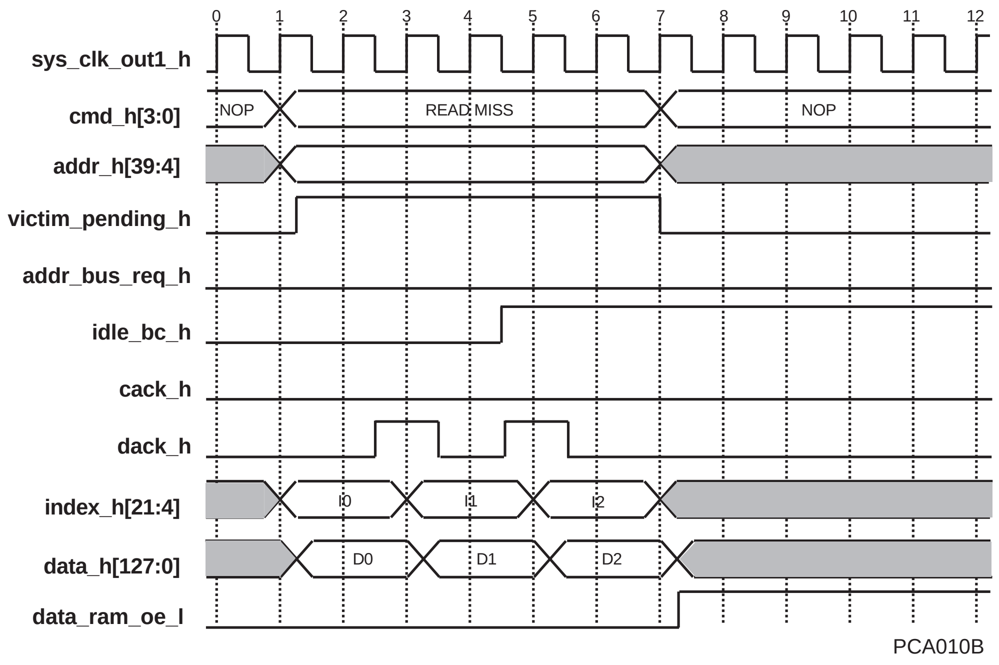
1. If **idle_bc_h** is not asserted and there are no valid requests in the BIU command buffer, then the BIU is free to perform any 21164PC request.
2. If a fill transaction is pending, the BIU only produces another READ MISS command, with a possible BCACHE VICTIM command. The BIU will not attempt any other command.
3. The assertion of **idle_bc_h**, or the sending of a system command other than NOP to the 21164PC, causes the BIU to idle. If the BIU has a command loaded in the pad ring, it removes the command and replaces it with a NOP command. The state of **cmd_h[3:0]** is unpredictable until the idle condition ends.
4. The idle condition ends when the 21164PC receives a deasserted **idle_bc_h**, and the 21164PC has responded to all the system commands that were sent.
5. The system must not assert **cack_h** during the idle condition.

6. There is one exception to rules 3, 4, and 5. If **idle_bc_h** or a system command arrives while the 21164PC is reading the Bcache, and that read transaction turns into a read miss transaction, and it does not produce a victim, then the 21164PC loads the miss into the pad ring. The system may assert **cack_h** for this read miss request at any time.
7. If **cack_h** is asserted at the same time as **idle_bc_h** or a valid system request, **cack_h** wins and the command is taken by the system. Signal **cack_h** should not be asserted if **idle_bc_h** has been asserted or a valid system command is under way.
8. A read miss with a Bcache victim transaction is treated as an atomic pair. If the READ MISS command is acknowledged with **cack_h**, then the BCACHE VICTIM command must be acknowledged with **cack_h** and all the data acknowledged with **dack_h**, before the 21164PC responds to any other request. The system must also guarantee that once the read miss operation has been cacked, system commands or fill transactions are not started until the read miss/Bcache victim pair have completed.
9. The **cack_h** acknowledgment for a write block or Bcache victim transaction must be received by the 21164PC with or before the last **dack_h** acknowledgment of the data. For write block and Bcache victim transactions, it is possible to acknowledge all but the last data, and then decide to do something else.
10. For a read miss transaction, **cack_h** must be received with or before the last data acknowledgment (**dack_h**) for the requested fill operation.

4.12.2 READ MISS with Victim Aborted by FILL Example

In Figure 4–30, the 21164PC asserts a READ MISS command with a victim. The system asserts **dack_h** for two data cycles received from the Bcache and then asserts **idle_bc_h**. This causes the 21164PC to remove the READ MISS command with victim pending. The 21164PC reasserts the READ MISS and BCACHE VICTIM commands, if needed, at a later time.

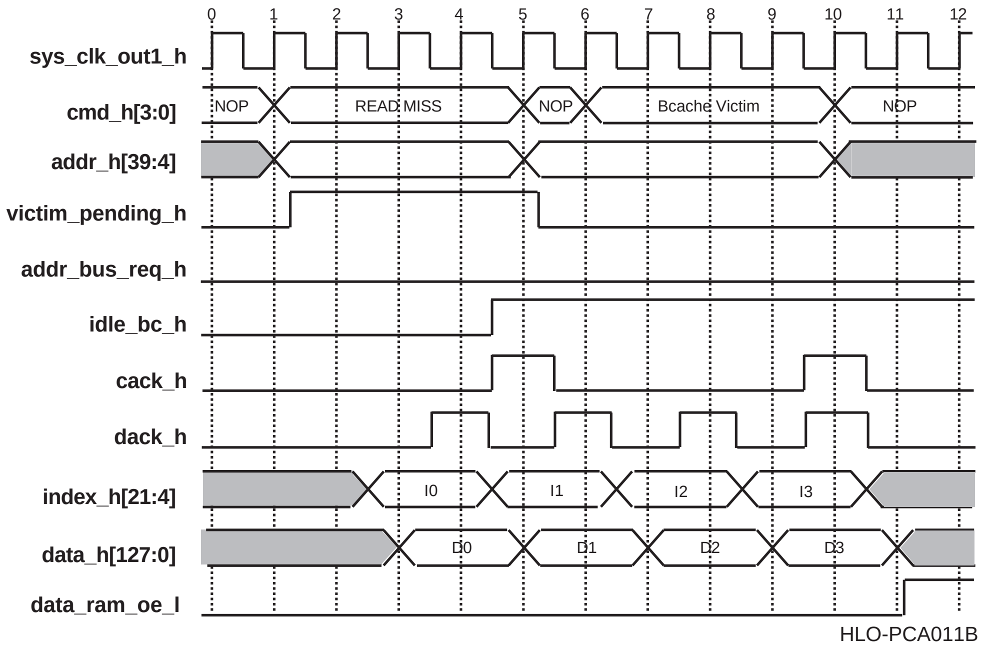
Figure 4–30 READ MISS with Victim Aborted by FILL Example



4.12.3 idle_bc_h and cack_h Race Example

In Figure 4–31, **idle_bc_h** and **cack_h** are asserted in the same **sysclk** cycle. The system takes the READ MISS and BCACHE VICTIM commands before doing anything else. The last **dack_h** meets the requirement that the **cack_h** arrive before or with the last **dack_h**.

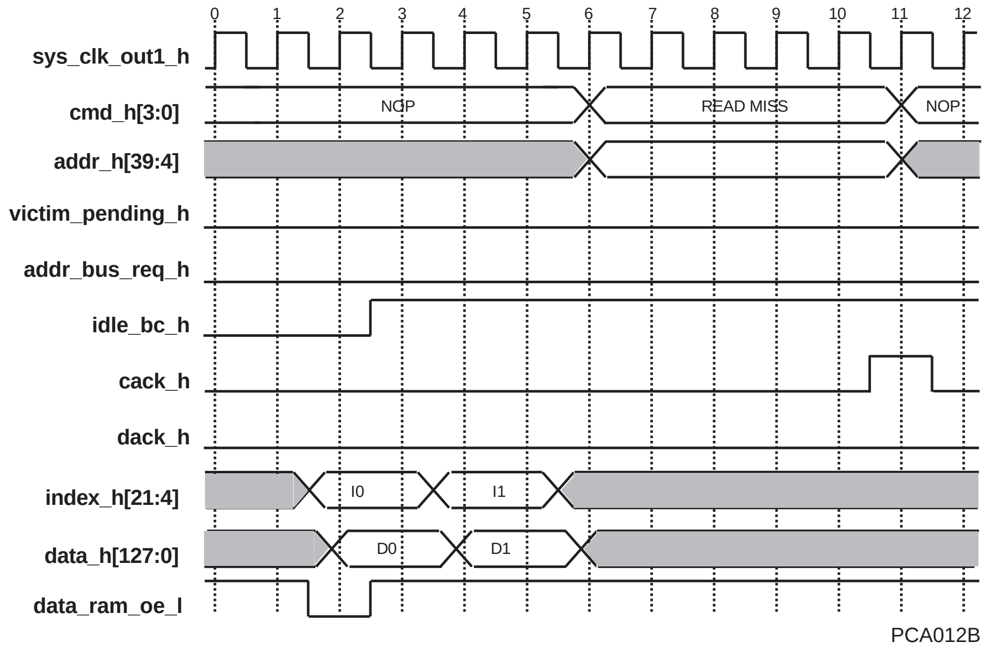
Figure 4–31 idle_bc_h and cack_h Race Examples



4.12.4 READ MISS with idle_bc_h Asserted Example

In Figure 4–32, the 21164PC has started a Bcache read operation that misses. The signal **idle_bc_h** is asserted, but no victim was created, so the read miss request is loaded into the pad ring. The system then takes the request.

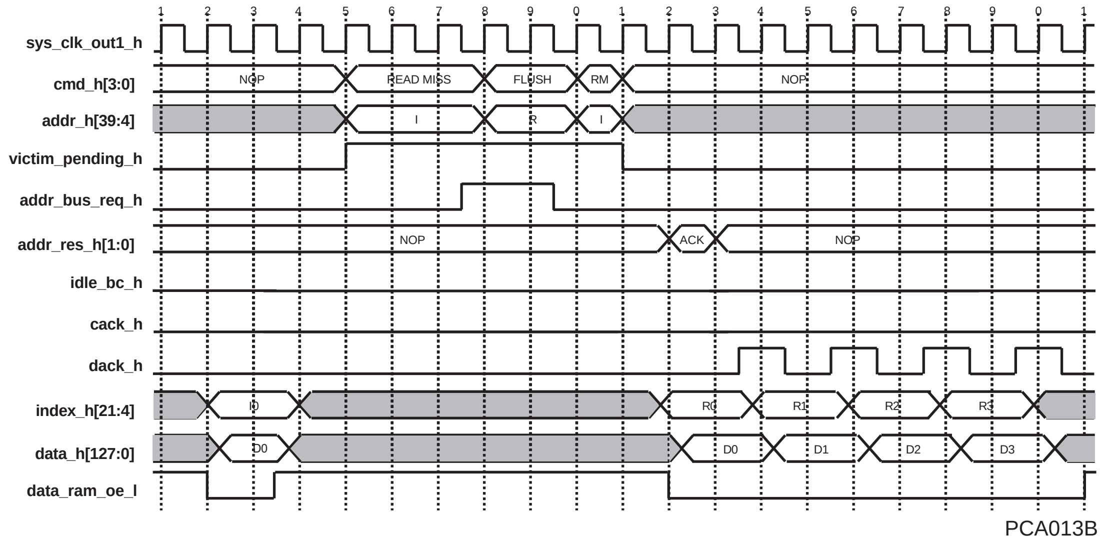
Figure 4–32 READ MISS with idle_bc_h Asserted Example



4.12.5 READ MISS with Victim Aborted by System Command Example

In Figure 4–33, the 21164PC produces a READ MISS command with a victim and is waiting for the system to take it when the system takes the bus and requests a flush transaction. The 21164PC drives the read miss request for one more cycle after it gets command of the bus and then removes the request. The 21164PC then responds to the FLUSH command and drives **index_h[21:4]** to read the Bcache. The 21164PC restarting the Bcache read operation, requesting the read miss with victim, is not shown in the timing diagram. If the victim block was invalidated by the system request, the 21164PC produces a clean read miss transaction.

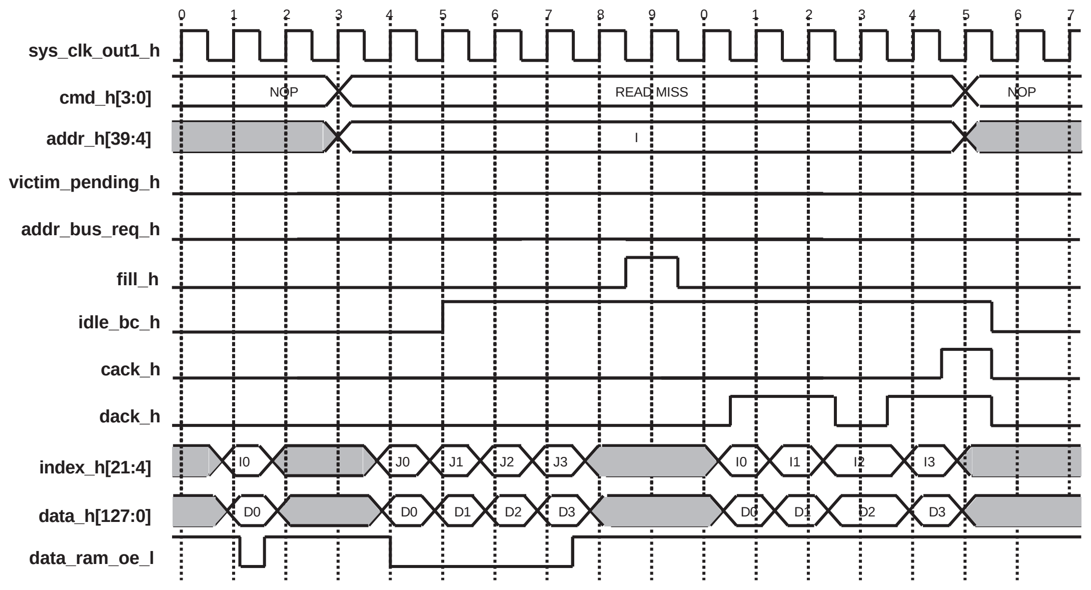
Figure 4–33 READ MISS with Victim Abort Example



4.12.6 Bcache Hit Under READ MISS Example

In Figure 4–34, the 21164PC produces a read miss transaction and requests a fill from the system. A Bcache hit to index j takes place while waiting for the fill. The system then returns the requested data in two bursts, asserting **cack_h** at the same time as the last assertion of **dack_h**.

Figure 4–34 Bcache Hit Under READ MISS Example



PCA014B

4.13 Data Integrity and Bcache Errors

Mechanisms for ensuring that errors on data received by the 21164PC from the Bcache, the system, or both are described in this section. Tag data errors are also described.

4.13.1 Data Parity

The 21164PC supports INT4 parity protection on the data bus for the external Bcache and memory system. When the 21164PC drives data to memory, it generates longword parity and places it on **lw_parity_h[3:0]** for write operations. Parity is checked for read operations. Parity for **data_h[31:0]** is driven on signal **lw_parity_h[0]** and so on.

4.13.2 Bcache Tag Data Parity

The signal line **tag_data_par_h** is used to maintain parity over **tag_data_h[32:19]**, **tag_valid_h**, and **tag_dirty_h**. A Bcache tag data parity error is usually not recoverable.

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

4.13.3 Fill Error

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

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

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

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

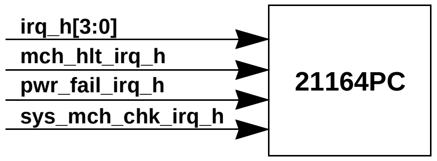
Asserting **fill_error_h** forces a trap to the PALcode at the MCHK entry point but has no other effect.

4.14 Interrupts

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

Figure 4–35 shows the 21164PC interrupt signals.

Figure 4–35 21164PC Interrupt Signals



4.14.1 Interrupt Signals During Initialization

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

4.14.2 Interrupt Signals During Normal Operation

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

4.14.3 Interrupt Priority Level

Table 4–11 shows which interrupts are enabled for a given interrupt priority level (IPL). An interrupt is enabled if the current IPL is less than the target IPL of the interrupt.

Table 4–11 Interrupt Priority Level Effect *(Sheet 1 of 2)*

Interrupt Source	Target IPL	Source
Software Interrupt Request 1	1	Internal
Software Interrupt Request 2	2	Internal
Software Interrupt Request 3	3	Internal
Software Interrupt Request 4	4	Internal
Software Interrupt Request 5	5	Internal
Software Interrupt Request 6	6	Internal
Software Interrupt Request 7	7	Internal
Software Interrupt Request 8	8	Internal
Software Interrupt Request 9	9	Internal
Software Interrupt Request 10	10	Internal
Software Interrupt Request 11	11	Internal
Software Interrupt Request 12	12	Internal
Software Interrupt Request 13	13	Internal
Software Interrupt Request 14	14	Internal

Table 4–11 Interrupt Priority Level Effect *(Sheet 2 of 2)*

Interrupt Source	Target IPL	Source
Software Interrupt Request 15	15	Internal
Asynchronous system trap ATR pending (for current or more privileged mode)	2	Internal
Performance counter interrupt	29	Internal
Powerfail interrupt[1]	30	**pwr_fail_irq_h**
System machine check interrupt[1]	31	**sys_mch_chk_irq_h** and internal
External interrupt 20[1]	20[2]	**irq_h[0]**
External interrupt 21[1]	21[2]	**irq_h[1]**
External interrupt 22[1]	22[2]	**irq_h[2]**
External interrupt 23[1]	23[2]	**irq_h[3]**
Halt[1]	Masked only by executing in PALmode.	**mch_hlt_irq_h**
Serial line interrupt	Masked only by executing in PALmode.	Internal

[1] These interrupts are from external sources. In some cases, the system environment provides the logic-OR of multiple interrupt sources at the same IPL to a particular pin.
[2] The external interrupts 20-23 are separately maskable by setting the appropriate bits in the ICSR register.

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

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

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

5 Internal Processor Registers

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

- Instruction fetch/decode unit and branch unit (IDU) IPRs
- Memory address translation unit (MTU) IPRs
- Cache control and bus interface unit (CBU) IPRs
- PAL storage registers
- Restrictions

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

CBU and backup cache (Bcache) IPRs are accessible in the physical address region FF FFF0 0000 to FF FFFF FFFF. Table 5–25 summarizes the CBU and Bcache IPRs. Table 5–31 lists restrictions on the IPRs.

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

Table 5–1 IDU, MTU, Dcache, and PALtemp IPR Encodings *(Sheet 1 of 4)*

IPR Mnemonic	Access	Index$_{16}$	IDU Slots to Pipe
IDU_IPRs			
ISR	R	100	E1
ITB_TAG	W	101	E1
ITB_PTE	R/W	102	E1
ITB_ASN	R/W	103	E1

Table 5–1 IDU, MTU, Dcache, and PALtemp IPR Encodings *(Sheet 2 of 4)*

IPR Mnemonic	Access	Index_{16}	IDU Slots to Pipe
ITB_PTE_TEMP	R	104	E1
ITB_IA	W	105	E1
ITB_IAP	W	106	E1
ITB_IS	W	107	E1
SIRR	R/W	108	E1
ASTRR	R/W	109	E1
ASTER	R/W	10A	E1
EXC_ADDR	R/W	10B	E1
EXC_SUM	R/W0C	10C	E1
EXC_MASK	R	10D	E1
PAL_BASE	R/W	10E	E1
ICM	R/W	10F	E1
IPLR	R/W	110	E1
INTID	R	111	E1
IFAULT_VA_FORM	R	112	E1
IVPTBR	R/W	113	E1
HWINT_CLR	W	115	E1
SL_XMIT	W	116	E1
SL_RCV	R	117	E1
ICSR	R/W	118	E1
IC_FLUSH_CTL	W	119	E1
ICPERR_STAT	R/W1C	11A	E1
PMCTR	R/W	11C	E1
PALtemp_IPRs			
PALtemp0	R/W	140	E1
PALtemp1	R/W	141	E1

Table 5–1 IDU, MTU, Dcache, and PALtemp IPR Encodings *(Sheet 3 of 4)*

IPR Mnemonic	Access	Index$_{16}$	IDU Slots to Pipe
PALtemp2	R/W	142	E1
PALtemp3	R/W	143	E1
PALtemp4	R/W	144	E1
PALtemp5	R/W	145	E1
PALtemp6	R/W	146	E1
PALtemp7	R/W	147	E1
PALtemp8	R/W	148	E1
PALtemp9	R/W	149	E1
PALtemp10	R/W	14A	E1
PALtemp11	R/W	14B	E1
PALtemp12	R/W	14C	E1
PALtemp13	R/W	14D	E1
PALtemp14	R/W	14E	E1
PALtemp15	R/W	14F	E1
PALtemp16	R/W	150	E1
PALtemp17	R/W	151	E1
PALtemp18	R/W	152	E1
PALtemp19	R/W	153	E1
PALtemp20	R/W	154	E1
PALtemp21	R/W	155	E1
PALtemp22	R/W	156	E1
PALtemp23	R/W	157	E1
MTU_IPRs			
DTB_ASN	W	200	E0
DTB_CM	W	201	E0
DTB_TAG	W	202	E0

Table 5–1 IDU, MTU, Dcache, and PALtemp IPR Encodings *(Sheet 4 of 4)*

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

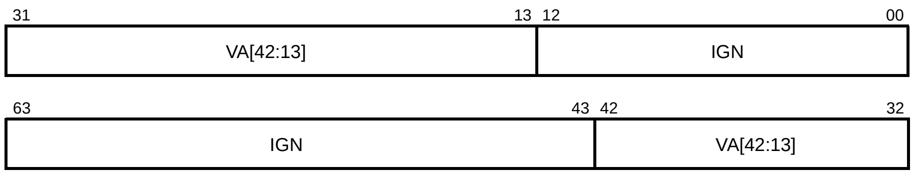
5.1 Instruction Fetch/Decode Unit and Branch Unit (IDU) IPRs

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

5.1.1 Istream Translation Buffer Tag (ITB_TAG) Register (101)

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

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



5.1.2 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register (102)

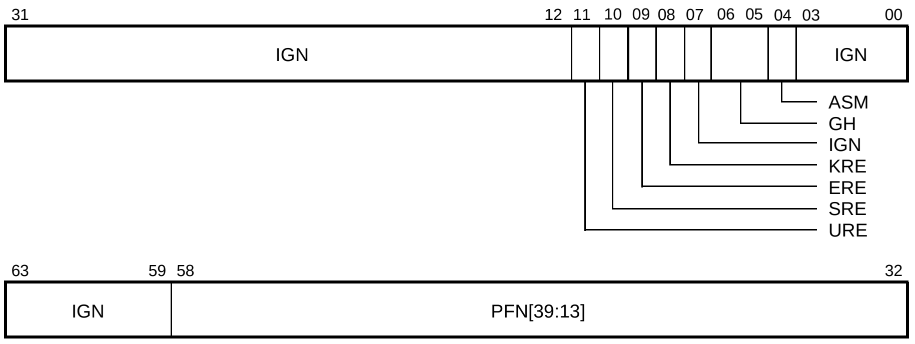
ITB_PTE is a read/write register.

Write Format

A write operation to this register writes both the PTE and TAG fields of an ITB location determined by a not-last-used replacement algorithm. The TAG and PTE fields are updated simultaneously to ensure the integrity of the ITB. A write operation to the ITB_PTE register increments the not-last-used (NLU) pointer, which allows for writing the entire set of ITB PTE and TAG entries. If the HW_MTPR ITB_PTE instruction falls in the shadow of a trapping instruction, the NLU pointer may be incremented multiple times. The TAG field of the ITB location is determined by the contents of the ITB_TAG register. The PTE field is provided by the HW_ MTPR

ITB_PTE instruction. Write operations to this register use the memory format bits, as described in the *Alpha Architecture Reference Manual*. Figure 5–2 shows the ITB_PTE register write format.

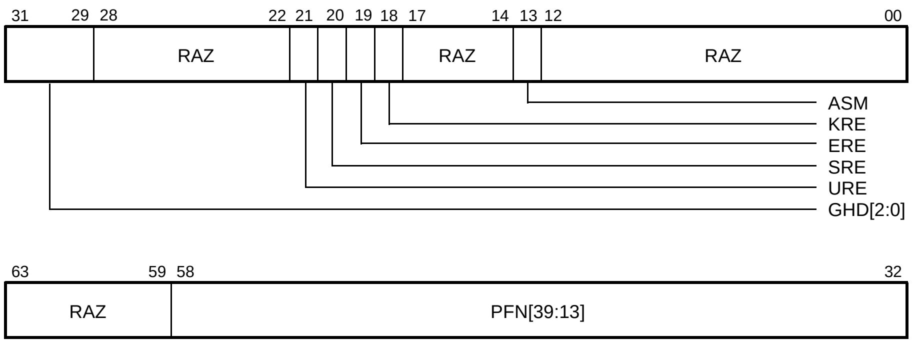
Figure 5–2 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Write Format



Read Format

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

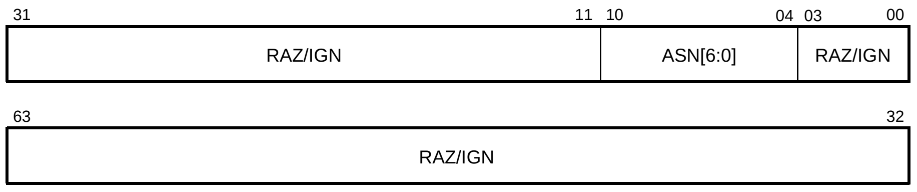
Figure 5–3 Instruction Translation Buffer Page Table Entry (ITB_PTE) Register Read Format



5.1.3 Instruction Translation Buffer Address Space Number (ITB_ASN) Register (103)

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

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



5.1.4 Instruction Translation Buffer Page Table Entry Temporary (ITB_PTE_TEMP) Register (104)

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

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

Table 5–2 Granularity Hint Bits in ITB_PTE_TEMP Read Format

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

5.1.5 Instruction Translation Buffer Invalidate All Process (ITB_IAP) Register (106)

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

5.1.6 Instruction Translation Buffer Invalidate All (ITB_IA) Register (105)

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

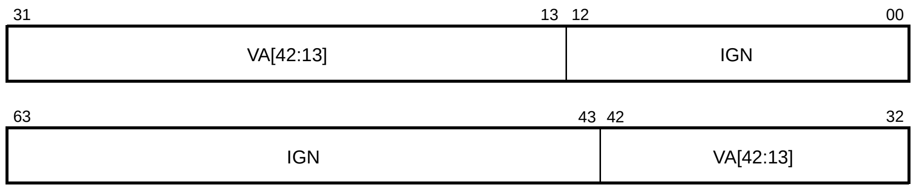
5.1.7 Instruction Translation Buffer IS (ITB_IS) Register (107)

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

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

Figure 5–5 shows the ITB_IS register format.

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



5.1.8 Formatted Faulting Virtual Address (IFAULT_VA_FORM) Register (112)

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

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

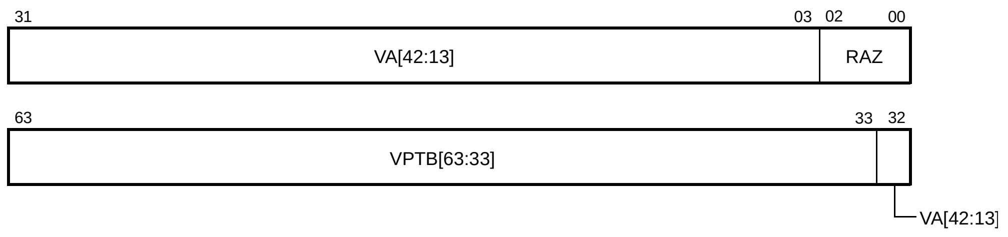
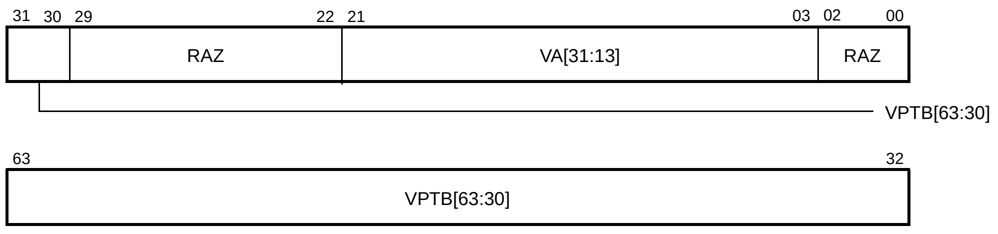


Figure 5–7 shows the IFAULT_VA_FORM register format in NT mode.

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



5.1.9 Virtual Page Table Base (IVPTBR) Register (113)

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

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

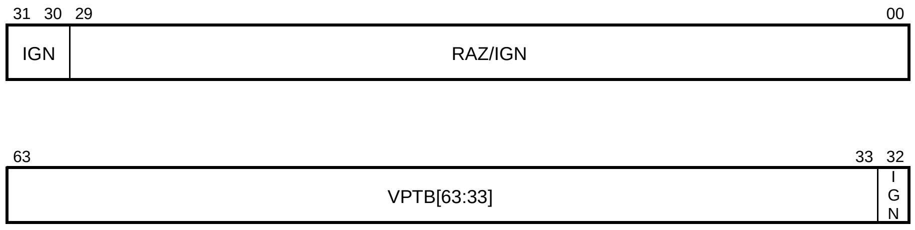
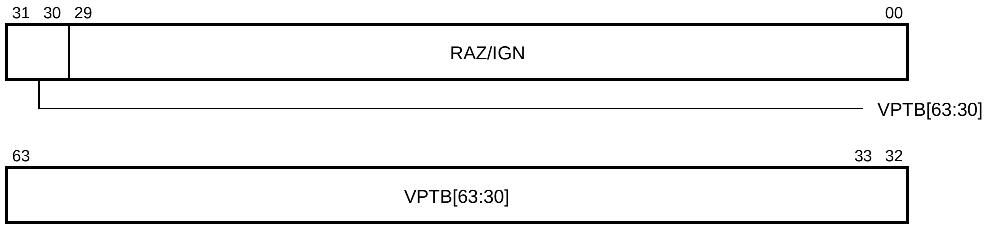


Figure 5–9 shows the IVPTBR register format in NT mode.

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



5.1.10 Icache Parity Error Status (ICPERR_STAT) Register (11A)

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

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

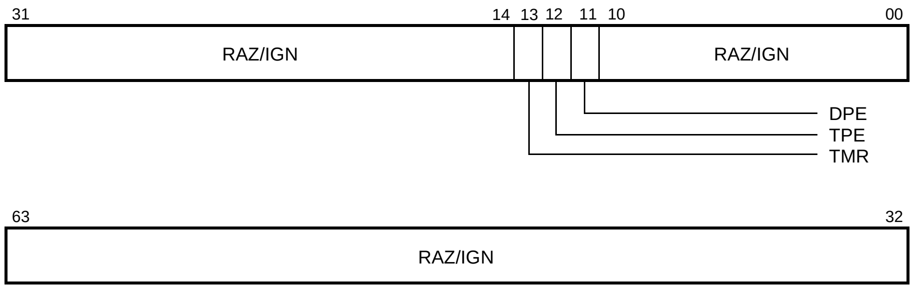


Table 5–3 Icache Parity Error Status Register Fields

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

5.1.11 Icache Flush Control (IC_FLUSH_CTL) Register (119)

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

5.1.12 Exception Address (EXC_ADDR) Register (10B)

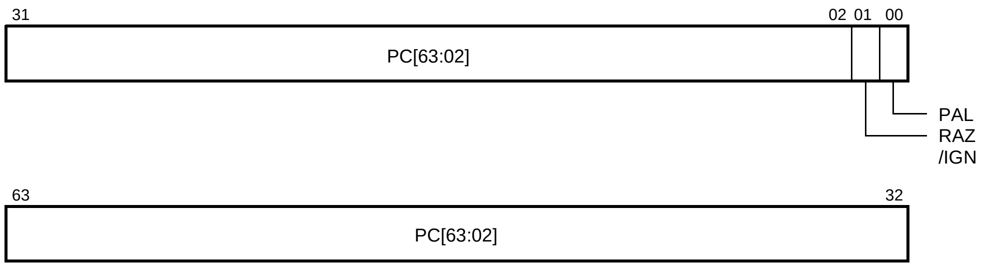
EXC_ADDR is a read/write register used to restart the system after exceptions or interrupts. The HW_REI instruction causes a return to the instruction pointed to by the EXC_ADDR register. This register can be written both by hardware and software. Hardware write operations occur as a result of exceptions/interrupts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

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

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

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

Figure 5–11 Exception Address (EXC_ADDR) Register



5.1.13 Exception Summary (EXC_SUM) Register (10C)

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

Figure 5–12 Exception Summary (EXC_SUM) Register

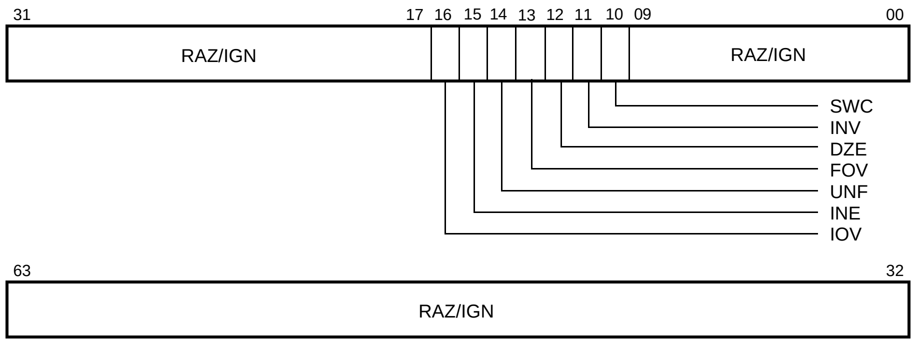


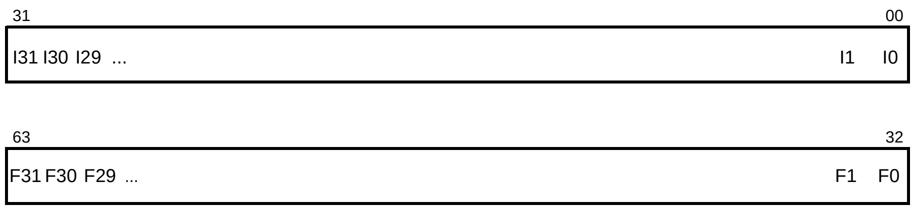
Table 5–4 Exception Summary Register Fields

Name	Extent	Type	Description
SWC	[10]	WA	Indicates software completion possible. This bit is set after a floating-point instruction containing the /S modifier completes with an arithmetic trap, and if all previous floating-point instructions that trapped since the last HW_MTPR EXC_SUM instruction also contained the /S modifier. The SWC bit is cleared whenever a floating-point instruction without the /S modifier completes with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps until the register is written by an HW_ MTPR instruction. The bit is always cleared upon any HW_MTPR write operation to the EXC_SUM register.
INV	[11]	WA	Indicates invalid operation.
DZE	[12]	WA	Indicates divide by zero.
FOV	[13]	WA	Indicates floating-point overflow.
UNF	[14]	WA	Indicates floating-point underflow.
INE	[15]	WA	Indicates floating inexact error.
IOV	[16]	WA	Indicates floating-point execution unit (FPU) convert to integer overflow or integer arithmetic overflow.

5.1.14 Exception Mask (EXC_MASK) Register (10D)

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

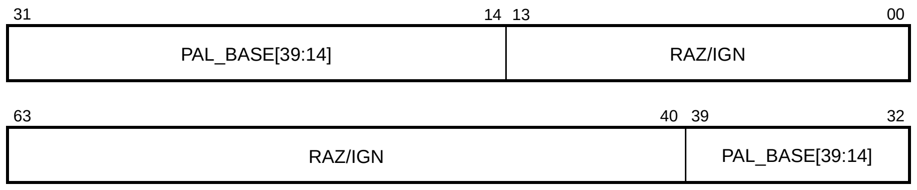
Figure 5–13 Exception Mask (EXC_MASK) Register



5.1.15 PAL Base Address (PAL_BASE) Register (10E)

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

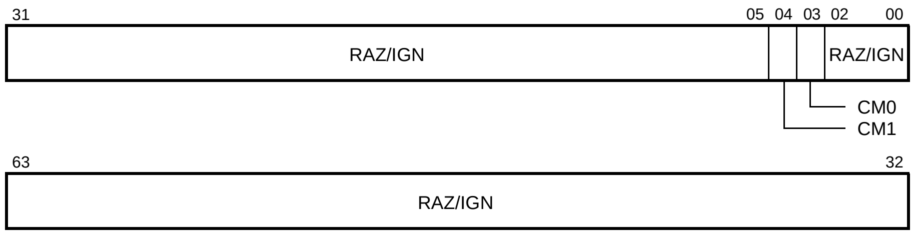
Figure 5–14 PAL Base Address (PAL_BASE) Register



5.1.16 IDU Current Mode (ICM) Register (10F)

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

Figure 5–15 IDU Current Mode (ICM) Register



5.1.17 IDU Control and Status (ICSR) Register (118)

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

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

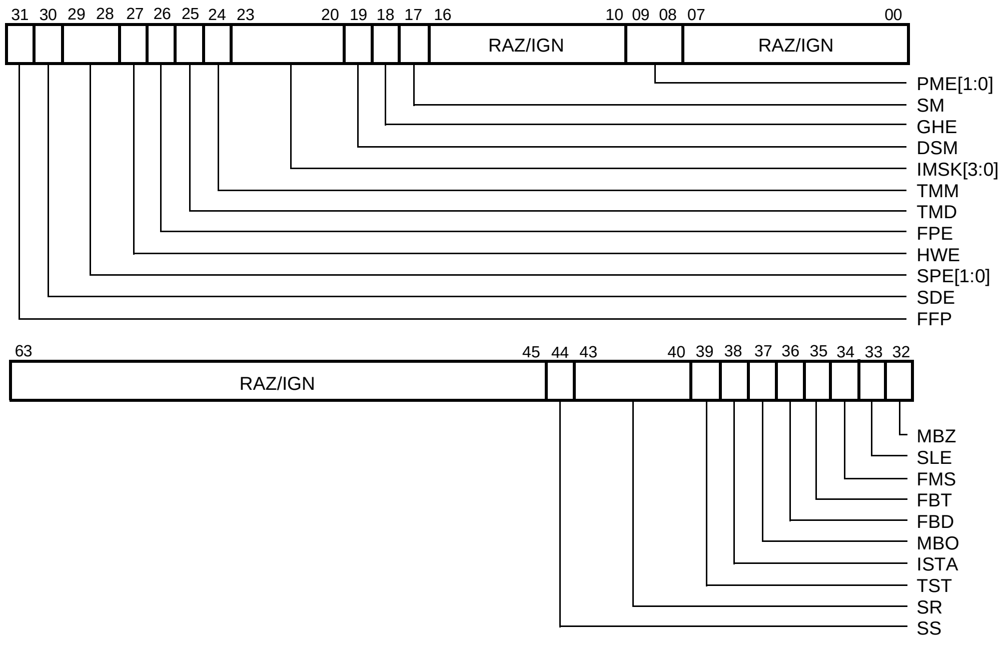


Table 5–5 IDU Control and Status Register Fields *(Sheet 1 of 3)*

Name	Extent	Type	Description
PME[1:0]	[09:08]	RW,0	Performance counter master enable bits. If both PME[1] and PME[0] are clear, all performance counters in the PMCTR IPR are disabled. If either PME[1] or PME[0] are set, the counter is enabled according to the settings of the PMCTR CTL fields.
SM	[17]	RW,0	Sleep mode bit. Setting this bit initiates the transition into C3 state, sequencing the clock frequency to 1/8. If DSM bit is also set, the frequency is set to 1/16.
GHE	[18]	RW,0	Global history enable. Setting this bit enables the use of global history during branch prediction.

Table 5–5 IDU Control and Status Register Fields *(Sheet 2 of 3)*

Name	Extent	Type	Description
DSM	[19]	RW,0	Deep sleep mode bit. Setting this bit along with the SM bit causes the clock frequency to sequence down to 1/16. If set without the SM bit, it causes undefined behavior.
IMSK[3:0]	[23:20]	RW,0	If set, each IMSK[3:0] signal disables the corresponding IRQ_ H[3:0] interrupt.
TMM	[24]	RW,0	If set, the timeout counter counts 5000 cycles before asserting timeout reset. If clear, the timeout counter counts 1 billion cycles before asserting timeout reset.
TMD	[25]	RW,0	If set, disables the IDU timeout counter. Does not affect **cfail_h**/no **cack_h** error.
FPE	[26]	RW,0	If set, floating-point instructions may be issued. If clear, floating-point instructions cause FEN exceptions.
HWE	[27]	RW,0	If set, allows PALRES instructions to be issued in kernel mode.
SPE[1:0]	[29:28]	RW,0	If SPE[1] is set, it enables superpage mapping of Istream virtual address VA[39:13] directly to physical address PA[39:13] assuming VA[42:41] = 10. Virtual address bit VA[40] is ignored in this translation. Access is allowed only in kernel mode. If SPE[0] is set (NT mode), it enables superpage mapping of Istream virtual addresses VA[42:30] = $1FFE_{16}$ directly to physical address PA[39:30] = 0_{16}. VA[30:13] is mapped directly to PA[30:13]. Access is allowed only in kernel mode.
SDE	[30]	RW,0	If set, enables PAL shadow registers.
FFP	[31]	RW,1	If set, forces all Icache fills to go to set 0. MBO until ITB_ASN is written with a valid value.
MBZ	[32]	RW,0	Reserved to COMPAQ. Must be zero.
SLE	[33]	RW,0	If set, enables serial line interrupts.

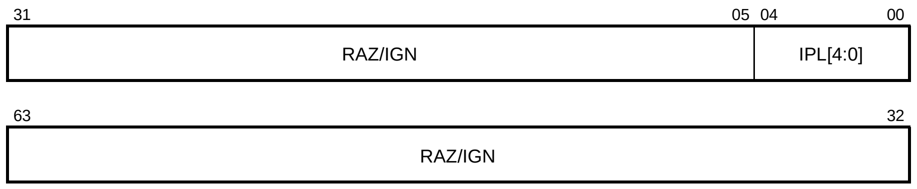
Table 5–5 IDU Control and Status Register Fields *(Sheet 3 of 3)*

Name	Extent	Type	Description
FMS	[34]	RW,0	If set, forces miss on Icache references. MBZ in normal operation.
FBT	[35]	RW,0	If set, forces bad Icache tag parity. MBZ in normal operation.
FBD	[36]	RW,0	If set, forces bad Icache data parity. MBZ in normal operation.
MBO	[37]	RW,1	Reserved to COMPAQ. Must be one.
ISTA	[38]	RO	Reading this bit indicates ICACHE BIST status. If set, ICACHE BIST was successful.
TST	[39]	RW,0	Writing a 1 to this bit asserts the **test_status_h[1]** signal.
SR	[43:40]	RW,0	Sysclk ratio, programmed during system reset. Refer to Chapter 10 for details.
SS	[44]	RO	If set, indicates the 21164PC is in C3-state, or is transitioning to C3-state. If clear, the 21164PC internal clock is running at full clock rate.

5.1.18 Interrupt Priority Level (IPLR) Register (110)

IPLR is a read/write register that is accessed by PALcode to set the value of the interrupt priority level (IPL). Whenever hardware detects an interrupt whose target IPL is greater than the value in IPLR[04:00], an interrupt is taken. Figure 5–17 shows the IPLR register format. Refer to Table 4–11 for a description of which interrupts are enabled for a given IPL.

Figure 5–17 Interrupt Priority Level (IPLR) Register



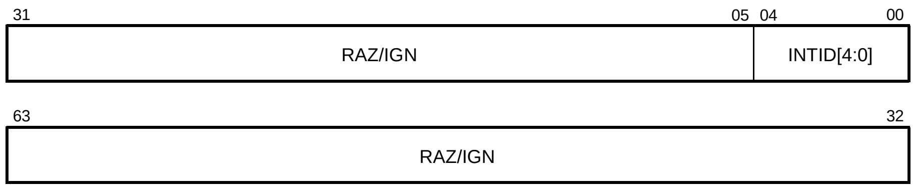
5.1.19 Interrupt ID (INTID) Register (111)

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

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

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

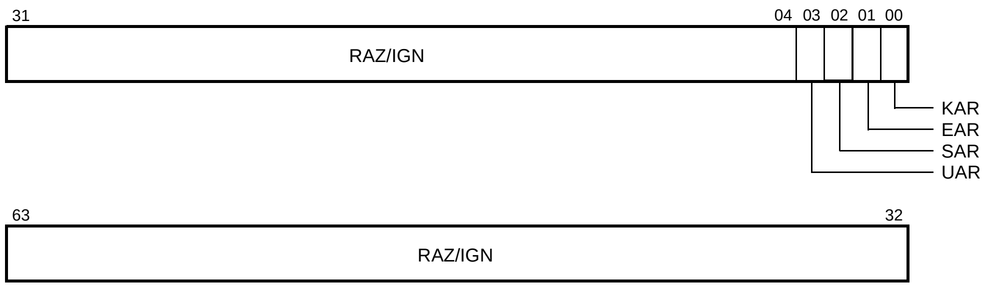
Figure 5–18 Interrupt ID (INTID) Register



5.1.20 Asynchronous System Trap Request (ASTRR) Register (109)

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

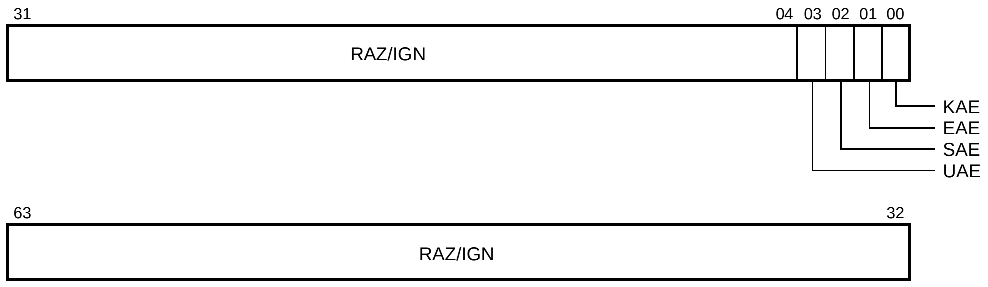
Figure 5–19 Asynchronous System Trap Request (ASTRR) Register



5.1.21 Asynchronous System Trap Enable (ASTER) Register (10A)

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

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



5.1.22 Software Interrupt Request (SIRR) Register (108)

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

Figure 5–21 Software Interrupt Request (SIRR) Register

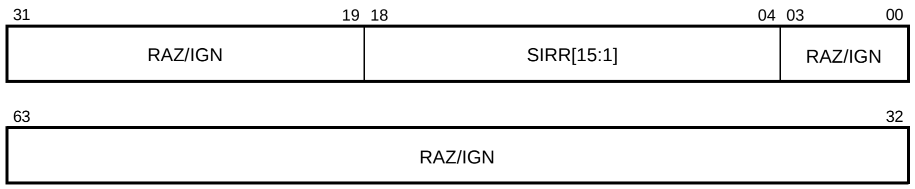


Table 5–6 Software Interrupt Request Register Fields

Name	Extent	Type	Description
SIRR[15:1]	[18:04]	RW	Request software interrupts.

5.1.23 Hardware Interrupt Clear (HWINT_CLR) Register (115)

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

Figure 5–22 Hardware Interrupt Clear (HWINT_CLR) Register

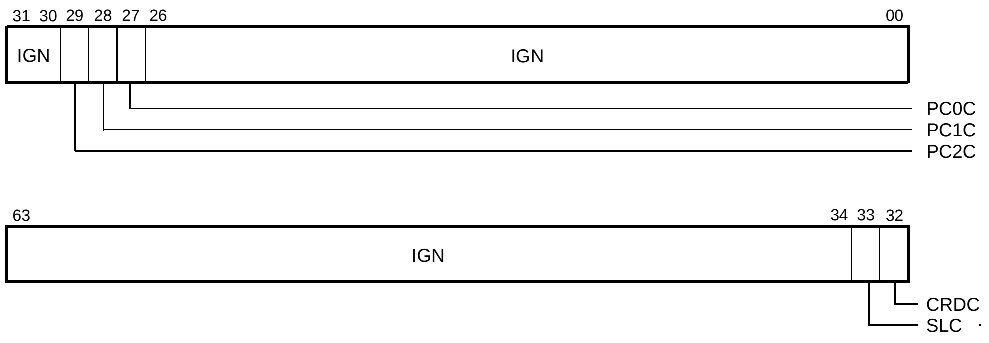


Table 5–7 Hardware Interrupt Clear Register Fields

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

5.1.24 Interrupt Summary (ISR) Register (100)

ISR is a read-only register containing information about all pending hardware, software, and asynchronous system trap (AST) interrupt requests. Figure 5–23 and Table 5–8 describe the ISR register format. Refer to Table 4–11 for a description of which interrupts are enabled for a given interrupt priority level (IPL).

Figure 5–23 Interrupt Summary (ISR) Register

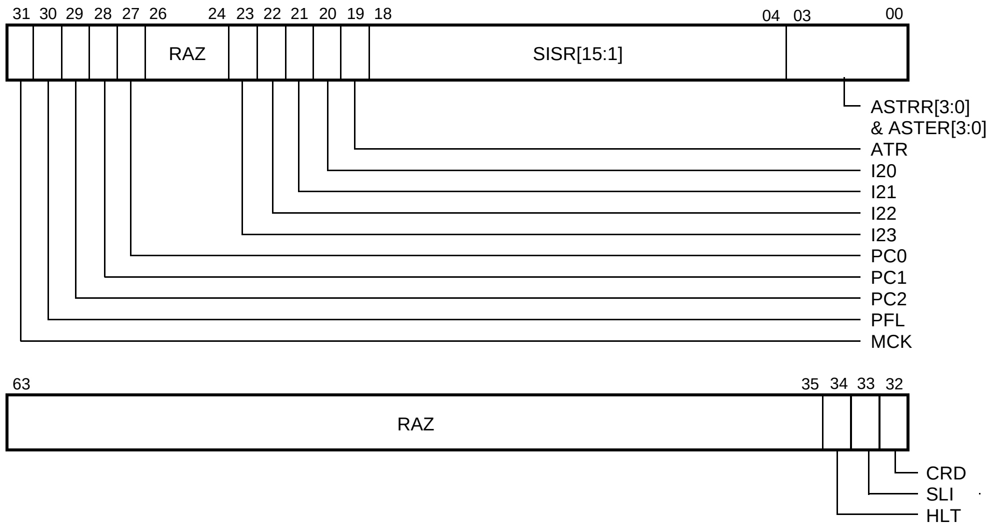


Table 5–8 Interrupt Summary Register Fields *(Sheet 1 of 2)*

Name	Extent	Type	Description
ASTRR[3:0] and ASTER[3:0]	[03:00]	RO	Boolean AND of ASTRR[USEK] with ASTER[USEK] used to indicate enabled AST requests.
SISR[15:1]	[18:04]	RO,0	Software interrupt requests 15 through 1 corresponding to IPL 15 through 1.
ATR	[19]	RO	Set if any AST request and corresponding enable bit is set and if the processor mode is equal to or higher than the AST request mode.
I20	[20]	RO	External hardware interrupt—**irq_h[0]**.
I21	[21]	RO	External hardware interrupt—**irq_h[1]**.
I22	[22]	RO	External hardware interrupt—**irq_h[2]**.

Table 5–8 Interrupt Summary Register Fields *(Sheet 2 of 2)*

Name	Extent	Type	Description
I23	[23]	RO	External hardware interrupt—**irq_h[3]**.
PC0	[27]	RO	External hardware interrupt—performance counter 0 (IPL 29).
PC1	[28]	RO	External hardware interrupt—performance counter 1 (IPL 29).
PC2	[29]	RO	External hardware interrupt—performance counter 2 (IPL 29).
PFL	[30]	RO	External hardware interrupt—power failure (IPL 30).
MCK	[31]	RO	External hardware interrupt—system machine check (IPL 31).
CRD	[32]	RO	Correctable ECC errors (IPL 31).
SLI	[33]	RO	Serial line interrupt.
HLT	[34]	RO	External hardware interrupt—halt.

5.1.25 Serial Line Transmit (SL_XMIT) Register (116)

SL_XMIT is a write-only register used to transmit bit-serial data out of the micro-processor chip under the control of a software timing loop. The value of the TMT bit is transmitted offchip on the **srom_clk_h** signal. In normal operation mode (not in debugging mode), the **srom_clk_h** signal serves both the serial line transmission and the Icache SROM interface (see Sections 7.4 and 7.5). Figure 5–24 and Table 5–9 describe the SL_XMIT register format.

Figure 5–24 Serial Line Transmit (SL_XMIT) Register

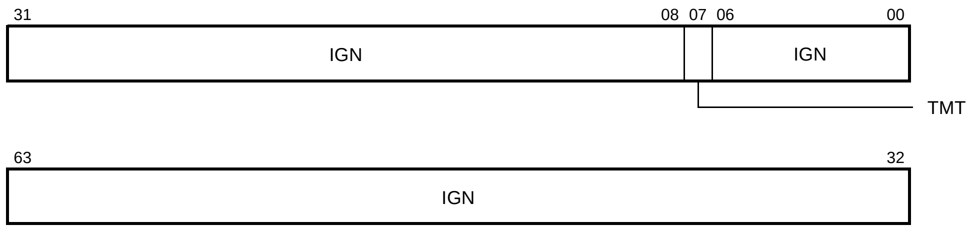


Table 5–9 Serial Line Transmit Register Fields

Name	Extent	Type	Description
TMT	[07]	WO,1	Serial line transmit data

5.1.26 Serial Line Receive (SL_RCV) Register (117)

SL_RCV is a read-only register used to receive bit-serial data under the control of a software timing loop. The RCV bit in the SL_RCV register is functionally connected to the **srom_data_h** signal. A serial line interrupt is requested whenever a transition is detected on the **srom_data_h** signal and the SLE bit in the ICSR is set. During normal operations (not in test mode), the **srom_data_h** signal serves both the serial line reception and the Icache SROM interface (see Sections 7.4 and 7.5).
Figure 5–25 and Table 5–10 describe the SL_RCV register format.

Figure 5–25 Serial Line Receive (SL_RCV) Register

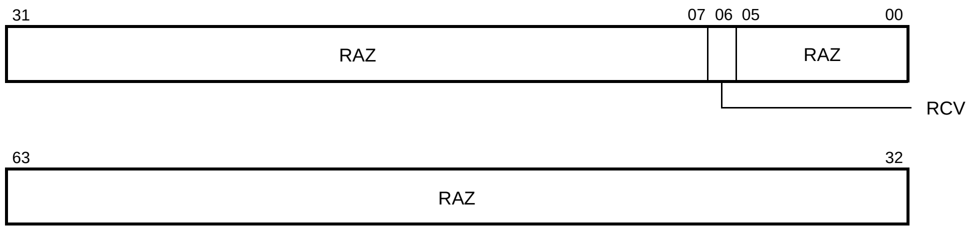


Table 5–10 Serial Line Receive Register Fields

Name	Extent	Type	Description
RCV	[06]	RO	Serial line receive data

5.1.27 Performance Counter (PMCTR) Register (11C)

PMCTR is a read/write register that controls the three onchip performance counters. Figure 5–26 and Table 5–11 describe the PMCTR register format. Performance counter interrupt requests are summarized in Section 5.1.24. CBU inputs to the counter select options are described in the PM0_ MUX[2:0] and PM1_ MUX[2:0] fields of the CBOX_CONFIG2 IPR (see Table 5–29). Section 2.8 describes the performance measurement support features.

Note: The arrangement of the select option tables is not meant to imply any restrictions on permitted combinations of selections. The only cases in which the selection for one counter influences another's count is SEL1=8 (SEL2=2, 3, other).

Figure 5–26 Performance Counter (PMCTR) Register

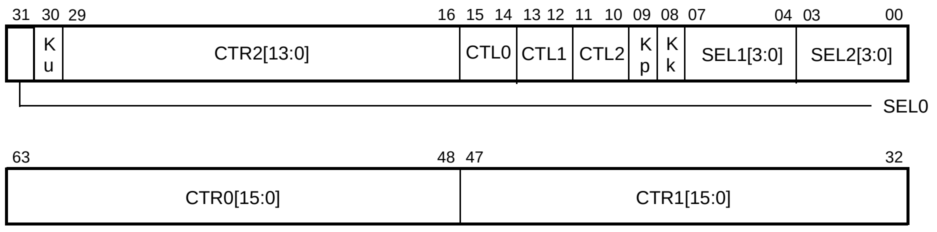


Table 5–11 Performance Counter Register Fields

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

Table 5–12 shows the PMCTR counter select options.

Table 5–12 PMCTR Counter Select Options *(Sheet 1 of 2)*

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

Table 5–12 PMCTR Counter Select Options *(Sheet 2 of 2)*

Counter0 SEL0[0]	Counter1 SEL1[3:0]	Counter2 SEL2[3:0]
		0xB: Reserved
		0xC: CPU cycles
		0xD: MB stall cycles
		0xE: LD*x*L instructions issued
	0xF: pick CBU[0] input	0xF: pick CBU[1] input

Table 5–13 shows the measurement mode bit settings.

Table 5–13 Measurement Mode Control

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

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

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

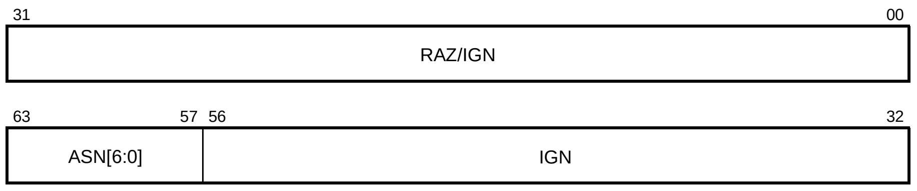
5.2 Memory Address Translation Unit (MTU) IPRs

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

5.2.1 Dstream Translation Buffer Address Space Number (DTB_ASN) Register (200)

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

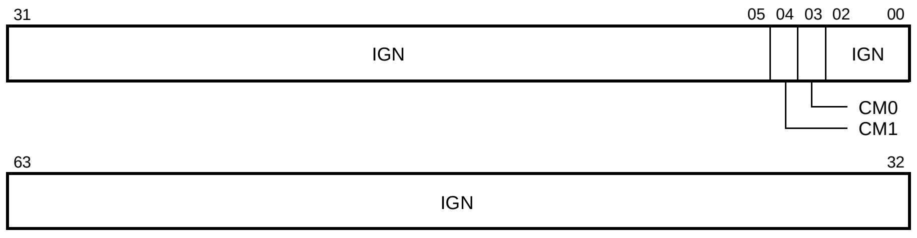
Figure 5–27 Dstream Translation Buffer Address Space Number (DTB_ASN) Register



5.2.2 Dstream Translation Buffer Current Mode (DTB_CM) Register (201)

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

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

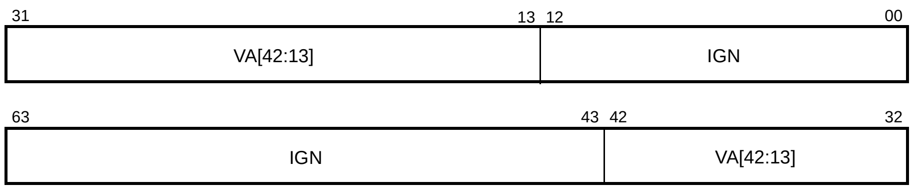


5.2.3 Dstream Translation Buffer Tag (DTB_TAG) Register (202)

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

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

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



5.2.4 Dstream Translation Buffer Page Table Entry (DTB_PTE) Register (203)

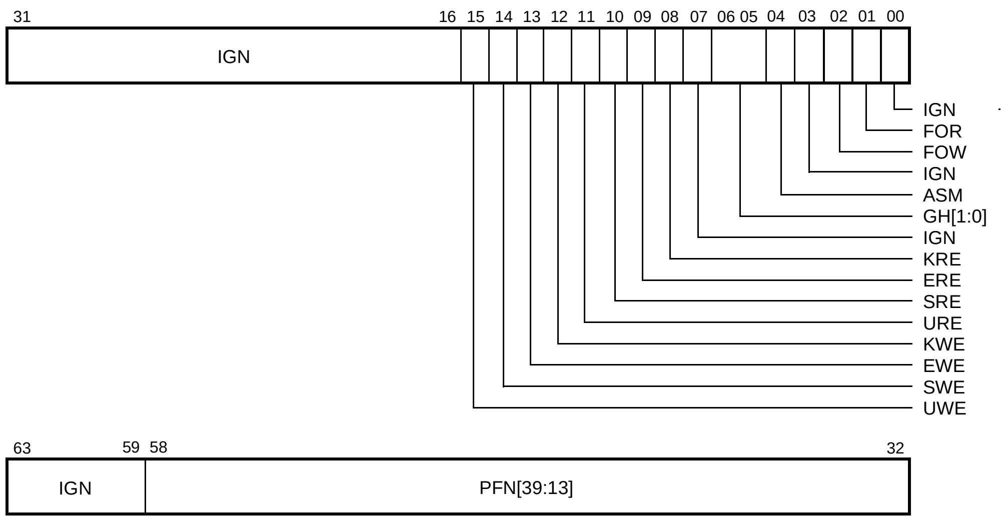
DTB_PTE is a read/write register representing the 64-entry DTB page table entries (PTEs). The entry to be written is chosen by a not-last-used replacement algorithm implemented in hardware. Write operations to DTB_PTE use the memory format bit positions, as described in the *Alpha Architecture Reference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

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

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

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

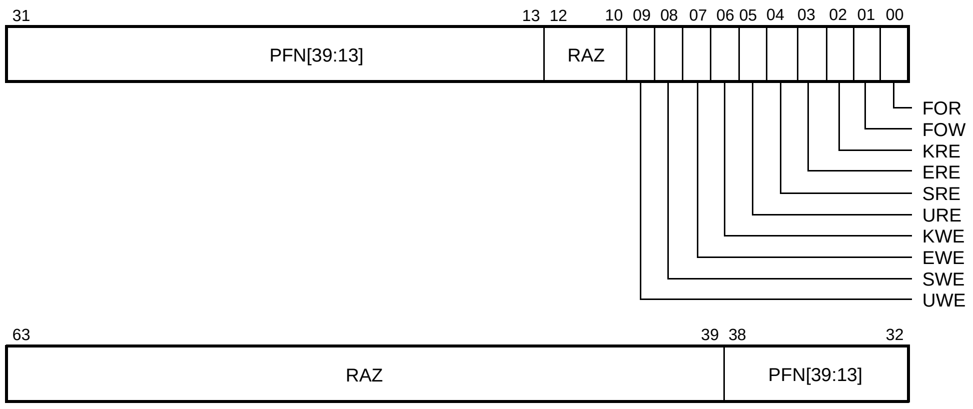
Figure 5–30 Dstream Translation Buffer Page Table Entry (DTB_PTE) Register—Write Format



5.2.5 Dstream Translation Buffer Page Table Entry Temporary (DTB_PTE_TEMP) Register (204)

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

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



5.2.6 Dstream Memory Management Fault Status (MM_STAT) Register (205)

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

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

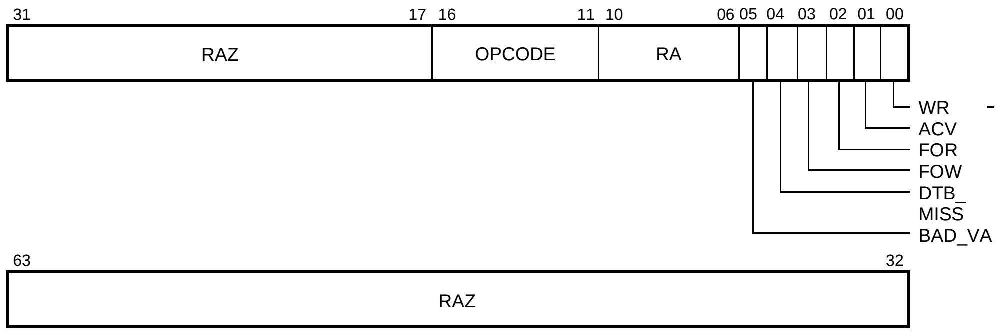


Table 5–14 Dstream Memory Management Fault Status Register Fields *(Sheet 1 of 2)*

Name	Extent	Type	Description
WR	[00]	RO	Set if reference that caused error was a write operation.
ACV	[01]	RO	Set if reference caused an access violation. Includes bad virtual address.
FOR	[02]	RO	Set if reference was a read operation and the PTE FOR bit was set.
FOW	[03]	RO	Set if reference was a write operation and the PTE FOW bit was set.
DTB_MISS	[04]	RO	Set if reference resulted in a DTB miss.

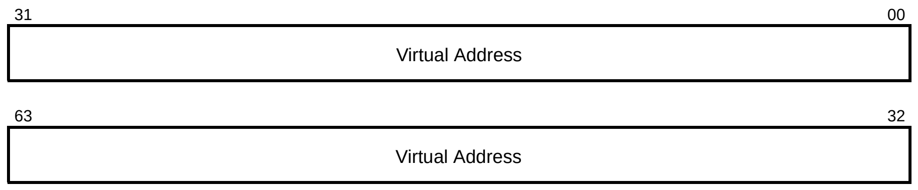
Table 5–14 Dstream Memory Management Fault Status Register Fields (Sheet 2 of 2)

Name	Extent	Type	Description
BAD_VA	[05]	RO	Set if reference had a bad virtual address.
RA	[10:06]	RO	RA field of the faulting instruction.
OPCODE	[16:11]	RO	Opcode field of the faulting instruction.

5.2.7 Faulting Virtual Address (VA) Register (206)

VA is a read-only register. When Dstream faults, DTB misses, or Dcache parity errors occur, the effective virtual address associated with the fault, miss, or error is latched in the VA register. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. The VA register is not unlocked on reset. Figure 5–33 shows the VA register format.

Figure 5–33 Faulting Virtual Address (VA) Register



5.2.8 Formatted Virtual Address (VA_FORM) Register (207)

VA_FORM is a read-only register containing the virtual page table entry (PTE) address calculated as a function of the faulting virtual address and the virtual page table base (VA and MVPTBR registers). This is done as a performance enhancement to the Dstream TBmiss PAL flow.

The virtual address is formatted as a 32-bit PTE when the NT_Mode bit (MCSR[01]) is set (see Figure 5–34). VA_ FORM is locked on any Dstream fault, DTB miss, or Dcache parity error. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. The VA_FORM register is not unlocked on reset. Figure 5–35 shows the VA_FORM register format when MCSR[01] is clear.

Figure 5–34 Formatted Virtual Address (VA_FORM) Register (NT_Mode=1)

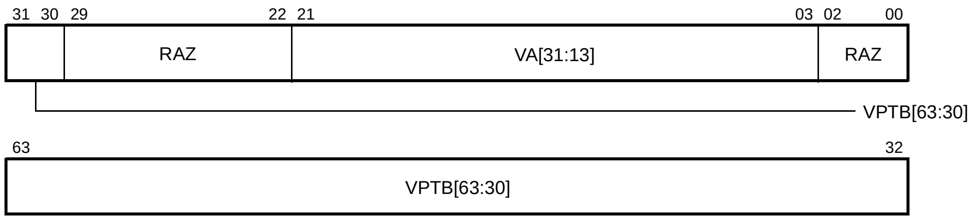


Figure 5–35 Formatted Virtual Address (VA_FORM) Register (NT_Mode=0)

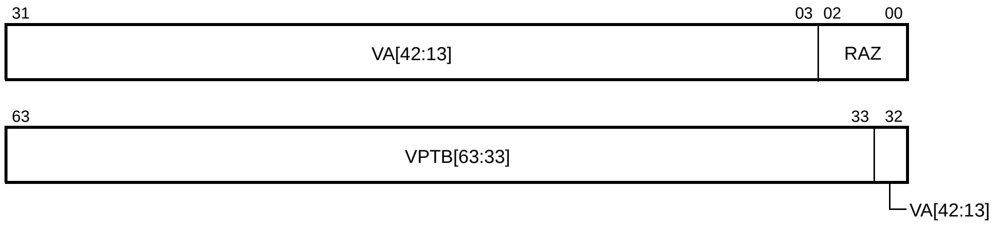


Table 5–15 describes the VA_FORM register fields.

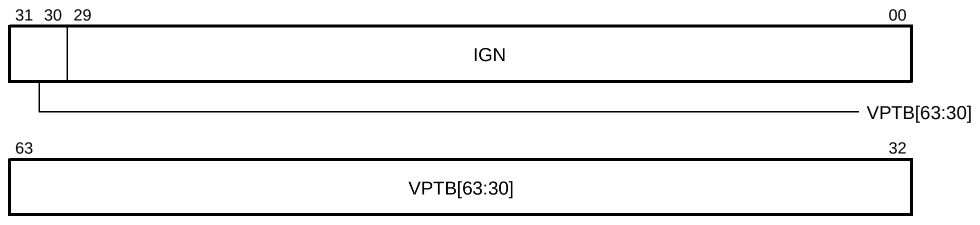
Table 5–15 Formatted Virtual Address Register Fields

Name	Extent	Type	Description
NT_Mode=0			
VPTB	[63:33]	RO	Virtual page table base address as stored in MVPTBR.
VA[42:13]	[32:03]	RO	Subset of the original faulting virtual address.
NT_Mode=1			
VPTB	[63:30]	RO	Virtual page table base address as stored in MVPTBR.
VA[31:13]	[21:03]	RO	Subset of the original faulting virtual address.

5.2.9 MTU Virtual Page Table Base (MVPTBR) Register (208)

MVPTBR is a write-only register containing the virtual address of the base of the page table structure. It is stored in the MTU to be used in calculating the VA_FORM value for the Dstream TBmiss PAL flow. Unlike the VA register, the MVPTBR is not locked against further updates when a Dstream fault, DTB miss, or Dcache parity error occurs. Figure 5–36 shows the MVPTBR register format.

Figure 5–36 MTU Virtual Page Table Base (MVPTBR) Register



5.2.10 Dcache Parity Error Status (DC_PERR_STAT) Register (212)

DC_PERR_STAT is a read/write register that locks and stores Dcache parity error status. The VA, VA_FORM, and MM_STAT registers are locked against further updates until software reads the VA register. If a Dcache parity error is detected while the Dcache parity error status register is unlocked, the error status is loaded into DC_PERR_STAT[05:02]. The LOCK bit is set and the register is locked against further updates (except for the SEO bit) until software writes a 1 to clear the LOCK bit.

The SEO bit is set when a Dcache parity error occurs while the Dcache parity error status register is locked. Once the SEO bit is set, it is locked against further updates until the software writes a 1 to DC_PERR_STAT[00] to unlock and clear the bit. The SEO bit is not set when Dcache parity errors are detected on both pipes within the same cycle. In this particular situation, the pipe0/pipe1 Dcache parity error status bits indicate the existence of a second parity error. The DC_PERR_STAT register is not unlocked or cleared on reset.

Figure 5–37 and Table 5–16 describe the DC_PERR_STAT register format.

Figure 5–37 Dcache Parity Error Status (DC_PERR_STAT) Register

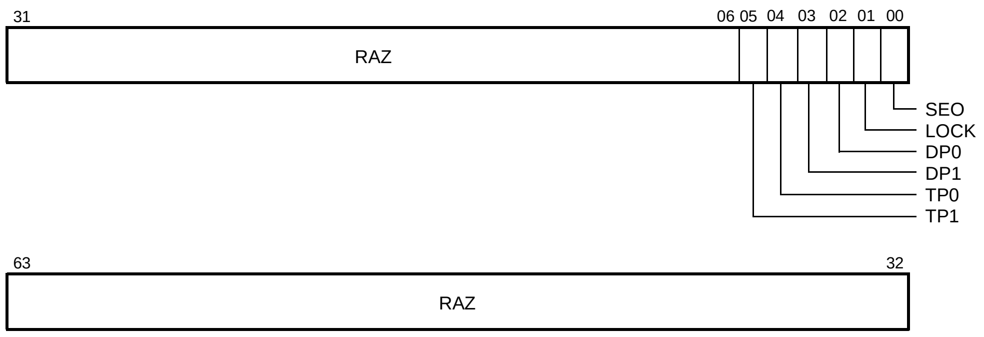


Table 5–16 Dcache Parity Error Status Register Fields

Name	Extent	Type	Description
SEO	[00]	W1C	Set if second Dcache parity error occurred in a cycle after the register was locked. The SEO bit is not set as a result of a second parity error that occurs within the same cycle as the first.
LOCK	[01]	W1C	Set if parity error is detected in Dcache. Bits [05:02] are locked against further updates when this bit is set. Bits [05:02] are cleared when the LOCK bit is cleared.
DP0	[02]	RO	Set on data parity error in Dcache bank 0.
DP1	[03]	RO	Set on data parity error in Dcache bank 1.
TP0	[04]	RO	Set on tag parity error in Dcache bank 0.
TP1	[05]	RO	Set on tag parity error in Dcache bank 1.

5.2.11 Dstream Translation Buffer Invalidate All Process (DTB_IAP) Register (209)

DTB_IAP is a write-only register. Any write operation to this register invalidates all data translation buffer (DTB) entries in which the address space match (ASM) bit is equal to zero.

5.2.12 Dstream Translation Buffer Invalidate All (DTB_IA) Register (20A)

DTB_IA is a write-only register. Any write operation to this register invalidates all 64 DTB entries, and resets the DTB not-last-used (NLU) pointer to its initial state.

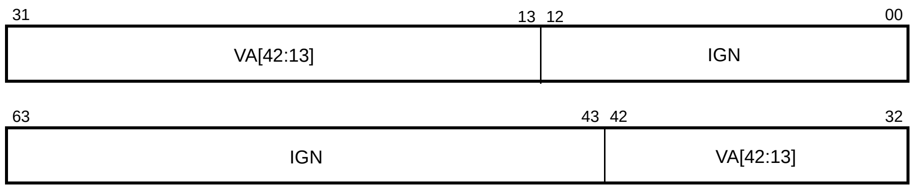
5.2.13 Dstream Translation Buffer Invalidate Single (DTB_IS) Register (20B)

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

- A DTB entry whose VA field matches DTB_IS[42:13] and whose ASN field matches DTB_ASN[63:57].
- A DTB entry whose VA field matches DTB_IS[42:13] and whose ASM bit is set.

Figure 5–38 shows the DTB_IS register format.

Figure 5–38 Dstream Translation Buffer Invalidate Single (DTB_IS) Register



Note: The DTB_IS register is written before the normal IDU trap point. The DTB invalidate single operation is aborted by the IDU only for the following trap conditions:

- ITB miss
- PC mispredict
- When the HW_MTPR DTB_IS is executed in user mode

5.2.14 MTU Control (MCSR) Register (20F)

MCSR is a read/write register that controls features and records status in the MTU. This register is cleared on chip reset but not on timeout reset. Figure 5–39 and Table 5–17 describe the MCSR register format.

Figure 5–39 MTU Control (MCSR) Register

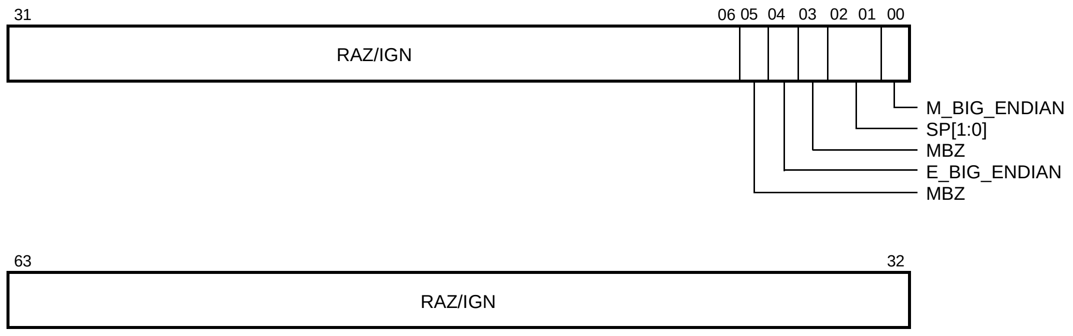


Table 5–17 MTU Control Register Fields

Name	Extent	Type	Description
M_BIG_ ENDIAN	[00]	RW,0	MTU Big Endian mode enable. When set, bit 2 of the physical address is inverted for all long-word Dstream references.
SP[1:0]	[02:01]	RW,0	Superpage mode enables. **Note:** Superpage access is only allowed in kernel mode. SP[1] enables superpage mapping when VA[42:41] = 2. In this mode, virtual addresses VA[39:13] are mapped directly to physical addresses PA[39:13]. Virtual address bit VA[40] is ignored in this translation. SP[0] enables one-to-one superpage mapping of Dstream virtual addresses with VA[42:30] = $1FFE_{16}$. In this mode, virtual addresses VA[29:13] are mapped directly to physical addresses PA[29:13], with bits [39:30] of physical address set to 0. SP[0] is the NT_Mode bit that is used to control virtual address formatting on a read operation from the VA_FORM register.
Reserved	[03]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).
E_BIG_ ENDIAN	[04]	RW,0	IEU Big Endian mode enable. This bit is sent to the IEU to enable Big Endian support for the EXT*xx*, MSK*xx*, and INS*xx* byte instructions. This bit causes the shift amount to be inverted (one's-complemented) prior to the shifter operation.
Reserved	[05]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).

5.2.15 Dcache Mode (DC_MODE) Register (216)

DC_MODE is a read/write register that controls diagnostic and test modes in the Dcache. This register is cleared on chip reset but not on timeout reset. Figure 5–40 and Table 5–18 describe the DC_MODE register format.

Note: The following bit settings are required for normal operation:

DC_ENA = 1
DC_FHIT = 0
DC_BAD_PARITY = 0
DC_PERR_DISABLE = 0

Figure 5–40 Dcache Mode (DC_MODE) Register

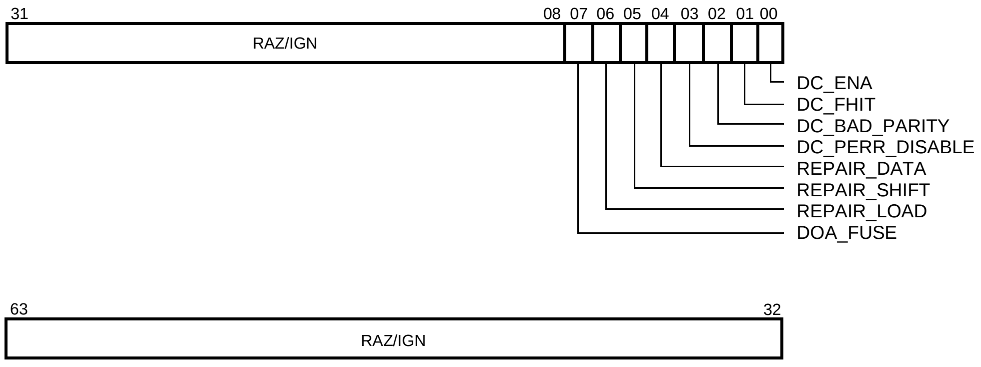


Table 5–18 Dcache Mode Register Fields

Name	Extent	Type	Description
DC_ENA	[00]	RW,0	Software Dcache enable. When set, the DC_ENA bit enables the Dcache. When clear, the Dcache command is not updated by ST or FILL operations, and all LD operations are forced to miss in the Dcache. Must be one (MBO) in normal operation.
DC_FHIT	[01]	RW,0	Dcache force hit. When set, the DC_FHIT bit forces all Dstream references to hit in the Dcache. Must be zero in normal operation.
DC_BAD_ PARITY	[02]	RW,0	When set, the DC_BAD_PARITY bit inverts the data parity inputs to the Dcache on integer stores. This has the effect of putting bad data parity into the Dcache on integer stores that hit in the Dcache. This bit has no effect on the tag parity written to the Dcache during FILL operations, or the data parity written to the CBU write data buffer on integer store instructions. Floating-point store instructions should *not* be issued when this bit is set because it may result in bad parity being written to the CBU write data buffer. Must be zero (MBZ) in normal operation.
DC_PERR_ DISABLE	[03]	RW,0	When set, the DC_PERR_DISABLE bit disables Dcache parity error reporting. When clear, this bit enables all Dcache tag and data parity errors. Parity error reporting is enabled during all other Dcache test modes unless this bit is explicitly set. Must be zero (MBZ) in normal operation.
REPAIR_ DATA	[04]	RW,0	Used during chip manufacturing testing.
REPAIR_ SHIFT	[05]	RW,0	Used during chip manufacturing testing.
REPAIR_ LOAD	[06]	RW,0	Used during chip manufacturing testing.
DOA_FUSE	[07]	RW,0	Used during chip manufacturing testing.

5.2.16 Miss Address File Mode (MAF_MODE) Register (217)

MAF_MODE is a read/write register that controls diagnostic and test modes in the MTU miss address file (MAF). This register is cleared on chip reset. MAF_MODE[05] is also cleared on timeout reset. Figure 5–41 and Table 5–19 describe the MAF_MODE register format.

Note: The following bit settings are required for normal operation:

DREAD_NOMERGE = 0
WB_FLUSH_ALWAYS = 0
WB_NOMERGE = 0
MAF_ARB_DISABLE = 0
WB_CNT_DISABLE = 0

Figure 5–41 Miss Address File Mode (MAF_MODE) Register

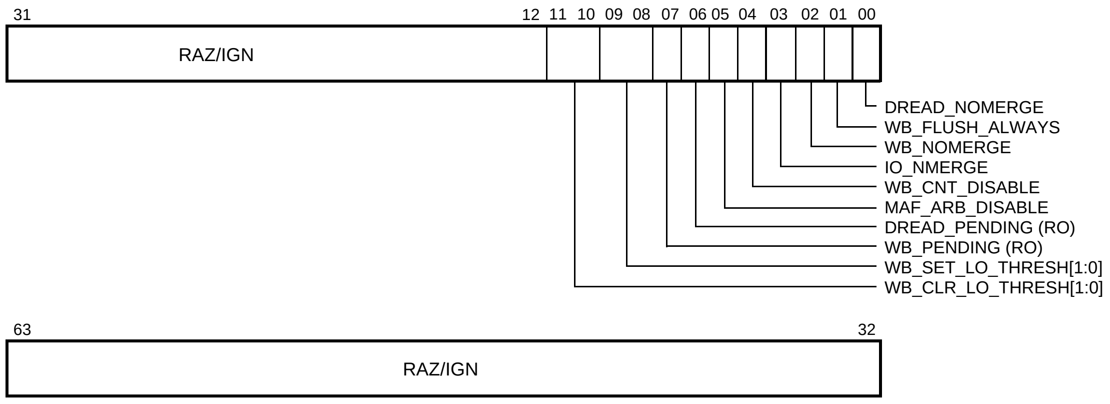


Table 5–19 Miss Address File Mode Register Fields *(Sheet 1 of 2)*

Name	Extent	Type	Description
DREAD_ NOMERGE	[00]	RW,0	Miss address file (MAF) DREAD Merge Disable. When set, this bit disables all merging in the DREAD portion of the MAF. Any load instruction that is issued when DREAD_NOMERGE is set is forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if DREAD_NOMERGE is cleared). Must be zero (MBZ) in normal operation.
WB_FLUSH_ ALWAYS	[01]	RW,0	When set, this bit forces the write buffer to flush whenever there is a valid WB entry. Must be zero (MBZ) in normal operation.
WB_ NOMERGE	[02]	RW,0	When set, this bit disables all merging in the write buffer. Any store instruction that is issued when WB_NOMERGE is set is forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if WB_ NOMERGE is cleared). Must be zero (MBZ) in normal operation.
IO_NMERGE	[03]	RW,0	When set, this bit prevents loads from I/O space (address bit [39]=1) from merging in the MAF. Should be zero (SBZ) in typical operation.
WB_CNT_ DISABLE	[04]	RW,0	When set, this bit disables the 256-cycle WB counter in the MAF arbiter. The top entry of the WB arbitrates at low priority only when a LD*x*_L instruction is issued or the number of WB entries equals or exceeds the value programmed in MAF_MODE[WB_LO_PRIO_THRESH]. Must be zero (MBZ) in normal operation.
MAF_ARB_ DISABLE	[05]	RW,0	When set, this bit disables all DREAD and WB requests in the MAF arbiter. WB_Reissue, Replay, Iref, and MB requests are not blocked from arbitrating. This bit is cleared on both timeout and chip reset. Must be zero (MBZ) in normal operation.
DREAD_ PENDING	[06]	R,0	Indicates the status of the MAF DREAD file. When set, there are one or more outstanding DREAD requests in the MAF file. When clear, there are no outstanding DREAD requests.

Table 5–19 Miss Address File Mode Register Fields *(Sheet 2 of 2)*

Name	Extent	Type	Description
WB_ PENDING	[07]	R,0	This bit indicates the status of the MAF WB file. When set, there are one or more outstanding WB requests in the MAF file. When clear, there are no outstanding WB requests.
WB_SET_LO_ THRESH[1:0]	[09:08]	RW,0	These bits set the threshold at which the WB begins arbitration at low priority. The thresholds are as follows: 00 3 entries 01 4 entries 10 5 entries 11 2 entries (21164 mode) WB_SET_LO_THRESH must be greater than WB_CLR_LO_THRESH.
WB_CLR_LO_ THRESH[1:0]	[11:10]	RW,0	These bits set the threshold at which the WB stops arbitration. The thresholds are as follows: 00 0 entries 01 1 entry (21164 mode) 10 2 entries 11 3 entries WB_SET_LO_THRESH must be greater than WB_CLR_LO_THRESH.

5.2.17 Dcache Flush (DC_FLUSH) Register (210)

DC_FLUSH is a write-only register. A write operation to this register clears all the valid bits in both banks of the Dcache.

5.2.18 Alternate Mode (ALT_MODE) Register (20C)

ALT_MODE is a write-only register that specifies the alternate processor mode used by some HW_LD and HW_ST instructions. Figure 5–42 and Table 5–20 describe the ALT_MODE register format.

Figure 5–42 Alternate Mode (ALT_MODE) Register

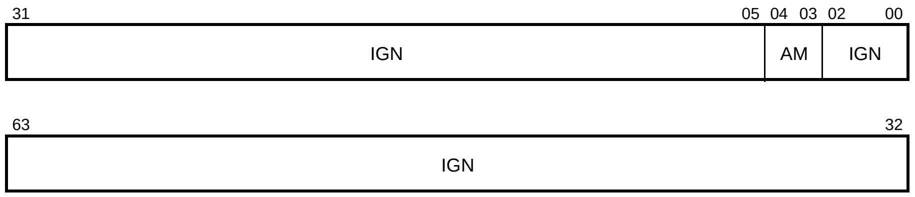


Table 5–20 Alternate Mode Register Settings

ALT_MODE[04:03]	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

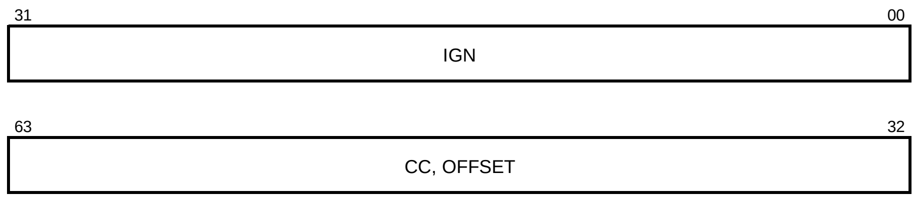
5.2.19 Cycle Counter (CC) Register (20D)

CC is a read/write register. The 21164PC supports it as described in the *Alpha Architecture Reference Manual*. The low half of the counter, when enabled, increments once each CPU cycle. The upper half of the CC register is the counter offset. An HW_MTPR instruction writes CC[63:32]. Bits [31:00] are unchanged. CC_CTL[32] is used to enable or disable the cycle counter. The CC[31:00] is written to CC_CTL by an HW_MTPR instruction.

The CC register is read by the RPCC instruction as defined in the *Alpha Architecture Reference Manual*. The RPCC instruction returns a 64-bit value. The cycle counter is enabled to increment only three cycles after the MTPR CC_CTL (with CC_CTL[32] set) instruction is issued. This means that an RPCC instruction issued four cycles after an HW_MTPR CC_CTL instruction that enables the counter reads a value that is one greater than the initial count.

The CC register is disabled on chip reset. Figure 5–43 shows the CC register format.

Figure 5–43 Cycle Counter (CC) Register



5.2.20 Cycle Counter Control (CC_CTL) Register (20E)

CC_CTL is a write-only register that writes the low 32 bits of the cycle counter to enable or disable the counter. Bits CC[31:04] are written with the value in CC_CTL[31:04] on a HW_MTPR instruction to the CC_CTL register. Bits CC[03:00] are written with zero. Bits CC[63:32] are not changed. If CC_CTL[32] is set, then the counter is enabled; otherwise, the counter is disabled. Figure 5–44 and Table 5–21 describe the CC_CTL register format.

Figure 5–44 Cycle Counter Control (CC_CTL) Register

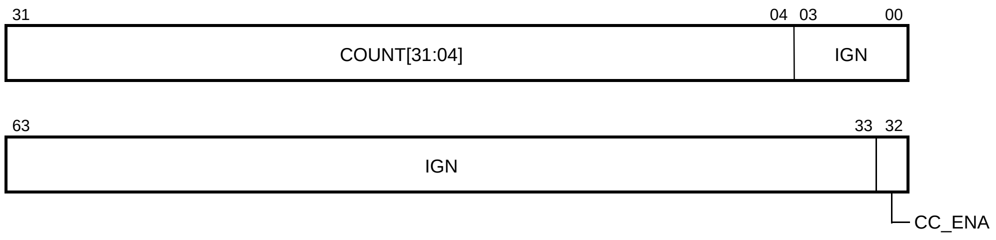


Table 5–21 Cycle Counter Control Register Fields

Name	Extent	Type	Description
COUNT[31:04]	[31:04]	WO	Cycle count. This value is loaded into CC[31:04].
CC_ENA	[32]	WO	Cycle Counter enable. When set, this bit enables the CC register to begin incrementing three cycles later. An RPCC that is issued four cycles after CC_CTL[32] is written “sees” the initial count incremented by 1.

5.2.21 Dcache Test Tag Control (DC_TEST_CTL) Register (213)

DC_TEST_CTL is a read/write register used exclusively for testing and diagnostics. An address written to this register is used to index into the Dcache array when reading or writing to the DC_TEST_TAG register. Figure 5–45 and Table 5–22 describe the DC_TEST_CTL register format. Section 5.2.22 describes how this register is used. DC_TEST_CTL[15] is cleared on reset.

Figure 5–45 Dcache Test Tag Control (DC_TEST_CTL) Register

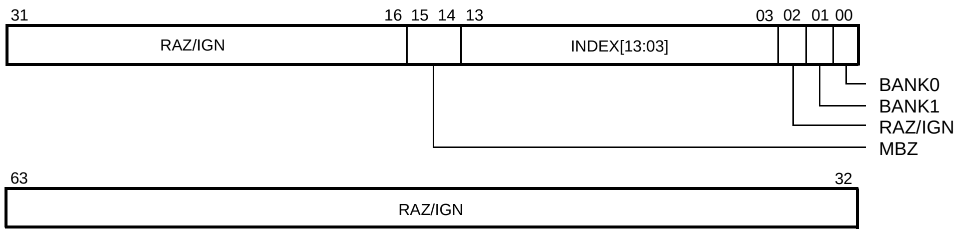


Table 5–22 Dcache Test Tag Control Register Fields

Name	Extent	Type	Description
BANK0	[00]	RW	Dcache Bank0 enable. When set, reads from DC_TEST_TAG return the tag from Dcache bank0, writes to DC_TEST_TAG write to Dcache bank0. When clear, reads from DC_TEST_TAG return the tag from Dcache bank1.
BANK1	[01]	RW	Dcache Bank1 enable. When set, writes to DC_TEST_TAG write to Dcache bank1. This bit has no effect on reads.
INDEX[13:03]	[13:03]	RW	Dcache tag index. This field is used on reads from and writes to the DC_TEST_TAG register to index into the Dcache tag array.
MBZ	[15:14]	RW	Must be zero.

5.2.22 Dcache Test Tag (DC_TEST_TAG) Register (214)

DC_TEST_TAG is a read/write register used exclusively for testing and diagnostics. When DC_TEST_TAG is read, the value in the DC_TEST_CTL register is used to index into the Dcache. The value in the tag, tag parity, valid, and data parity bits for that index are read out of the Dcache and loaded into the DC_TEST_TAG_TEMP register. A zero value is returned to the integer register file (IRF). If BANK0 is set, the read operation is from Dcache bank0. Otherwise, the read operation is from Dcache bank1.

When DC_TEST_TAG is written, the value written to DC_TEST_ TAG is written to the Dcache index referenced by the value in the DC_TEST_CTL register. The tag, tag parity, and valid bits are affected by this write operation. Data parity bits are not affected by this write operation (use DC_MODE[02] and force hit modes). If BANK0 is set, the write operation is to Dcache bank0. If BANK1 is set, the write operation is to Dcache bank1. If both are set, both banks are written.

Figure 5–46 and Table 5–23 describe the DC_TEST_TAG register format.

Figure 5–46 Dcache Test Tag (DC_TEST_TAG) Register

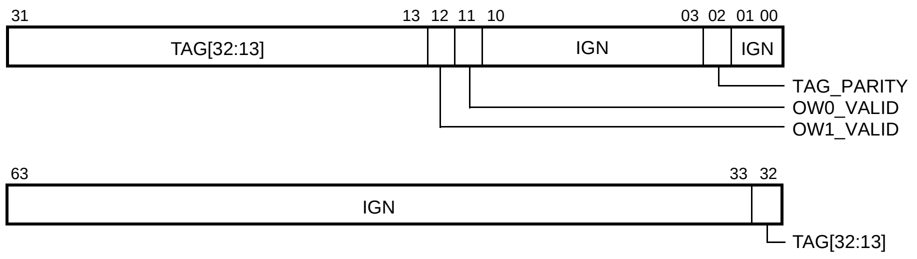


Table 5–23 Dcache Test Tag Register Fields

Name	Extent	Type	Description
TAG_PARITY	[02]	WO	Tag parity. This bit refers to the Dcache tag parity bit that covers tag bits 32 through 13 (valid bits not covered).
OW0_VALID	[11]	WO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low-order octaword within a Dcache 32-byte block.
OW1_VALID	[12]	WO	Octaword valid bit 1. This bit refers to the Dcache valid bit for the high-order octaword within a Dcache 32-byte block.
TAG[32:13]	[32:13]	WO	TAG[32:13]. These bits refer to the tag field in the Dcache array.

5.2.23 Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register (215)

DC_TEST_TAG_TEMP is a read-only register used exclusively for testing and diagnostics.

Reading the Dcache tag array requires a two-step read process:

1. The first read operation from DC_TEST_TAG reads the tag array and data parity bits and loads them into the DC_ TEST_TAG_TEMP register. An UNDEFINED value is returned to the integer register file (IRF).
2. The second read operation of the DC_TEST_TAG_TEMP register returns the Dcache test data to the integer register file (IRF).

Figure 5–47 and Table 5–24 describe the DC_TEST_TAG_TEMP register format.

Figure 5–47 Dcache Test Tag Temporary (DC_TEST_TAG_TEMP) Register

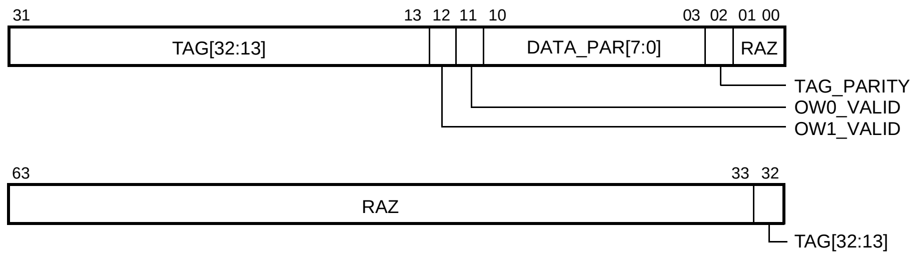


Table 5–24 Dcache Test Tag Temporary Register Fields

Name	Extent	Type	Description
TAG_PARITY	[02]	RO	Tag parity. This bit refers to the Dcache tag parity bit that covers tag bits 32 through 13 (valid bits not covered).
DATA_PAR[7:0]	[10:03]	RO	Data parity. When any of these bits are set, it indicates a parity error occurred in a read of DC_TEST_TAG, in the bank specified in DC_TEST_CTL.
OW0_VALID	[11]	RO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low-order octaword within a Dcache 32-byte block.
OW1_VALID	[12]	RO	Octaword valid bit 1. This bit refers to the Dcache valid bit for the high-order octaword within a Dcache 32-byte block.
TAG[32:13]	[32:13]	RO	TAG[32:13]. These bits refer to the tag field in the Dcache array.

5.3 External Interface Control (CBU) IPRs

Table 5–25 lists specific IPRs for controlling Bcache, system configuration, and logging error information. These IPRs cannot be read or written from the system. They are placed in the 1MB region of 21164PC-specific I/O address space ranging from FF FFF0 0000 to FF FFFF FFFF. Any read or write operation to an undefined IPR in this address space produces UNDEFINED behavior. The operating system should not map any address in this region as writable in any mode.

The CBU internal processor registers are described in Section 5.3.1 through Section 5.3.4.

Table 5–25 CBU Internal Processor Register Descriptions

Register	Address	Type	Description
CBOX_CONFIG	FF FFF0 0008	RW	Contains Bcache configuration parameters.
CBOX_ADDR	FF FFF0 0088	R	Contains the address for Bcache/system-related errors.
CBOX_STATUS	FF FFF0 0108	R	Contains system-to-CPU clock ratio, chip-ID information, and logs Bcache/system-related errors.
CBOX_CONFIG2	FF FFF0 0188	RW	Contains additional Bcache configuration parameters.

5.3.1 CBU Configuration (CBOX_CONFIG) Register (FF FFF0 0008)

CBOX_CONFIG is a read/write register that controls Bcache activity. Figure 5–48 and Table 5–26 describe the CBOX_CONFIG register format. The bits in this register are initialized to the value indicated in Table 5–26 on reset, but not on timeout reset.

Figure 5–48 CBU Configuration (CBOX_CONFIG) Register

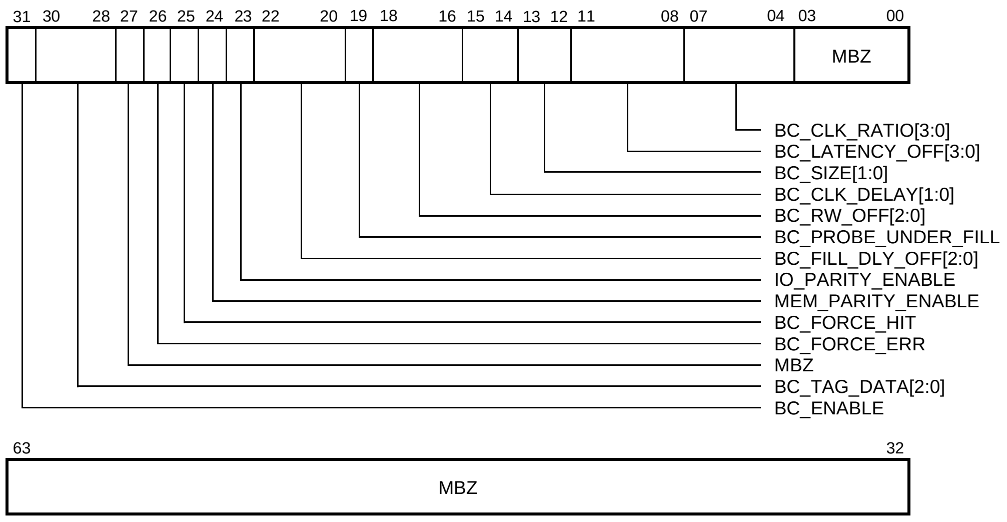


Table 5–26 CBU Configuration Register Fields *(Sheet 1 of 3)*

Name	Extent	Type	Description
Reserved	[03:00]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).
BC_CLK_ RATIO[3:0]	[07:04]	RW,3	This field determines the Bcache clock period (**st_clk**) in number of CPU cycles. At power-up, the **st_clk** remains 0 until the Bcache is enabled. The supported range of values is 2 to10.
BC_ LATENCY_ OFF[3:0]	[11:08]	RW,0	This offset field determines the number of CPU cycles to wait from the CPU clock edge that launches the index until the data is latched into the 21164PC. (Total Latency = 5 + BC_LATENCY_OFF[3:0].) At power-up, this field is initialized to 0, which represents a total Bcache latency of five CPU cycles. The supported range of values for this field is 0 to15, which provides a total latency range of 5 to 20 CPU cycles.
BC_ SIZE[1:0]	[13:12]	RW,0	This field is used to indicate the size of the Bcache. At power-up, this field is initialized to a value that represents a 512KB Bcache. The field encoding is as follows:
BC_CLK_ DELAY[1:0]	[15:14]	RW,1	This field represents the number of CPU cycles to delay the **st_clk** from the driving of the index field during Bcache transactions. At power-up, this field is initialized to 1, indicating a clock delay of one CPU cycle. The supported field range is 0 to 3.

BC_SIZE[1:0]	Size
00	512KB
01	1MB
10	2MB
11	4MB

Table 5–26 CBU Configuration Register Fields *(Sheet 2 of 3)*

Name	Extent	Type	Description
BC_RW_ OFF[2:0]	[18:16]	RW,0	This offset field is used to determine the number of CPU cycles to insert for read-to-write spacing when switching from private Bcache reads to private Bcache writes. (Total read-to-write spacing = 1 + BC_RW_OFF[3:0].) At power-up, this field is initialized to 2, which represents a total read-to-write spacing of three CPU cycles. The supported range of values for this field is 2 to 7, which provides a total read-to-write spacing of three to eight CPU cycles. For other data movement commands, such as FLUSH or FILL from main memory, it is up to the system to direct systemwide data movement in a way that is safe.
BC_PROBE_ UNDER_ FILL	[19]	RW,0	When set, this bit enables Bcache tag probes under fills. This is a performance-enhancement feature that allows the tag store to be read (tag probe) for the next transaction, while the data store is written with fill data from a previous transaction. It allows systems to gain better bus utilization during streaming read misses.
BC_FILL_ DLY_ OFF[2:0]	[22:20]	RW,1	This offset field represents the additional number of CPU cycles to delay the **st_clk** when processing FILL commands. It allows the system designer flexibility to position the Bcache clock within the fill data window. (Total delay of **st_clk** = 1 + BC_FILL_DLY_OFF + BC_CLK_DELAY.)
IO_PARITY_ ENABLE	[23]	RW,0	When set, the 21164PC checks even longword parity during read operations to I/O space (PA[39]=1).
MEM_ PARITY_ ENABLE	[24]	RW,0	When set, the 21164PC checks even longword parity during read operations to memory space (PA[39]=0).
BC_FORCE_ HIT	[25]	RW,0	When set, all read and write operations with PA[39]=0 hit in the Bcache. This is useful when initializing the Bcache on power-up.
BC_FORCE_ ERR	[26]	RW,0	When set, bit zero of each longword written into the Bcache is inverted.
Reserved	[27]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).

Table 5–26 CBU Configuration Register Fields *(Sheet 3 of 3)*

Name	Extent	Type	Description	
BC_TAG_ DATA[2:0]	[30:28]	RW,0	When BC_FORCE_HIT=1, BC_TAG_DATA is used to write the tag field:	
			Bcache Tag Data	**Description**
			BC_TAG_DATA[2]	Bcache tag parity
			BC_TAG_DATA[1]	Bcache tag valid
			BC_TAG_DATA[0]	Bcache tag dirty
BC_ENABLE	[31]	RW,0	When set, this bit enables caching of data and instructions in the Bcache. When clear, Iread and Dread references go to read memory. Dwrites to memory are not allowed.	

5.3.2 CBU Address (CBOX_ADDR) Register (FF FFF0 0088)

CBOX_ADDR is a read-only register that contains the physical address associated with errors reported by the CBOX_STATUS register. Its contents is meaningful only when one of the error bits is set. A read of CBOX_STATUS unlocks the CBOX_ADDR register. Figure 5–49 and Table 5–27 describe the CBOX_ADDR register format.

Figure 5–49 CBU Address (CBOX_ADDR) Register

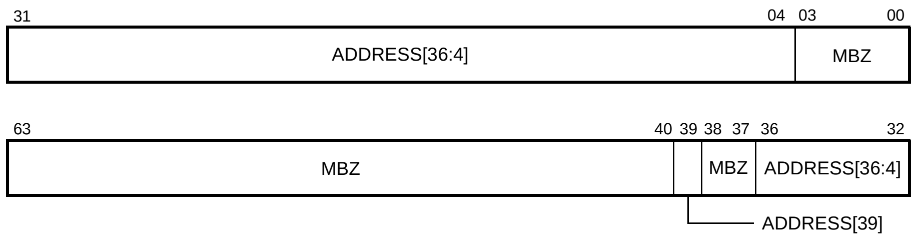


Table 5–27 CBU Address Register Fields

Name	Extent	Type	Description
Reserved	[03:00]	RO	Reserved to COMPAQ. Must be zero (MBZ).
ADDRESS[36:4]	[36:04]	RO	Error address.
Reserved	[38:37]	RO	Reserved to COMPAQ. Must be zero (MBZ).
ADDRESS[39]	[39]	RO	Error address bit 39.
Reserved	[63:40]	RO	Reserved to COMPAQ. Must be zero (MBZ).

5.3.3 CBU Status (CBOX_STATUS) Register (FF FFF0 0108)

CBOX_STATUS is a read-only register. It is locked when any of the error bits are set. Additional errors set the MULTI_ERR error bit in CBOX_STATUS. A read of CBOX_STATUS unlocks and clears CBOX_STATUS and unlocks CBOX_ADDR.

Figure 5–50 and Table 5–28 describe the CBOX_STATUS register format.

Figure 5–50 CBU Status (CBOX_STATUS) Register

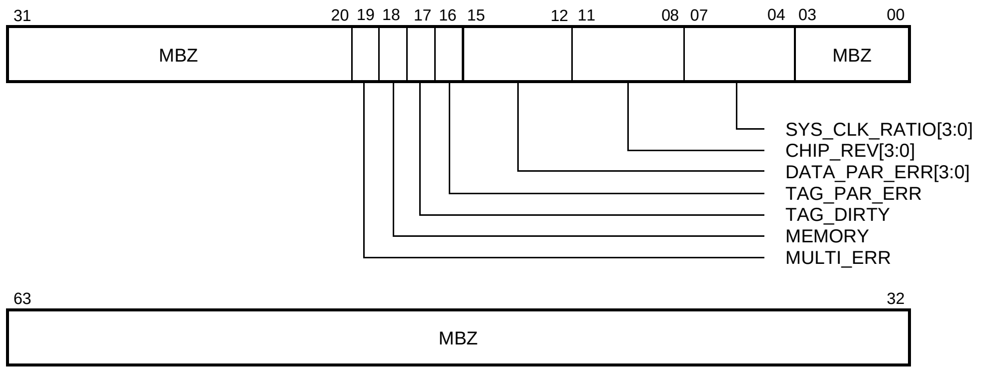


Table 5–28 CBU Status Register Fields *(Sheet 1 of 2)*

Name	Extent	Type	Description
Reserved	[03:00]	RO,0	Reserved to COMPAQ. Must be zero (MBZ).
SYS_CLK_ RATIO[3:0]	[07:04]	RO,0	The **sysclk** period in CPU cycles. The **sysclk** ratio is loaded from the IRQ pins on reset. Note that this field is read only.
CHIP_REV[3:0]	[11:08]	RO,0	This field displays 0001, the current revision of the chip. Future update revisions of the chip will return different unique values.
DATA_PAR_ ERR[3:0]	[15:12]	RO,0	If set, this field indicates that the corresponding long-word had a parity error. Bit[0] corresponds to **data_h[31:0]**, bit[3] corresponds to **data_h[127:96]**.
TAG_PAR_ERR	[16]	RO,0	If set, a parity error was detected on the Bcache tag store.

Table 5–28 CBU Status Register Fields *(Sheet 2 of 2)*

Name	Extent	Type	Description
TAG_DIRTY	[17]	RO,0	This bit is the value of the TAG_DIRTY bit for the failing address. If set, the data had been modified and not written to memory.
MEMORY	[18]	RO,0	If set, the error was detected during a fill from memory.
MULTI_ERR	[19]	RO,0	If set, another error was detected after the register was locked.
Reserved	[63:20]	RO,0	Reserved to COMPAQ. Must be zero (MBZ).

5.3.4 CBU Configuration #2 (CBOX_CONFIG2) Register (FF FFF0 0188)

CBOX_CONFIG2 is a read/write register that controls Bcache and memory, the performance counters, and the debug test port. Figure 5–51 and Table 5–29 describe the CBOX_CONFIG2 register format.

Figure 5–51 CBU Configuration #2 (CBOX_CONFIG2) Register

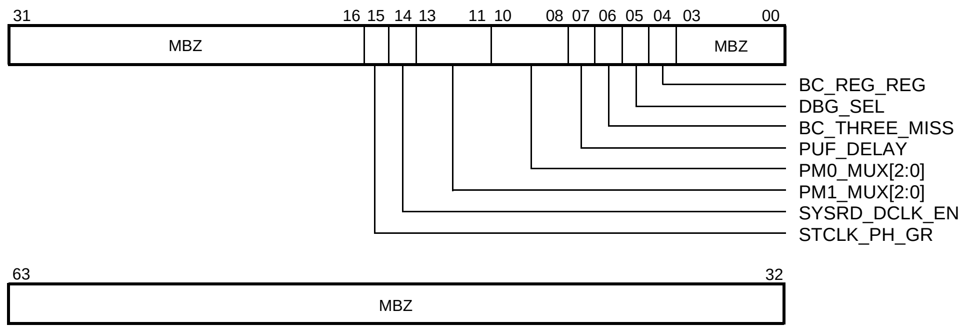


Table 5–29 CBU Configuration #2 Register Fields *(Sheet 1 of 4)*

Name	Extent	Type	Description	
Reserved	[03:00]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).	
BC_REG_REG	[04]	RW,1	When set, this bit indicates that the Bcache is built from REG/REG SSRAM. When clear, it indicates that the Bcache is built from REG/FT SSRAM. This bit is used to delay the deassertion of **data_ram_oe_l** during system Bcache read transactions (for example, Bcache victims or system probes that require data movement).	
DBG_SEL	[05]	RW,0	Selects the Cbox debug information for the debug port.	
			DBG_SEL=0	**DBG_SEL=1**
			biu_trans	head merge
			NOP cmd	tail merge
			rty or abt	RMW tail
			wr_now	stxc
			ri_wr req	fmc != NOP

Table 5–29 CBU Configuration #2 Register Fields *(Sheet 2 of 4)*

Name	Extent	Type	Description	
			DBG_SEL=0	**DBG_SEL=1**
			spa req	spc != NOP
			replay req	scc code[0]
			io_wr or rmv req	scc code[1]
BC_THREE_MISS	[06]	RW,0	Allow three read misses to be launched to the system. This feature assumes the system can guarantee that fills can be returned in order.	
PUF_DELAY	[07]	RW,0	When set in conjunction with PROBE_UNDER_FILL, this bit deasserts **tag_ram_oe_l** in the following manner:	
			PUF_DELAY	**tag_ram_oe_l Deassertion**
			0	Deasserts at third **sysclk** edge after driving the index.
			1	Deasserts one CPU cycle prior to the third **sysclk** edge after driving the index.
PM0_MUX[2:0]	[10:08]	RW,0	This field selects the CBU events used for performance counter #0.	
			PM0_MUX [2:0]	**Counter 0 is used to count:**
			0x0	Total Bcache read requests (the total number of read requests from the MTU).
			0x1	Bcache Dstream read hits (total number of Dstream read requests that hit in the Bcache).
			0x2	Bcache Dstream read fills (the total number of Dstream read fill requests to the Bcache).
			0x3	Bcache write operations (the total number of write requests from the MTU).
			0x4	Undefined.
			0x5	Bcache clean write hits (the total number of write operations that hit a clean block in the Bcache).

Table 5–29 CBU Configuration #2 Register Fields *(Sheet 3 of 4)*

Name	Extent	Type	Description	
			PM0_MUX [2:0]	**Counter 0 is used to count:**
			0x6	Bcache victims (the total number of VICTIM commands issued by the 21164PC).
			0x7	Read miss 2 launched (the number of times a second READ MISS request is sent to the system while there is already an outstanding READ MISS command).
PM1_MUX[2:0]	[13:11]	RW,0	This field selects the CBU events used for performance counter #1.	
			PM1_MUX [2:0]	**Counter 1 is used to count:**
			0x0	Bcache Dstream read requests (the total number of Dstream read requests from the MTU).
			0x1	Bcache read hits (the total number of read requests that hit in the Bcache).
			0x2	Bcache read fills (the total number of read fill operations in the Bcache).
			0x3	Bcache write hits (the total number of write operations that hit in the Bcache).
			0x4	Bcache write fills (the total number of write fill operations in the Bcache).
			0x5	System read/flush Bcache hits (the total number of system READ or FLUSH hits in the Bcache).
			0x6	System read/flush Bcache misses (the total number of system READ or FLUSH requests).
			0x7	Read miss 3 launched (the number of times a third READ MISS request is sent to the system while there are already two READ MISSes outstanding).

Table 5–29 CBU Configuration #2 Register Fields *(Sheet 4 of 4)*

Name	Extent	Type	Description
SYSRD_DCLK_EN	[14]	RW,0	When set, this bit is used to support pipelined SSRAM in a 21174-based core-chip environment. It aids module timing for sampling system Bcache reads (Bcache victims and system probes with data movement). Restrictions: 1. 6 ≤ **sysclk**_ratio ≤ 11. 2. **bc_clk**_ratio ≤ ROUND_DOWN(**sysclk**_ratio/2).
STCLK_PH_GR	[15]	RW,0	When set, this bit delays **st_clkx_h** and additional CPU phase.
Reserved	[63:16]	RW,0	Reserved to COMPAQ. Must be zero (MBZ).

5.4 PALcode Storage Registers

The 21164PC IEU register file has eight extra registers that are called the PALshadow registers. The PALshadow registers overlay R8 through R14 and R25 when the CPU is in PALmode and ICSR[SDE] is set. Thus, PALcode can consider R8 through R14 and R25 as local scratch. PALshadow registers cannot be written in the last two cycles of a PALcode flow. The normal state of the CPU is ICSR[SDE] = ON. PALcode disables SDE for the unaligned trap and for error flows.

The IDU holds a bank of 24 PALtemp registers. The PALtemp registers are accessed with the HW_MTPR and HW_MFPR instructions. The latency from a PALtemp read operation to availability is one cycle.

5.5 Restrictions

The following sections list all known register access restrictions. A software tool called the PALcode violation checker (PVC) is available. This tool can be used to verify adherence to many of the PALcode restrictions.

5.5.1 CBU IPR PALcode Restrictions

Table 5–30 describes the CBU IPR PALcode restrictions.

Table 5–30 CBU IPR PALcode Restrictions

Condition	Restriction
Store to CBOX_CONFIG or CBOX_CONFIG2.	Must be preceded by MB, must be followed by MB, must have no concurrent cacheable Istream references or concurrent system commands.
Load from any CBOX IPR at initialization prior to the Bcache being enabled.	Must guarantee that there are *no* outstanding read misses.
Load from CBOX_STATUS.[1]	Unlocks CBOX_ADDR and CBOX_STATUS.
Any CBU IPR address.	No LD*x*_L or ST*x*_C.
Any undefined CBU IPR address.	No store instructions.
Bcache in force hit mode.	No ST*x*_C to cacheable space.
Clearing of BC_FORCE_HIT in CBOX_CONFIG.	Must be followed by MB, read operation of CBOX_STATUS, then MB prior to subsequent store.

[1] CBOX_ADDR must be read *before* CBOX_STATUS to ensure error address.

5.5.2 PALcode Restrictions—Instruction Definitions

MTU instructions are: LD*x*, LDQ_U, LD*x*_L, HW_LD, ST*x*, STQ_U, ST*x*_C, HW_ST, and FETCH*x*.

Virtual MTU instructions are: LD*x*, LDQ_U, LD*x*_L, HW_LD (virtual), ST*x*, STQ_U, ST*x*_C, HW_ST (virtual), and FETCH*x*.

Load instructions are: LD*x*, LDQ_U, LD*x*_L, and HW_LD.

Store instructions are: ST*x*, STQ_U, ST*x*_C, and HW_ST.

Restrictions

Table 5–31 lists PALcode restrictions.

Table 5–31 PALcode Restrictions *(Sheet 1 of 4)*

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
CALL_PAL entry	No HW_REI or HW_REI_STALL in cycle 0. No HW_MFPR EXC_ADDR in cycle 0,1.	Y Y
PALshadow write instruction	No HW_REI or HW_REI_STALL in 0, 1.	Y
HW_LD, lock bit set	PAL must slot to E0. No other MTU instruction in 0.	
HW_LD, VPTE bit set	No other virtual reference in 0.	
Any load instruction	No MTU HW_MTPR or HW_MFPR in 0. No HW_MFPR MAF_MODE in 1,2 (DREAD_PENDING may not be updated). No HW_MFPR DC_PERR_STAT in 1,2. No HW_MFPR DC_TEST_TAG slotted in 0.	Y Y Y
Any store instruction	No HW_MFPR DC_PERR_STAT in 1,2. No HW_MFPR MAF_MODE in 1,2 (WB_PENDING may not be updated).	Y Y
Any virtual MTU instruction	No HW_MTPR DTB_IS in 1.	Y
Any MTU instruction or WMB, if it traps	HW_MTPR any IDU IPR not aborted in 0,1 (except that EXC_ADDR is updated with correct faulting PC). HW_MTPR DTB_IS not aborted in 0,1.	Y Y
Any IDU trap except PC-mispredict, ITBMISS, or OPCDEC due to user mode	HW_MTPR DTB_IS not aborted in 0,1.	
HW_REI_STALL	Only one HW_REI_STALL in an aligned block of four instructions.	
HW_MTPR any undefined IPR number	Illegal in any cycle.	
ARITH trap entry	No HW_MFPR EXC_SUM or EXC_MASK in cycle 0,1.	Y
Machine check trap entry	No register file read or write access in 0,1,2,3,4,5,6,7. No HW_MFPR EXC_SUM or EXC_MASK in cycle 0,1.	 Y

Table 5–31 PALcode Restrictions *(Sheet 2 of 4)*

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR any IDU IPR (including PALtemp registers)	No HW_MFPR same IPR in cycle 1,2. No floating-point conditional branch in 0. No FEN or OPCDEC instruction in 0.	Y
HW_MTPR ASTRR, ASTER	No HW_MFPR INTID in 0,1,2,3,4,5. No HW_REI in 0,1.	Y Y
HW_MTPR SIRR	No HW_MFPR INTID in 0,1,2,3,4.	Y
HW_MTPR EXC_ADDR	No HW_REI in cycle 0,1.	Y
HW_MTPR IC_FLUSH_CTL	Must be followed by 44 inline PALcode instructions.	
HW_MTPR ICSR: HWE	No HW_REI in 0,1,2,3.	Y
HW_MTPR ICSR: FPE	No floating-point instructions in 0, 1, 2, 3. No HW_REI in 0,1,2.	
HW_MTPR ICSR: FFP	Should not set FFP = 0 until ITB_ASN has been written with a valid value.	
HW_MTPR ICSR: SPE, FMS	If HW_REI_STALL, then no HW_REI_STALL in 0,1. If HW_REI, then no HW_REI in 0,1,2,3,4.	Y Y
HW_MTPR ICSR: SPE	Must flush Icache.	
HW_MTPR ICSR: SDE	No PALshadow read/write access in 0,1,2,3. No HW_REI in 0,1,2.	 Y
HW_MTPR ITB_ASN	Must be followed by HW_REI_STALL. No HW_REI_STALL in cycle 0,1,2,3,4. No HW_MTPR ITB_IS in 0,1,2,3.	 Y Y
HW_MTPR ITB_PTE	Must be followed by HW_REI_STALL.	
HW_MTPR ITB_IAP, ITB_IS, ITB_IA	Must be followed by HW_REI_STALL.	
HW_MTPR ITB_IS	HW_REI_STALL must be in the same Istream octaword.	
HW_MTPR IVPTBR	No HW_MFPR IFAULT_VA_FORM in 0,1,2.	Y
HW_MTPR PAL_BASE	No CALL_PAL in 0,1,2,3,4,5,6,7. No HW_REI in 0,1,2,3,4,5,6.	Y Y

Restrictions

Table 5–31 PALcode Restrictions *(Sheet 3 of 4)*

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR ICM	No HW_REI in 0,1,2.	Y
	No private CALL_PAL in 0,1,2,3.	
HW_MTPR CC, CC_CTL	No RPCC in 0,1,2.	Y
	No HW_REI in 0,1.	Y
HW_MTPR DC_FLUSH	No MTU instructions in 1,2.	Y
	No outstanding fills in 0.	
	No HW_REI in 0,1.	Y
HW_MTPR DC_MODE	No MTU instructions in 1,2,3,4.	Y
	No HW_MFPR DC_MODE in 1,2.	Y
	No outstanding fills in 0.	
	No HW_REI in 0,1,2,3.	Y
	No HW_REI_STALL in 0,1.	Y
HW_MTPR DC_PERR_STAT	No load or store instructions in 1.	Y
	No HW_MFPR DC_PERR_STAT in 1,2.	Y
HW_MTPR DC_TEST_CTL	No HW_MFPR DC_TEST_TAG in 1,2,3.	Y
	No HW_MFPR DC_TEST_CTL issued or slotted in 1,2.	
HW_MTPR DC_TEST_TAG	No outstanding DC fills in 0.	
	No HW_MFPR DC_TEST_TAG in 1,2,3.	Y
HW_MTPR DTB_ASN	No virtual MTU instructions in 1,2,3.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR DTB_CM, ALT_MODE	No virtual MTU instructions in 1,2.	Y
	No HW_REI in 0,1.	Y
HW_MTPR DTB_PTE	No virtual MTU instructions in 2.	Y
	No HW_MTPR DTB_ASN, DTB_CM, ALT_MODE, MCSR, MAF_MODE, DC_MODE, DC_ PERR_STAT, DC_TEST_CTL, DC_TEST_TAG in 2.	Y
HW_MTPR DTB_TAG	No virtual MTU instructions in 1,2,3.	Y
	No HW_MTPR DTB_TAG in 1.	Y
	No HW_MFPR DTB_PTE in 1,2.	Y
	No HW_MTPR DTB_IS in 1,2.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR DTB_IAP, DTB_IA	No virtual MTU instructions in 1,2,3.	Y
	No HW_MTPR DTB_IS in 0,1,2.	Y
	No HW_REI in 0,1,2.	Y

Table 5–31 PALcode Restrictions *(Sheet 4 of 4)*

The following in cycle 0:	Restrictions (Note: Numbers refer to cycle number):	Y if checked by PVC[1]
HW_MTPR DTB_IA	No HW_MFPR DTB_PTE in 1.	Y
HW_MTPR MAF_MODE	No MTU instructions in 1,2,3.	Y
	No WMB in 1,2,3.	Y
	No HW_MFPR MAF_MODE in 1,2.	Y
	No HW_REI in 0,1,2.	Y
HW_MTPR MCSR	No virtual MTU instructions in 0,1,2,3,4.	Y
	No HW_MFPR MCSR in 1,2.	Y
	No HW_MFPR VA_FORM in 1,2,3.	Y
	No HW_REI in 0,1,2,3.	Y
	No HW_REI_STALL in 0,1.	Y
HW_MTPR MVPTBR	No HW_MFPR VA_FORM in 1,2.	Y
HW_MFPR ITB_PTE	No HW_MFPR ITB_PTE_TEMP in 1,2,3.	Y
HW_MFPR DC_TEST_TAG	No outstanding DC fills in 0.	
	No HW_MFPR DC_TEST_TAG_TEMP issued or slotted in 1.	
	No LD*x* instructions slotted in 0.	
	No HW_MTPR DC_TEST_CTL between HW_ MFPR DC_TEST_TAG and HW_MFPR DC_TEST_ TAG_TEMP.	
HW_MFPR DTB_PTE	No MTU instructions in 0,1.	Y
	No HW_MTPR DC_TEST_CTL, DC_TEST_TAG in 0,1.	Y
	No HW_MFPR DTB_PTE_TEMP issued or slotted in 1,2,3.	
	No HW_MFPR DTB_PTE in 1.	
	No virtual MTU instructions in 0,1,2.	Y
		Y
HW_MFPR VA	Must be done in ARITH, MACHINE CHECK, DTBMISS_SINGLE, UNALIGN, DFAULT traps and ITBMISS flow after the VPTE load.	

[1] PALcode violation checker.

6 Privileged Architecture Library Code

This chapter describes the 21164PC privileged architecture library code (PALcode). The chapter is organized as follows:

- PALcode description
- PALmode environment
- Invoking PALcode
- PALcode entry points
- Required PALcode function codes
- 21164PC implementation of the architecturally reserved opcodes

6.1 PALcode Description

Privileged architecture library code (PALcode) is macrocode that provides an architecturally defined operating-system-specific programming interface that is common across all Alpha microprocessors. The actual implementation of PALcode differs for each operating system.

PALcode runs with privileges enabled, instruction stream mapping disabled, and interrupts disabled. PALcode has privilege to use five special opcodes that allow functions such as physical data stream references and internal processor register (IPR) manipulation.

PALcode can be invoked by the following events:

- Reset
- System hardware exceptions (MCHK, ARITH)
- Memory-management exceptions
- Interrupts

- CALL_PAL instructions

PALcode has characteristics that make it appear to be a combination of microcode, ROM BIOS, and system service routines, though the analogy to any of these other items is not exact. PALcode exists for several major reasons:

- There are some necessary support functions that are too complex to implement directly in a processor chip's hardware, but that cannot be handled by a normal operating system software routine. Routines to fill the translation buffer (TB), acknowledge interrupts, and dispatch exceptions are some examples. In some architectures, these functions are handled by microcode, but the Alpha architecture is careful not to mandate the use of microcode so as to allow reasonable chip implementations.
- There are functions that must run atomically, yet involve long sequences of instructions that may need complete access to all the underlying computer hardware. An example of this is the sequence that returns from an exception or interrupt.
- There are some instructions that are necessary for backward compatibility or ease of programming; however, these are not used often enough to dedicate them to hardware, or are so complex that they would jeopardize the overall performance of the computer. For example, an instruction that does a VAX style interlocked memory access might be familiar to someone used to programming on a CISC machine, but is not included in the Alpha architecture. Another example is the emulation of an instruction that has no direct hardware support in a particular chip implementation.

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

6.2 PALmode Environment

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

- Istream memory mapping is disabled. Because the PALcode is used to implement translation buffer fill routines, Istream mapping clearly cannot be enabled. Dstream mapping is still enabled.

- The program has privileged access to all the computer hardware. Most of the functions handled by PALcode are privileged and need control of the lowest levels of the system.
- Interrupts are disabled. If a long sequence of instructions need to be executed atomically, interrupts cannot be allowed.

An important aspect of PALcode is that it uses normal Alpha instructions for most of its operations; that is, the same instruction set that nonprivileged Alpha programmers use. There are a few extra instructions that are only available in PALmode, and will cause a dispatch to the OPCDEC PALcode entry point if attempted while not in PALmode. The Alpha architecture allows some flexibility in what these special PALmode instructions do. In the 21164PC, the special PALmode-only instructions perform the following functions:

- Read or write internal processor registers (HW_MFPR, HW_ MTPR).
- Perform memory load or store operations without invoking the normal memory-management routines (HW_LD, HW_ST).
- Return from an exception or interrupt (HW_REI) .

When executing in PALmode, there are certain restrictions for using the privileged instructions because PALmode gives the programmer complete access to many of the internal details of the 21164PC. Refer to Section 6.6 for information on these special PALmode instructions.

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

6.3 Invoking PALcode

PALcode is invoked at specific entry points, under certain well-defined conditions. These entry points provide access to a series of callable routines, with each routine indexed as an offset from a base address. The base address of the PALcode is programmable (stored in the PAL_BASE IPR), and is normally set by the system reset code. Refer to Section 6.4 for additional information on PALcode entry points.

PC[00] is used as the PALmode flag both to the hardware and to PALcode itself. When the CPU enters a PALflow, the IDU sets PC[00]. This bit remains set as instructions are executed in the PAL Istream. The IDU hardware ignores this and behaves as if the PC were still longword aligned for the purposes of Istream fetch and execute. On HW_REI, the new state of PALmode is copied from EXC_ADDR[00].

When an event occurs that needs to invoke PALcode, the 21164PC first drains the pipeline. The current PC is loaded into the EXC_ADDR IPR, and the appropriate PALcode routine is dispatched. These operations occur under direct control of the chip hardware, and the machine is now in PALmode. When the HW_REI instruction is executed at the end of the PALcode routine, the hardware executes a jump to the address contained in the EXC_ ADDR IPR. The LSB is used to indicate PALmode to the hardware. Generally, the LSB is clear upon return from a PALcode routine, in which case, the hardware loads the new PC, enables interrupts, enables memory mapping, and dispatches back to the user.

The most basic use of PALcode is to handle complex hardware events, and it is called automatically when the particular hardware event is sensed. This use of PALcode is similar to other architectures' use of microcode.

There are several major categories of hardware-initiated invocations of PALcode:

- When the 21164PC is reset, it enters PALmode and executes the RESET PALcode. The system will remain in PALmode until a HW_ REI instruction is executed and EXC_ADDR[00] is cleared. It then continues execution in non-PALmode (native mode), as just described. It is during this initial RESET PALcode execution that the rest of the low-level system initialization is performed, including any modification to the PALcode base register.
- When a system hardware error is detected by the 21164PC, 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 21164PC senses an interrupt, it dispatches the acknowledgment of the interrupt to a PALcode routine that does the necessary information gathering, then handles the situation appropriately for the given interrupt.
- When a Dstream or Istream translation buffer miss occurs, one of several PALcode routines is called to perform the TB fill.

The 21164PC IEU register file has eight extra registers that are called the PALshadow registers. The PALshadow registers overlay R8, R9, R10, R11, R12, R13, R14, and R25 when the CPU is in PALmode and ICSR[SDE] is asserted. For additional PAL scratch, the IDU has a register bank of 24 PALtemp registers, which are accessible via HW_MTPR and HW_MFPR instructions.

6.4 PALcode Entry Points

PALcode is invoked at specific entry points. The 21164PC has two types of PALcode entry points: CALL_PAL and traps.

6.4.1 CALL_PAL Entry Points

CALL_PAL entry points are used whenever the IDU encounters a CALL_PAL instruction in the instruction stream (Istream). CALL_ PAL instructions start at the following offsets:

- Privileged CALL_PAL instructions start at offset 2000_{16}.
- Nonprivileged CALL_PAL instructions start at offset 3000_{16}.

The CALL_PAL itself is issued into pipe E1 and the IDU stalls for the minimum number of cycles necessary to perform an implicit TRAPB. The PC of the instruction immediately following the CALL_PAL is loaded into EXC_ADDR and is pushed onto the return prediction stack.

The IDU contains special hardware to minimize the number of cycles in the TRAPB at the start of a CALL_PAL. Software can benefit from this by scheduling CALL_PALs such that they do not fall in the shadow of:

- IMUL
- Any floating-point operate, especially FDIV

Each CALL_PAL instruction includes a function field that will be used in the calculation of the next PC. The PAL OPCDEC flow will be started if the CALL_PAL function field is:

- In the range 40_{16} to $7F_{16}$ inclusive
- Greater than BF_{16}
- Between 00_{16} and $3F_{16}$ inclusive, and ICM[04:03] is not equal to kernel

If no OPCDEC is detected on the CALL_PAL function, then the PC of the instruction to execute after the CALL_PAL is calculated as follows:

- PC[63:14] = PAL_BASE IPR[63:14]
- PC[13] = 1
- PC[12] = CALL_PAL function field[7]
- PC[11:06] = CALL_PAL function field[5:0]

- PC[05:01] = 0
- PC[00] = 1 (PALmode)

The minimum number of cycles for a CALL_PAL execution is four.

Number of Cycles	Description
1	Minimum TRAPB for empty pipe. Typically this will be four cycles.
1	Issue the CALL_PAL instruction.
2	The minimum length of a PAL flow. However, in most cases there will be more than two cycles of work for the CALL_PAL.

6.4.2 PALcode Trap Entry Points

Chip-specific trap entry points start PALcode. (No PALcode assist is required for replay and mispredict type traps.) EXC_ ADDR is loaded with the return PC and the IDU performs a TRAPB in the shadow of the trap. The return prediction stack is pushed with the PC of the trapping instruction for precise traps, and with some later PC for imprecise traps.

Table 6–1 shows the PALcode trap entry points and their offset from the PAL_BASE IPR. Entry points are listed from highest to lowest priority. (Prioritization among the Dstream traps works because DTBMISS is suppressed when there is a sign check error. The priority of ITBMISS and interrupt is reversed if there is an Icache miss.)

Table 6–1 PALcode Trap Entry Points *(Sheet 1 of 2)*

Entry Name	Offset_{16}	Description
RESET	0000	Reset
IACCVIO	0080	Istream access violation or sign check error on PC
INTERRUPT	0100	Interrupt: hardware, software, and AST
ITBMISS	0180	Istream TBMISS
DTBMISS_SINGLE	0200	Dstream TBMISS
DTBMISS_DOUBLE	0280	Dstream TBMISS during virtual page table entry (PTE) fetch
UNALIGN	0300	Dstream unaligned reference
DFAULT	0380	Dstream fault or sign check error on virtual address

Table 6–1 PALcode Trap Entry Points *(Sheet 2 of 2)*

Entry Name	Offset$_{16}$	Description
MCHK	0400	Uncorrected hardware error
OPCDEC	0480	Illegal opcode
ARITH	0500	Arithmetic exception
FEN	0580	Floating-point operation attempted with: • Floating-point instructions (LD, ST, and operates) disabled through FPE bit in the ICSR IPR • Floating-point IEEE operation with data type other than S, T, or Q

6.5 Required PALcode Function Codes

Table 6–2 lists opcodes required for all Alpha implementations. The notation used is *oo.ffff*, where *oo* is the hexadecimal 6-bit opcode and *ffff* is the hexadecimal 26-bit function code.

Table 6–2 Required PALcode Function Codes

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

6.6 21164PC Implementation of the Architecturally Reserved Opcodes

PALcode uses the Alpha instruction set for most of its operations. Table 6–3 lists the opcodes reserved by the Alpha architecture for implementation-specific use. These opcodes are privileged and are only available in PALmode.

21164PC Implementation of the Architecturally Reserved Opcodes

Note: These architecturally reserved opcodes contain different options to the 21064 opcodes of the same names.

Table 6–3 Opcodes Reserved for PALcode

21164PC Mnemonic	Opcode	Architecture Mnemonic	Function
HW_LD	1B	PAL1B	Performs Dstream load instructions.
HW_ST	1F	PAL1F	Performs Dstream store instructions.
HW_REI	1E	PAL1E	Returns instruction flow to the program counter (PC) pointed to by EXC_ ADDR IPR.
HW_MFPR	19	PAL19	Accesses the IDU, MTU, and Dcache internal processor registers (IPRs).
HW_MTPR	1D	PAL1D	Accesses the IDU, MTU, and Dcache IPRs.

These instructions produce an OPCDEC exception if executed while not in the PALmode environment. If ICSR[HWE] is set, these instructions can be executed in kernel mode. Any software executing with ICSR[HWE] set must use extreme care to obey all restrictions listed in this chapter and in Chapter 5.

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

Note: Explicit software timing is required for accessing the hardware-specific IPRs and the PAL_TEMP registers. These constraints are described in Table 5–31.

6.6.1 HW_LD Instruction

PALcode uses the HW_LD instruction to access memory outside of the realm of normal Alpha memory management and to do special forms of Dstream loads. Figure 6–1 and Table 6–4 describe the format and fields of the HW_LD instruction. Data alignment traps are inhibited for HW_LD instructions.

21164PC Implementation of the Architecturally Reserved Opcodes

Figure 6–1 HW_LD Instruction Format

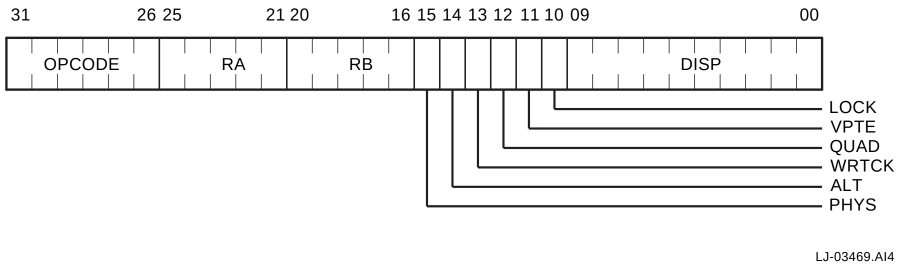


Table 6–4 HW_LD Format Description

Field	Value	Description
OPCODE	$1B_{16}$	The OPCODE field contains $1B_{16}$.
RA	—	Destination register number.
RB	—	Base register for memory address.
PHYS	0	The effective address for the HW_LD is virtual.
	1	The effective address for the HW_LD is physical. Translation and memory-management access checks are inhibited.
ALT	0	Memory-management checks use MTU IPR DTB_CM for access checks.
	1	Memory-management checks use MTU IPR ALT_MODE for access checks.
WRTCK	0	Memory-management checks fault on read (FOR) and read access violations.
	1	Memory-management checks FOR, fault on write (FOW), read, and write access violations.
QUAD	0	Length is longword.
	1	Length is quadword.
VPTE	1	Flags a virtual PTE fetch. Used by trap logic to distinguish single TBMISS from double TBMISS. Access checks are performed in kernel mode.
LOCK	1	Load lock version of HW_LD. PAL must slot to E0 pipe.
DISP	—	Holds a 10-bit signed byte displacement.

6.6.2 HW_ST Instruction

PALcode uses the HW_ST instruction to access memory outside of the realm of normal Alpha memory management and to do special forms of Dstream store instructions. Figure 6–2 and Table 6–5 describe the format and fields of the HW_ST instruction. Data alignment traps are inhibited for HW_ST instructions. The IDU logic will always slot HW_ST to pipe E0.

Figure 6–2 HW_ST Instruction Format

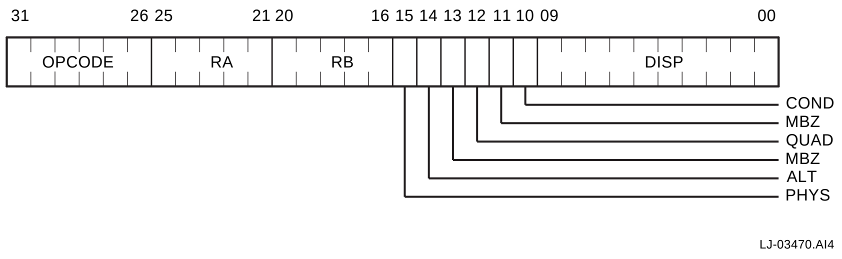


Table 6–5 HW_ST Format Description

Field	Value	Description
OPCODE	$1F_{16}$	The OPCODE field contains $1F_{16}$.
RA	—	Write data register number.
RB	—	Base register for memory address.
PHYS	0 1	The effective address for the HW_ST is virtual. The effective address for the HW_ST is physical. Translation and memory-management access checks are inhibited.
ALT	0 1	Memory-management checks use MTU IPR DTB_CM for access checks. Memory-management checks use MTU IPR ALT_MODE for access checks.
QUAD	0 1	Length is longword. Length is quadword.
COND	1	Store_conditional version of HW_ST. In this case, RA is written with the value of LOCK_ FLAG.
DISP	—	Holds a 10-bit signed byte displacement.
MBZ	—	HW_ST[13,11] must be zero.

6.6.3 HW_REI Instruction

The HW_REI instruction is used to return instruction flow to the PC pointed to by the EXC_ADDR IPR. The value in EXC_ADDR[0] will be used as the new value of PALmode after the HW_REI instruction.

The IDU uses the return prediction stack to speed the execution of HW_REI. There are two different types of HW_REI:

- Prefetch: In this case, the IDU begins fetching the new Istream as soon as possible. This is the version of HW_REI that is normally used.
- Stall prefetch: This encoding of HW_REI inhibits Istream fetch until the HW_REI itself is issued. Thus, this is the method used to synchronize IDU changes (such as ITB write instructions) with the HW_REI. There is a rule that PALcode can have only one such HW_REI in an aligned block of four instructions.

Figure 6–3 and Table 6–6 describe the format and fields of the HW_ REI instruction. The IDU logic will slot HW_REI to pipe E1.

Figure 6–3 HW_REI Instruction Format

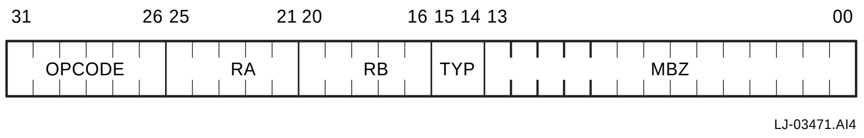


Table 6–6 HW_REI Format Description

Fields	Value	Description
OPCODE	$1E_{16}$	The OPCODE field contains $1E_{16}$.
RA/RB	—	Register numbers; should be R31 to avoid unnecessary stalls.
TYP	10 11	Normal version. Stall version.
MBZ	0	HW_REI[13:00] must be zero.

6.6.4 HW_MFPR and HW_MTPR Instructions

The HW_MFPR and HW_MTPR instructions are used to access internal state from the IDU, MTU, and Dcache. The HW_MFPR from IDU IPRs has a latency of one cycle (HW_MFPR in cycle *n* results in data available to the using instruction in cycle *n*+1). HW_MFPR from MTU and Dcache IPRs has a latency of two cycles. IDU hardware slots each type of MXPR to the correct IEU pipe (refer to Table 5–1).

Figure 6–4 and Table 6–7 describe the format and fields of the HW_MFPR and HW_MTPR instructions.

Figure 6–4 HW_MFPR and HW_MTPR Instruction Format

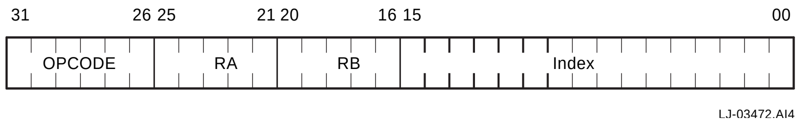


Table 6–7 HW_MFPR and HW_MTPR Format Description

Field	Value	Description
OPCODE	19_{16} $1D_{16}$	The OPCODE field contains 19_{16} for HW_MFPR. The OPCODE field contains $1D_{16}$ for HW_MTPR.
RA/RB	—	Must be the same; source register for HW_MTPR and destination register for HW_MFPR.
Index	—	Specifies the IPR. Refer to Table 5–1 for field encoding. Refer to Chapter 5 for more details about specific IPRs.

7 Initialization and Configuration

This chapter provides information on 21164PC-specific microprocessor/system initialization and configuration. It is organized as follows:

- Input signals **sys_reset_l** and **dc_ok_h** and booting
- **sysclk** ratio and delay
- Built-in self-test (BiSt)
- Serial read-only memory (SROM) interface port
- Serial terminal port
- Cache initialization
- External interface initialization
- Internal processor register (IPR) reset state
- Timeout reset
- IEEE 1149.1 test port reset

7.1 Input Signals sys_reset_l and dc_ok_h and Booting

The 21164PC reset sequence uses two input signals: **sys_reset_l** and **dc_ok_h**. When transitioning from a powered-down state to a powered-up state, signal **dc_ok_h** must be deasserted, and signal **sys_reset_l** must be asserted until power has reached the proper operating point and the input clock to the 21164PC is stable. If the input clock is derived from a PLL, it may take many milliseconds for the input oscillator to start and the PLL output to stabilize.

Input Signals sys_reset_l and dc_ok_h and Booting

After power has reached the proper operating point, signal **dc_ok_h** must be asserted. Then, signal **sys_reset_l** must be deasserted. At this point, the 21164PC recognizes a powered-up state. If signal **dc_ok_h** is not asserted, signal **sys_reset_l** is forced asserted internally. After **sys_reset_l** is deasserted, the 21164PC begins the following sequence of operations:

1. Icache built-in self-test (BiSt).
2. An optional automatic Icache initialization, using an external serial ROM (SROM) interface.
3. Dispatch to the reset PALcode trap entry point (physical location 0).
   a. If step 2 initialized the Icache by using the SROM interface, the cache should contain code that appears to be at location 0, that is, the cache should be initialized such that it hits on the dispatch. Typically the code in the Icache should configure the IPRs in the 21164PC as necessary before causing any offchip read or write commands. This allows the 21164PC to be configured to match the external system implementation.
   b. If step 2 did not initialize the Icache, the Icache has been flushed by reset. The reset PALcode trap dispatch misses in the Icache and produces an offchip read command. The external system implementation must be compatible with the default configuration of the 21164PC after reset (refer to Section 7.8). The code that is executed at this point should complete the 21164PC configuration as necessary.
4. After configuring the 21164PC, control can be transferred to code anywhere in memory, including the noncacheable regions. If the SROM interface was used to initialize the Icache, the Icache can be flushed by a write operation to IC_FLUSH_CTL after control is transferred. This transfer of control should be to addresses not loaded in the Icache by the SROM interface or the Icache may provide unexpected instructions.
5. Typically, PALbase and any state required by PALcode are initialized and the console is started (switching out of PALmode and into native mode). The console code initializes and configures the system and boots an operating system from an I/O device such as a disk or the network.

Signal **sys_reset_l** forces the CPU into a known state. Signal **sys_reset_l** must remain asserted while signal **dc_ok_h** is deasserted, and for some period of time after **dc_ok_h** assertion. It should remain asserted for at least 400 internal CPU cycles in length. Then, signal **sys_reset_l** may be deasserted. Signal **sys_reset_l** deassertion need not be synchronous with respect to **sysclk**. Section 7.8 lists the reset state of each IPR.

Input Signals sys_reset_l and dc_ok_h and Booting

Table 7–1 provides the reset state of each external signal pin.

Table 7–1 21164PC Signal Pin Reset State *(Sheet 1 of 3)*

Signal	Reset State
Clocks	
clk_in_h	Must be clocking.
clk_in_l	Must be clocking.
clk_mode_h[1:0]	NA (input).
cpu_clk_out_h	Clock output.
st_clk1_h	Deasserted.
st_clk2_h	Deasserted.
st_clk3_h	Deasserted.
sys_clk_out1_h	Clock output.
sys_clk_out2_h	Clock output.
sys_reset_l	NA (input).
Bcache	
data_h[127:0]	Tristated.
data_adsc_l	Deasserted.
data_adv_l	Deasserted.
data_ram_oe_l	Deasserted.
data_ram_we_l[3:0]	Deasserted.
index_h[21:4]	Unspecified.
lw_parity_h[3:0]	Tristated.
tag_data_h[32:19]	Tristated.
tag_data_par_h	Tristated.
tag_dirty_h	Tristated.
tag_ram_oe_l	Deasserted.
tag_ram_we_l	Deasserted.
tag_valid_h	Tristated.

Input Signals sys_reset_l and dc_ok_h and Booting

Table 7–1 21164PC Signal Pin Reset State *(Sheet 2 of 3)*

Signal	Reset State
System Interface	
addr_h[39:4]	Driven or tristated depending upon **addr_bus_req_h** at most recent **sysclk** edge. If driven, the value is unspecified.
addr_bus_req_h	NA (input).
addr_res_h[1:0]	NOP.
cack_h	Must be deasserted.
cmd_h[3:0]	Driven or tristated depending upon **addr_bus_req_h** at most recent **sysclk** edge. If driven, the command is NOP.
dack_h	Must be deasserted.
data_bus_req_h	NA (input).
fill_h	Must be deasserted.
fill_dirty_h	NA (input).
fill_error_h	Must be deasserted.
fill_id_h	Must be deasserted.
idle_bc_h	Must be deasserted.
int4_valid_h[3:0]	Unspecified.
victim_pending_h	Unspecified.
Interrupts	
irq_h[3:0]	**sysclk** divisor ratio input.
mch_hlt_irq_h	**sysclk** delay input.
pwr_fail_irq_h	**sysclk** delay input.
sys_mch_chk_irq_h	**sysclk** delay input.
Test Modes	
dc_ok_h	NA (input).
port_mode_h[1:0]	NA (input).
srom_clk_h	Deasserted.

Table 7–1 21164PC Signal Pin Reset State *(Sheet 3 of 3)*

Signal	Reset State
srom_data_h	NA (input).
srom_oe_l	Deasserted.
srom_present_l	NA (input).
tck_h	NA (input).
tdi_h	NA (input).
tdo_h	NA (input).
temp_sense	NA (input).
test_status_h[1]	Deasserted.
tms_h	NA (input).
trst_l	Must be asserted (input).

While signal **dc_ok_h** is deasserted, the 21164PC provides its own internal clock source from an onchip ring oscillator. When **dc_ok_h** is asserted, the 21164PC clock source is the differential clock input pins **clk_in_h** and **clk_in_l**.

When the 21164PC is free-running from the internal ring oscillator, the internal clock frequency is in the range of 10 MHz to 100 MHz (varies from chip to chip). The **sysclk** divisor and **sys_clk_out2_h** delay are determined by input pins while signal **sys_reset_l** remains asserted. Refer to Section 4.2.2 and Section 4.2.3 for ratio and delay values.

7.1.1 Pin State with dc_ok_h Not Asserted

While **dc_ok_h** is deasserted, and **sys_reset_l** is asserted, every output and bidirectional 21164PC pin is tristated and pulled weakly to ground by a small pull-down transistor.

7.2 sysclk Ratio and Delay

While in reset, the 21164PC reads **sysclk** configuration parameters from the interrupt signal pins. These inputs should be driven with the correct configuration values whenever **sys_reset_l** is asserted. Refer to Section 4.2.2 and Section 4.2.3 for relevant input signals and ratio/delay values.

If the signal inputs reflecting configuration parameters change while **sys_reset_l** is asserted, allow 20 internal CPU cycles before the new **sysclk** behavior is correct.

7.3 Built-In Self-Test (BiSt)

Upon deassertion of signal **sys_reset_l**, the 21164PC automatically executes the Icache built-in self-test (BiSt). The Icache is automatically tested and the result is made available in the ICSR IPR and on signal **test_status_h[1]**. Internally, the CPU reset continues to be asserted throughout the BiSt process. For additional information, refer to Section 9.4.4.1.

7.4 Serial Read-Only Memory Interface Port

The serial read-only memory (SROM) interface provides the initialization data load path from a system SROM to the instruction cache (Icache). Following initialization, this interface can function as a diagnostic port using privileged architecture library code (PALcode).

The following signals make up the SROM interface:

srom_present_l
srom_data_h
srom_oe_l
srom_clk_h

During system reset, the 21164PC samples the **srom_present_l** signal for the presence of SROM. If **srom_present_l** is deasserted, the SROM load is disabled and the reset sequence clears the Icache valid bits. This causes the first instruction fetch to miss the Icache and read instructions from offchip memory.

If **srom_present_l** is asserted during setup, then the system performs an SROM load as follows:

1. The **srom_oe_l** signal supplies the output enable to the SROM.
2. The **srom_clk_h** signal supplies the clock to the ROM that causes it to advance to the next bit. The cycle time of this clock is 126± times the CPU clock period.
3. The **srom_data_h** signal inputs the SROM data.

7.4.1 Serial Instruction Cache Load Operation

All Icache bits, including each block's tag, address space number (ASN), address space match (ASM), valid, and branch history bits, can be loaded serially from off-chip serial ROMs. Once the serial load has been invoked by the chip reset sequence, the first 8KB of the Icache is loaded automatically from the lowest to the highest addresses. The second 8KB of the Icache cannot be loaded serially. The tag valid bits for this bank should reflect this.

The automatic serial Icache fill invoked by the chip reset sequence operates internally at a frequency of 126 × CPU clock period. However, due to the synchronization with the system clocks, consecutive access cycles to SROM may shrink or stretch by a system cycle. For example, for a system with a system clock ratio of 15, the time between the two consecutive SROM accesses may be anywhere in the range 111 to 141 CPU cycles. The SROM used in the system must be able to support access times in this range. Refer to Section 9.4.4 for additional SROM timing information.

The serial bits are received in a 256-bit-long fill scan path, from which they are written in parallel into the Icache address. The fill scan path is organized as shown in the text following this paragraph. The farthest bit is shifted in first and the nearest bit is shifted in last. The data and predecode bits in the data array are interleaved. The placeholders are merely for padding the record to a power of 2 and are "don't cares."

```
srom_data_h          serial input -]
BHT Array             0 -]   1 -]  ... -]   7 -]
Data                127 -]  95 -]  126 -]  94 -] ...  -]  96 -]  64 -]
Predecodes           19 -]  14 -]   18 -]  13 -] ...  -]  15 -]  10 -]
Data parity           1 -]   0 -]
Predecodes            9 -]   4 -]    8 -]   3 -] ...  -]   5 -]   0 -]
Data                 63 -]  31 -]   62 -]  30 -] ...  -]  32 -]   0 -]
Tag Parity            b -]
Tag Valids            0 -]   1 -]    2 -]   3 -]
TAG Phy.Address       b -]
TAG ASN               0 -]   1 -]  ... -]   6 -]
TAG ASM               b -]
TAGs                 14 -]  ...  -]  42 -]
Placeholder           0 -]  ...  -]  54
  b = Single bit signal
```

Refer to Appendix C for an example of C code that calculates the predecode values of a serial Icache load.

7.5 Serial Terminal Port

After the SROM data is loaded into the Icache, the three SROM interface signals can be used as a software “UART” and the pins become parallel I/O pins that can drive a diagnostic terminal by using an interface such as RS-232 or RS-423.

7.6 Cache Initialization

Regardless of whether the Icache BiSt is executed, the Icache is flushed during the reset sequence prior to the SROM load. If the SROM load is bypassed, the Icache will be in the flushed state initially.

The data cache (Dcache) is disabled by reset. It is not initialized or flushed by reset. It should be initialized by PALcode before being enabled.

The external board-level Bcache is disabled by reset. It should be initialized by PAL-code before being enabled.

7.6.1 Icache Initialization

The Icache is not kept coherent with memory. When it is necessary to make it coherent with memory, the following procedure is used. The CALL_PAL IMB function performs this function by using this procedure.

1. Execute an MB instruction. This forces all write data in the write buffer into memory.
   - Stall until write buffer is drained.
   - Carry load or issue a HW_MFPR from any MTU IPR.
2. Write to IC_FLUSH_CTL with an HW_MTPR to flush the Icache.
3. Execute a total of 44 NOP instructions (BIS r31,r31,r31) to clear the prefetch buffers and IDU pipeline. The 44 NOP instructions must start on an INT16 boundary. Pad with additional NOP instructions if necessary.

7.6.2 Flushing Dirty Blocks

During a power failure recovery, dirty blocks must be flushed out of the backup cache (Bcache).

To flush out dirty blocks from the Bcache on power failure, the following sequence must be used to guarantee that all the dirty blocks have been written back to main memory:

```
Perform loads at a stride of Bcache block size = 2 × size of the Bcache
```

7.7 External Interface Initialization

After reset, the cache control and bus interface unit (CBU) is in the default configuration dictated by the reset state of the IPR bits that select the configuration options. The CBU response to system commands and internally generated memory accesses is determined by this default configuration. System environments that are not compatible with the default configuration must use the SROM Icache load feature to initially load and execute a PALcode program. This program configures the external interface control (CBU) IPRs as needed.

7.8 Internal Processor Register Reset State

Many IPR bits are not initialized by reset. They are located in error-reporting registers and other IPR states. They must be initialized by initialization PALcode.
Table 7–2 lists the state of all internal processor registers (IPRs) immediately following reset. The table also specifies which registers need to be initialized by power-up PALcode.

Table 7–2 Internal Processor Register Reset State *(Sheet 1 of 3)*

IPR	Reset State	Comments
IDU Registers		
ITB_TAG	UNDEFINED	
ITB_PTE	UNDEFINED	
ITB_ASN	UNDEFINED	PALcode must initialize.
ITB_PTE_TEMP	UNDEFINED	
ITB_IAP	UNDEFINED	
ITB_IA	UNDEFINED	PALcode must initialize.
ITB_IS	UNDEFINED	
IFAULT_VA_FORM	UNDEFINED	
IVPTBR	UNDEFINED	PALcode must initialize.
ICPERR_STAT	UNDEFINED	PALcode must initialize.
IC_FLUSH_CTL	UNDEFINED	
EXC_ADDR	UNDEFINED	
EXC_SUM	UNDEFINED	PALcode must clear exception summary and exception register write mask by writing EXC_SUM.
EXC_MASK	UNDEFINED	
PAL_BASE	Cleared	Cleared on reset.
ICM	UNDEFINED	PALcode must set current mode.

Table 7–2 Internal Processor Register Reset State *(Sheet 2 of 3)*

IPR	Reset State	Comments
ICSR	See Comments	All bits are cleared on reset, except ICSR[37,31,18], which are set; ICSR[44,38], which are UNDEFINED; and ICSR[43:40], which are defined by the state of the interrupt pins during reset.
IPLR	UNDEFINED	PALcode must initialize.
INTID	UNDEFINED	
ASTRR	UNDEFINED	PALcode must initialize.
ASTER	UNDEFINED	PALcode must initialize.
SIRR	UNDEFINED	PALcode must initialize.
HWINT_CLR	UNDEFINED	PALcode must initialize.
ISR	UNDEFINED	
SL_XMIT	Cleared	Appears on external pin.
SL_RCV	UNDEFINED	
PMCTR	See Comments	PMCTR[15:10] are cleared on reset. All other bits are UNDEFINED.
MTU Registers		
DTB_ASN	UNDEFINED	PALcode must initialize.
DTB_CM	UNDEFINED	PALcode must initialize.
DTB_TAG	Cleared	Valid bits are cleared on chip reset but not on timeout reset.
DTB_PTE	UNDEFINED	
DTB_PTE_TEMP	UNDEFINED	
MM_STAT	UNDEFINED	Must be unlocked by PALcode by reading VA register.
VA	UNDEFINED	Must be unlocked by PALcode by reading VA register.
VA_FORM	UNDEFINED	Must be unlocked by PALcode by reading VA register.

Internal Processor Register Reset State

Table 7–2 Internal Processor Register Reset State *(Sheet 3 of 3)*

IPR	Reset State	Comments
MVPTBR	UNDEFINED	PALcode must initialize.
DC_PERR_STAT	UNDEFINED	PALcode must initialize.
DTB_IAP	UNDEFINED	
DTB_IA	UNDEFINED	
DTB_IS	UNDEFINED	
MCSR	Cleared	Cleared on chip reset but not on timeout reset.
DC_MODE	Cleared	Cleared on chip reset but not on timeout reset.
MAF_MODE	Cleared	Cleared on chip reset. MAF_MODE[05] cleared on timeout reset.
DC_FLUSH	UNDEFINED	PALcode must write this register to clear Dcache valid bits.
ALT_MODE	UNDEFINED	
CC	UNDEFINED	CC is disabled on chip reset.
CC_CTL	UNDEFINED	
DC_TEST_CTL	[15] cleared	Cleared on chip reset but not on timeout reset.
DC_TEST_TAG	UNDEFINED	
DC_TEST_TAG_TEMP	UNDEFINED	
CBU Registers		
CBOX_CONFIG		See Section 5.3.1 for power-up state.
CBOX_CONFIG2		See Section 5.3.4 for power-up state.
CBOX_ADDR		See Section 5.3.2 for power-up state.
CBOX_STATUS		See Section 5.3.3 for power-up state.

7.9 Timeout Reset

The instruction fetch/decode unit and branch unit (IDU) contains a timer that times out when a very long period of time passes with no instruction completing. When this timeout occurs, an internal reset event occurs. This clears sufficient internal state to allow the CPU to begin executing again. Registers, IPRs (except as noted in Table 7–2), and caches are not affected. Dispatch to the PALcode MCHK trap entry point occurs immediately.

7.10 IEEE 1149.1 Test Port Reset

Signal **trst_l** must be asserted when **sys_reset_l** is asserted or when **dc_ok_h** is deasserted. Continuous **trst_l** assertion during normal operation is used to guarantee that the IEEE 1149.1 test port does not affect 21164PC operation.

8
Error Detection and Error Handling

This chapter provides an overview of the error handling strategy of the 21164PC. It is organized as follows:

- Error flows
- MCHK flow
- MCK_INTERRUPT flow

Each internal cache (instruction cache [Icache] and data cache [Dcache]) implements parity protection for tag and data. Longword parity protection is implemented for memory and backup cache (Bcache) data. Bcache tag and control (valid and dirty bits) are parity protected. The instruction fetch/decode unit and branch unit (IDU) implements logic that detects when no progress has been made for a very long time and forces a machine check trap.

PALcode handles all error traps (machine checks and parity error interrupts). Where possible, the address of affected data is latched in an onchip IPR. Most of the Istream errors can be retried by the operating system because the machine check occurs before any part of the instruction causing the error is executed. In some other cases, the system may be able to recover from an error by terminating all processes that had access to the affected memory location.

8.1 Error Flows

The following flows describe the events that take place during an error, the recommended responses necessary to determine the source of the error, and the suggested actions to resolve them.

8.1.1 Icache Data or Tag Parity Error

- Machine check occurs before the instruction causing the parity error is executed.

- EXC_ADDR contains either the PC of the instruction that caused the parity error or that of an earlier trapping instruction.
- ICPERR_STAT[TPE] or [DPE] is set.
- Can be retried.

Note: The Icache is not flushed by hardware in this event. If an Icache parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

- **Recommendation**: Flush the Icache early in the MCHK routine.

8.1.2 Dcache Data Parity Error

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred.
- DCPERR_STAT: [DP0] or [DP1] is set. [LOCK] is set. [SEO] is set if there are multiple errors.

Note: For multiple parity errors in the same cycle, the [SEO] bit is not set, but more than one error bit will be set.

- VA: Contains the virtual address of the quadword with the error.
- MM_STAT locked. Contents contain information about instruction causing parity error.

Note: Fault information on another instruction in same cycle may be lost.

8.1.3 Dcache Tag Parity Error

- Machine check occurs. Machine state may have changed.
- DCPERR_STAT: [TP0] or [TP1] is set. [LOCK] is set. [SEO] is set if there are multiple errors.

Note: For multiple parity errors in the same cycle, the [SEO] bit is not set, but more than one error bit will be set.

- VA: Contains the virtual address of the Dcache block (hexword) with the error.

- MM_STAT locked. Contents contain information about instruction causing parity error. [WR] bit is set if error occurred on a store instruction.

Note: Fault information on another instruction in the same cycle may be lost.

- Probably will not be able to recover by deleting a single process, because exact address is unknown, and a load may have falsely hit.

8.1.4 Istream Data Parity Errors (Bcache or Memory)

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Can be retried if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- CBOX_STATUS: [DATA_PAR_ERR[3:0]] is set; [MULTI_ERR] is set if there are multiple errors.
- CBOX_STATUS: [MEMORY] is set if source of fill data is memory/system; is clear if source is Bcache.
- CBOX_ADDR: Contains the physical address bits [39:04] of the octaword associated with the error.

Note: If the Istream parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

- **Recommendation**: On data parity errors, it may be feasible for the operating system to “flush” the block of data out of the Bcache by requesting a block of data with the same Bcache index, but a different tag. If the requested block is loaded with no problems, then the “bad data” has been replaced. If the “bad data” is marked dirty, then when the new data tries to replace the old data, another parity error may result during the write-back (this is a reason not to attempt this in PALcode, because a MCHK from PALcode is always fatal).

8.1.5 Dstream Data Parity Errors (Bcache or Memory)

- Machine check occurs. Machine state may have changed.

- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred.
- CBOX_STATUS: [DATA_PAR_ERR[3:0]] is set; [MULTI_ERR] is set if there are multiple errors.
- CBOX_STATUS: [MEMORY] is set if source of fill data is memory/system; is clear if source is Bcache.
- CBOX_ADDR: Contains the physical address bits [39:04] of the octaword associated with the error.

8.1.6 Bcache Tag Parity Errors—Istream

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache refill buffer and validated.
- Can be retried if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- CBOX_STATUS: [TAG_PAR_ERR] is set; [MULTI_ERR] is set if there are multiple errors.
- CBOX_ADDR: Contains the physical address bits [39:04] of the octaword associated with the error.

Note: The Bcache hit is determined based on the tag alone, not the parity bit. The victim is processed according to the status bits in the tag, ignoring the control field parity. PALcode can distinguish fatal from nonfatal occurrences by checking for the case in which a potentially dirty block is replaced without the victim being properly written back and the case of false hit when the tag parity is incorrect.

8.1.7 Bcache Tag Parity Errors—Dstream

- Machine check occurs. Machine state may have changed.
- Cannot be retried, but may only need to delete the process if data is confined to a single process and no second error occurred. The Bcache hit is determined based on the tag alone, not the parity bit. The victim is processed according to the status bits in the tag, ignoring the control field parity. PALcode can distinguish fatal

from nonfatal occurrences by checking for the case in which a potentially dirty block is replaced without the victim being properly written back and the case of false hit when the tag parity is incorrect.

- CBOX_STATUS: [TAG_PAR_ERR] is set; [MULTI_ERR] is set if there are multiple errors.
- CBOX_ADDR: Contains the physical address bits [39:04] of the octaword associated with the error.

8.1.8 System Read Operations of the Bcache

The 21164PC does not check the parity on outgoing Bcache data. If it is bad, the receiving processor will detect it.

8.1.9 Fill Timeout (FILL_ERROR_H)

- For systems in which fill timeout can occur, the system environment should detect fill timeout and cleanly terminate the reference to 21164PC. If the system environment expects fill timeout to occur, it should detect them. If it does not expect them (as might be true in small systems with fixed memory access timing), it is likely that the internal IDU timeout will eventually detect a stall if a fill fails to occur. To properly terminate a fill in an error case, the **fill_error_h** pin is asserted for one cycle and the normal fill sequence involving the **fill_h**, **fill_id_h**, and **dack_h** pins is generated by the system environment.
- A **fill_error_h** assertion forces a PALcode trap to the MCHK entry point, but has no other effect.

Note: No internal status is saved to show that this happened. If necessary, systems must save this status, and include read operations of the appropriate status registers in the MCHK PALcode.

8.1.10 System Machine Check

- The 21164PC has a maskable machine check interrupt input pin. It is used by system environments to signal fatal errors that are not directly connected to a read access from the 21164PC. It is masked at IPL 31 and anytime the 21164PC is in PALmode.
- ISR: [MCK] is set.

8.1.11 IDU Timeout

- When the IDU detects a timeout, it causes a PALcode trap to the MCHK entry point.
- Simultaneously, a partial internal reset occurs: most states (except the IPR state) are reset. This should not be depended on by systems in which fill timeouts occur in typical use (such as, operating system or console code probing locations to determine if certain hardware is present). The purpose of this error detection mechanism is to attempt to prevent system hang in order to write a machine check stack frame.
- ICPERR_STAT: [TMR] is set.

8.2 MCHK Flow

The following flow is the recommended IPR access order to determine the source of a machine check.

- Must flush Icache to remove bad data on Istream errors. The Icache refill buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- Read EXC_ADDR.
- If EXC_ADDR=PAL, then halt.
- Issue MB to clear out MTU/CBU before reading CBU registers or issuing DC_FLUSH.
- Flush Dcache to remove bad data on Dstream errors.
- Read ICSR.
- Read ICPERR_STAT.
- Read DCPERR_STAT.
- Read CBOX_ADDR.
- Use an MB instruction to ensure that read operations of CBOX_ADDR occur before subsequent read operations of CBOX_STATUS.
- Read CBOX_STATUS and save (unlocks CBOX_STATUS and CBOX_ADDR).
- Read CBOX_STATUS again to be sure it is unlocked; discard result.

- Check for cases that cannot be retried. If any one of the following are true, then skip retry:
  - CBOX_STATUS[TAG_PAR_ERR]
  - CBOX_STATUS[DATA_PAR_ERR]
  - CBOX_STATUS[MULTI_ERR]
- If none of the previous conditions are true, then there is either an IRD that can be retried or the source of the MCHK is a **fill_error_h**. Add code for query of system status.
- The case can be retried if any one or several of the following are true (and none of the previous conditions were true):
  - CBOX_STATUS[DATA_PAR_ERR]
  - ICPERR_STAT[TPE]
  - ICPERR_STAT[DPE]
- Unlock the following IPRs:
  - ICPERR_STAT (write 0x1800).
  - DCPERR_STAT (write 0x03).
  - VA and CBOX_STATUS are already unlocked.
- Check for arithmetic exceptions:
  - Read EXC_SUM.
  - Check for arithmetic errors and handle according to operating-system-specific requirements.
  - Clear EXC_SUM (unlocks EXC_MASK).
- Report the processor-uncorrectable MCHK according to operating-system-specific requirements.

8.3 MCK_INTERRUPT Flow

- Arrived here through interrupt routine because ISR[MCK] bit is set.
- Report the system-uncorrectable MCHK according to operating-system-specific requirements.

9 Electrical Data

This chapter describes the electrical characteristics of the 21164PC component and its interface pins. It is organized as follows:

- Electrical characteristics
- dc characteristics
- Clocking scheme
- ac characteristics
- Power supply considerations

9.1 Electrical Characteristics

Table 9–1 lists the maximum ratings for the 21164PC and Table 9–2 lists the operating voltages.

Table 9–1 21164PC Absolute Maximum Ratings *(Sheet 1 of 2)*

Characteristics	Ratings
Storage temperature	–55°C to 125°C (–67°F to 257°F)
Junction temperature	15°C to 75°C (59°F to 167°F)
Supply voltage	**Vss** = –0.5 V, **Vddi** = 2.1 V, **Vdd** = 2.625 V

Table 9–1 21164PC Absolute Maximum Ratings *(Sheet 2 of 2)*

Characteristics	Ratings
Signal input or output applied[1]	–0.5 V to 2.625 V
Typical **Vdd** worst case power @ **Vdd** = 2.5 V	
Frequency = 533 MHz	1.5 W
Frequency = 600 MHz	1.5 W
Frequency = 666 MHz	2.5 W
Typical **Vddi** worst case power @ **Vddi** = 2.0 V	
Frequency = 533 MHz	16.5 W
Frequency = 600 MHz	18.5 W
Frequency = 666 MHz	20.5 W

[1] For input signals, the overshoot voltage must not exceed 3.5 V. The undershoot voltage must not exceed –0.8 V. The duty cycle for overshoot and undershoot must not exceed 20%.

Caution: Stress beyond the absolute maximum rating can cause permanent damage to the 21164PC. Exposure to absolute maximum rating conditions for extended periods of time can affect the 21164PC reliability.

Table 9–2 Operating Voltages

Nominal		Maximum		Minimum	
Vdd	**Vddi**	**Vdd**	**Vddi**	**Vdd**	**Vddi**
2.5 V	2.05 V	2.625 V	2.1 V	2.375 V	2.0 V

9.2 DC Characteristics

The 21164PC is designed to run in a 2.5-V CMOS environment. The 21164PC is tested and characterized in a CMOS environment.

9.2.1 Power Supply

The **Vss** pins are connected to 0.0 V, the **Vddi** pins are connected to 2.05 V ±0.05 V, the **Vdd** pins are connected to 2.5 V ±5%, and the **Vddref** pins are connected to **Vdd**/2 (1.25 V ±5%).

The **Vdd_PLL** pin is connected to 3.3 V ±5%.

9.2.2 Input Signal Pins

Nearly all input signals are ordinary CMOS inputs sampled with respect to **Vddref** (see Table 9–3). (See Section 9.3.1 for a description of an exception—**clk_in_h** and **clk_in_l**.)

After power has been applied, input and bidirectional pins can be driven to a maximum dc voltage of **Vclamp** at a maximum current of **Iclamp** without harming the 21164PC. Refer to Table 9–3 for **Vclamp** and **Iclamp** values. Inputs greater than **Vclamp** will be clamped to **Vclamp** provided that the current does not exceed **Iclamp**. The 21164PC may be damaged if the voltage exceeds **Vclamp** or the current exceeds **Iclamp**.

9.2.3 Output Signal Pins

Output pins are ordinary 2.5-V CMOS outputs. Although output signals are rail-to-rail, timing is specified to **Vdd**/2.

Note: The 21164PC microprocessor chips do not have an onchip resistor for an output driver.

Bidirectional pins are either input or output pins, depending on control timing. When functioning as output pins, they are ordinary 2.5-V CMOS outputs.

Table 9–3 shows the CMOS dc input and output pins.

Table 9–3 CMOS DC Input/Output Characteristics *(Sheet 1 of 2)*

Parameter		Requirements			
Symbol	**Description**	**Min.**	**Max.**	**Units**	**Test Conditions**
Vih	High-level input voltage	1.7	—	V	—
Vil	Low-level input voltage	—	0.7	V	—
Voh	High-level output voltage	2.2	—	V	**Ioh** = –6.0 mA
Vol	Low-level output voltage	—	0.4	V	**Iol** = 6.0 mA
Iil_pd	Input with pull-down leakage current	—	±50	µA	**Vin** = 0 V
Iih_pd	Input with pull-down current	—	250	µA	**Vin** = 2.4 V
Iil_pu	Input with pull-up current	—	–800	µA	**Vin** = 0.4 V

DC Characteristics

Table 9–3 CMOS DC Input/Output Characteristics *(Sheet 2 of 2)*

Parameter		Requirements			
Symbol	**Description**	**Min.**	**Max.**	**Units**	**Test Conditions**
Iih_pu	Input with pull-up leakage current	—	±50	µA	**Vin** = **Vdd** V
Iozl_pd	Output with pull-down leakage current (tristate)	—	±100	µA	**Vin** = 0 V
Iozh_pd	Output with pull-down current (tristate)	—	250	µA	**Vin** = 2.4 V
Iozl_pu	Output with pull-up current (tristate)	—	–800	µA	**Vin** = 0.4 V
Iozh_pu	Output with pull-up leakage current (tristate)	—	±100	µA	**Vin** = **Vdd** V
Vclamp	Maximum clamping voltage	—	**Vdd** + 1.0	V	**Iclamp** = 100 mA
Idd	Peak power supply current for **Vdd** power supply	—	0.63[1]	A	**Vdd** = 2.625 V Frequency = 533 MHz
Idd	Peak power supply current for **Vdd** power supply	—	0.63[1]	A	**Vdd** = 2.625 V Frequency = 600 MHz
Idd	Peak power supply current for **Vdd** power supply	—	1.05[1]	A	**Vdd** = 2.625 V Frequency = 666 MHz
Iddi	Peak power supply current for **Vddi** power supply	—	9.25	A	**Vddi** = 2.1 V Frequency = 533 MHz
Iddi	Peak power supply current for **Vddi** power supply	—	10.5	A	**Vddi** = 2.1 V Frequency = 600 MHz
Iddi	Peak power supply current for **Vddi** power supply	—	11.75	A	**Vddi** = 2.1 V Frequency = 666 MHz

[1] This assumes a **sysclk** ratio of 4 and worst-case loading of output pins.

Most pins have low current pull-down devices to **Vss**. However, two pins have a pull-up device to **Vdd**. The pull-downs (or pull-ups) are always enabled. This means that some current will flow from the 21164PC (if the pin has a pull-up device) or into the 21164PC (if the pin has a pull-down device) even when the pin is in the high-impedance state. All pins have pull-down devices, except for the pins in the following table:

Signal Name	Notes
tms_h	Has a pull-up device
tdi_h	Has a pull-up device
clk_in_h	50 Ω to **Vterm** (≈ **Vddi**/2) (See Figure 9–1)
clk_in_l	50 Ω to **Vterm** (≈ **Vddi**/2) (See Figure 9–1)
temp_sense	150 Ω to **Vss**

9.3 Clocking Scheme

The differential input clock signals **clk_in_h** and **clk_in_l** run at the internal frequency of the time base for the 21164PC. The output signal **cpu_clk_out_h** toggles with an unspecified propagation delay relative to the transitions on **clk_in_h** and **clk_in_l**.

The 21164PC provides a system clock to run the chip synchronous to the system.

The 21164PC generates and drives out a system clock, **sys_clk_out1_h**. It runs synchronous to the system clock at a selected ratio of the internal clock frequency. There is a small clock skew between the internal clock and **sys_clk_out1_h**.

Refer to Section 4.2 for more information on clock functions.

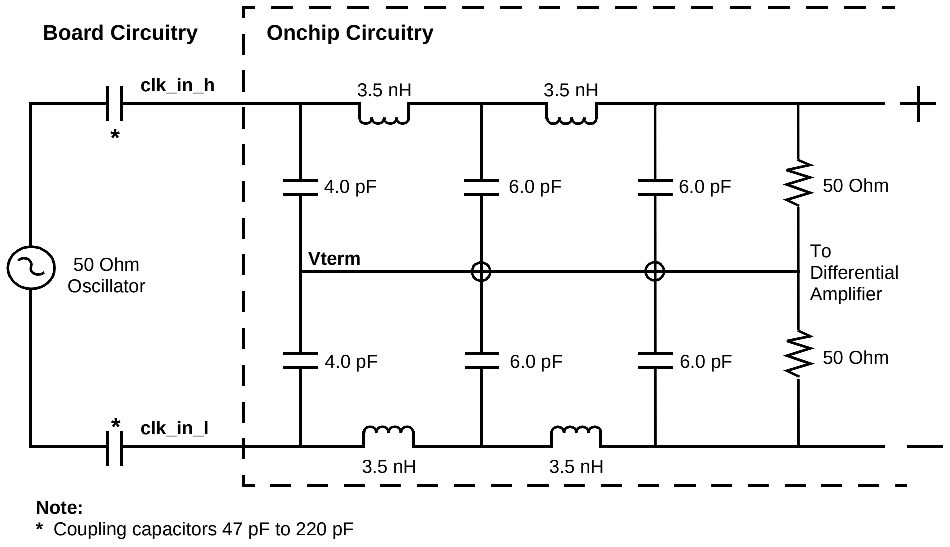
9.3.1 Input Clocks

The differential input clocks **clk_in_h** and **clk_in_l** provide the time base for the chip when **dc_ok_h** is asserted. These pins are self-biasing, and must be capacitively coupled to the clock source on the module.

Note: It is not desirable to drive the **clk_in_h** and **clk_in_l** pins directly.

The terminations on these signals are designed to be compatible with system oscillators of arbitrary dc bias. The oscillator must have a duty cycle of 60%/40% or tighter. Figure 9–1 shows the input network and the schematic equivalent of **clk_in_h** and **clk_in_l** terminations.

Figure 9–1 clk_in_h and clk_in_l Input Network and Terminations



Ring Oscillator

When signal **dc_ok_h** is deasserted, the clock outputs follow the internal ring oscillator. The 21164PC runs off the ring oscillator, just as it would when an external clock is applied. The frequency of the ring oscillator varies from chip to chip within a range of 10 MHz to 100 MHz. This corresponds to an internal CPU clock frequency range of 5 MHz to 50 MHz. The system clock divisor is forced to 8, and the **sys_clk_out2** delay is forced to 3.

Clock Sniffer

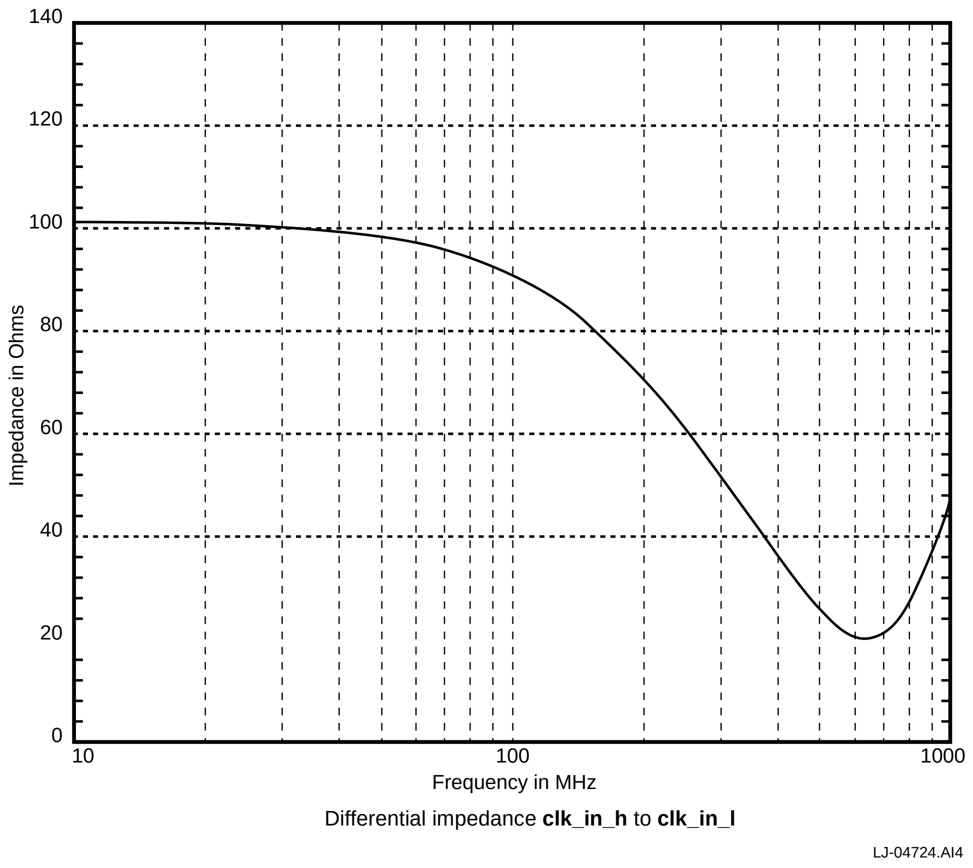
A special onchip circuit monitors the **clk_in** pins and detects when input clocks are not present. When activated, this circuit switches the 21164PC clock generator from the **clk_in** pins to the internal ring oscillator. This happens independently of the state of the **dc_ok_h** pin. The **dc_ok_h** pin functions normally if clocks are present on the **clk_in** pins.

9.3.2 Clock Termination and Impedance Levels

In Figure 9–1, the clock is designed to approximate a 50-Ω termination for the purpose of impedance matching for those systems that drive input clocks across long traces. The clock input pins appear as a 50-Ω series termination resistor connected to a high-impedance voltage source. The voltage source produces a nominal voltage value of **Vddi**/2. The source has an impedance of between 130 Ω and 600 Ω. This voltage is called the self-bias voltage and sources current when the applied voltage at the clock input pins is less than the self-bias voltage. It sinks current when the applied voltage exceeds the self-bias voltage. This high-impedance bias driver allows a clock source of arbitrary dc bias to be ac coupled to the 21164PC. The peak-to-peak amplitude of the clock source must be between 0.6 V and 2.5 V. Either a square-wave or a sinusoidal source may be used. Full-rail clocks may be driven by testers. In any case, the oscillator should be ac coupled to the **clk_in_h** and **clk_in_l** inputs by 47 pF through 220 pF capacitors.

Figure 9–2 shows a plot of the simulated impedance versus the clock input frequency. Figure 9–1 is a simplified circuit of the complex model used to create Figure 9–2.

Figure 9–2 Impedance vs Clock Input Frequency



9.3.3 AC Coupling

Using series coupling (blocking) capacitors renders the 21164PC clock input pins insensitive to the oscillator's dc level. When connected this way, oscillators with any dc offset relative to **Vss** can be used provided they can drive a signal into the **clk_in_h** and **clk_in_l** pins with a peak-to-peak level of at least 600 mV, but no greater than 2.5 V peak-to-peak.

The value of the coupling capacitor is not overly critical. However, it should be sufficiently low impedance at the clock frequency so that the oscillator's output signal (when measured at the **clk_in_h** and **clk_in_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 666 MHz. A high-quality dielectric such as NPO is required to avoid dielectric losses.

Table 9–4 shows the input clock specification.

Table 9–4 Input Clock Specification

Signal Parameter	Nominal Bin	Unit
clk_in_h and **clk_in_l** symmetry	50 ± 10	%
clk_in_h and **clk_in_l** minimum voltage	0.6	V (peak-to-peak)
clk_in_h and **clk_in_l** Z input	50	Ω

9.4 AC Characteristics

This section describes the ac timing specifications for the 21164PC.

9.4.1 Test Configuration

All input signal timing is specified with respect to the internal CPU clock as shown in Figure 9–3. The input signals should meet the following specifications.

1. Minimum slew rate of 1 V/ns.
2. Minimum input signal level:

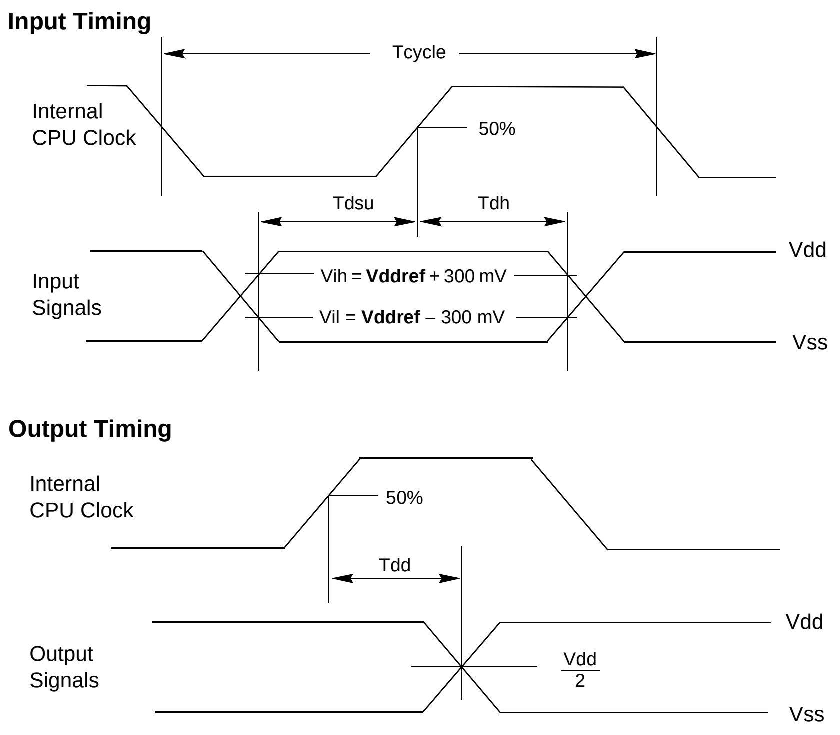
   Vih = **Vddref** + 300 mV

   Vil = **Vddref** – 300 mV

Note: During JTAG test mode only, TTL input levels of Vil = 0.7 V and Vih = 1.7 V are required for proper operation of Boundary Scan Logic.

AC Characteristics

Figure 9–3 Input/Output Pin Timing



Because the speed and complexity of microprocessors has increased substantially over the years, it is necessary to change the way they are tested. Traditional assumptions that all loads can be lumped into some accumulation of capacitance cannot be employed any more. Rather, the model of a transmission line with discrete loads is a much more realistic approach for current test technology.

Typically, printed circuit board (PCB) etch has a characteristic impedance of approximately 75 Ω. This may vary from 60 Ω to 90 Ω with tolerances. If the line is driven in the electrical center, the load could be as low as 30 Ω. Therefore, a characteristic impedance range of 30 Ω to 90 Ω could be experienced.

The 21164PC output drivers are designed with typical printed circuit board applications in mind rather than trying to accommodate a 40-pF test load specification. As such, it “launches” a voltage step into a characteristic impedance, ranging from 30 Ω to 90 Ω.

There is no source termination resistor in the 21164PC fabricated in 0.28-µm CMOS process technology. The source impedance of the driver is approximately 32 Ω ±17. The circuit is designed to deliver a CMOS-level signal as specified under worst-case conditions. Under light load, high drive voltages, and fast process conditions there may be considerable overdrive. It may be necessary to install termination or clamping elements to the signal etches or loads.

9.4.2 Pin Timing

The following sections describe Bcache loop timing and the two system clock modes, namely the **sys_clk_out**-based and the **ref_clk_in**-based system timing models.

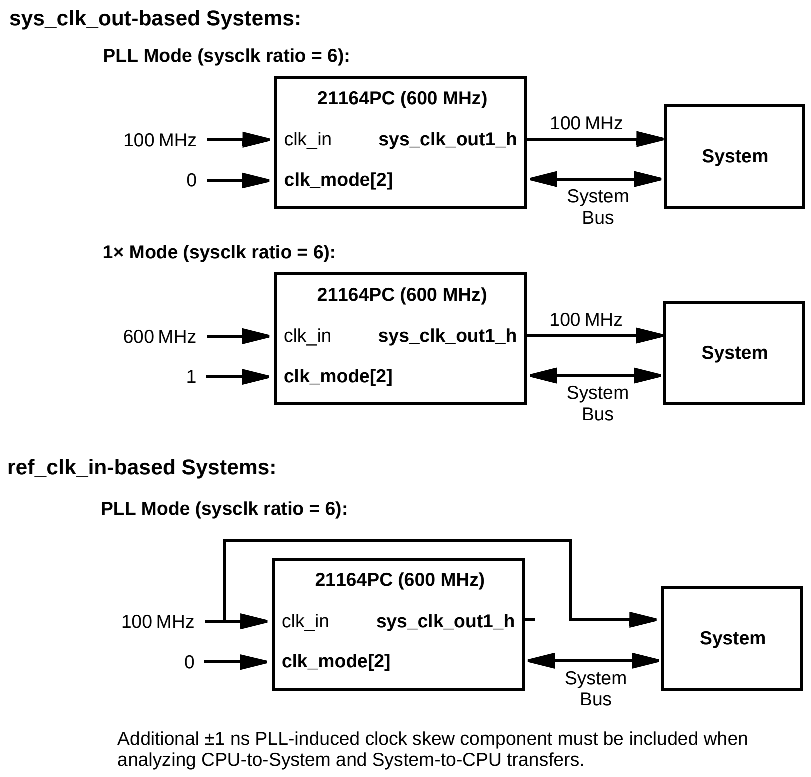
The 21164PC supports several clock modes (see Section 9.4.5). When **clk_mode_h[2]** = 1, the internal PLL is disabled, and the **clk_in_h**/**clk_in_l** input clock is used to generate the internal clock directly. However, when **clk_mode_h[2]** = 0, the internal PLL is enabled and the **clk_in_h**/**clk_in_l** input frequency is driven with the system bus frequency. The 21164PC onchip PLL frequency multiplies **clk_in_h** and **clk_in_l** to generate the internal CPU clock. When the internal PLL is enabled, the maximum PLL-induced clock skew from **clk_in_h** and **clk_in_l** to **sys_clk_out1_h** is ±1 ns.

The system designer may choose between several global system clock distribution schemes. The one selected directly impacts the system ac timing analysis.

You may choose:

- A **sys_clk_out**-based scheme, which uses the 21164PC **sys_clk_out1_h** as the root of the clock distribution network (see Figure 9–4). In such a scheme, all 21164PC input/output timing is relative to the **sys_clk_out1_h** pin, and the **sys_clk_out**-based ac timing tables can be used directly.
- A **ref_clk_in**-based clock distribution scheme, in which the 21164PC is treated as a leaf node in the overall system bus clocking network (see Figure 9–4). Since all 21164PC input/output timing is relative to the **sys_clk_out1_h** pin, a **ref_clk_in**-based system must account for the additional PLL-induced clock skew (±1 ns) above and beyond what is called out in the **sys_clk_out**-based ac timing tables when analyzing CPU-to-system or system-to-CPU transfers.

Figure 9–4 System Bus Clock Distribution Schemes


sys_clk_out-based Systems:
PLL Mode (sysclk ratio = 6):
21164PC (600 MHz)
100 MHz
clk_in
sys_clk_out1_h
100 MHz
System
0
clk_mode[2]
System Bus
1× Mode (sysclk ratio = 6):
21164PC (600 MHz)
600 MHz
clk_in
sys_clk_out1_h
100 MHz
System
1
clk_mode[2]
System Bus
ref_clk_in-based Systems:
PLL Mode (sysclk ratio = 6):
21164PC (600 MHz)
100 MHz
clk_in
sys_clk_out1_h
System
0
clk_mode[2]
System Bus
Additional ±1 ns PLL-induced clock skew component must be included when analyzing CPU-to-System and System-to-CPU transfers.

9.4.2.1 Backup Cache Loop Timing

The 21164PC must be configured to support an offchip L2 backup cache (Bcache). Private Bcache read or write transactions initiated by the 21164PC are independent of the system clocking scheme. Bcache loop timing must be an integer multiple of the 21164PC cycle time.

Table 9–5 lists the Bcache loop timing.

Table 9–5 Bcache Loop Timing

Signal	Specification	Value	Name
data_h[127:0]	Input setup	0.6 ns	**Tdsu**
data_h[127:0]	Input hold	0.0 ns	**Tdh**
data_h[127:0], index_h[21:4], st_clk1_h[1]**, st_clk2_h**[1]**, st_clk3_h**[1]	Output delay	**Tdd** + 0.2 ns[2]	**Tiod**
data_h[127:0] index_h[21:4], st_clk1_h[1]**, st_clk2_h**[1]**, st_clk3_h**[1]	Output hold time	**Tmdd**	**Tioh**

[1] See Section 4.6 for the positioning of **st_clk1_h**, **st_clk2_h**, and **st_clk3_h** with respect to the Bcache index pins.
[2] The value 0.2 ns accounts for onchip driver and clock skew.

Outgoing Bcache index and data signals are driven off the internal clock edge and the incoming Bcache tag and data signals are latched on the same internal clock edge. Table 9–6 shows the output driver characteristics.

Table 9–6 Normal Output Driver Characteristics

Specification	40-pF Load	10-pF Load	Name
Maximum driver delay	2.6 ns	1.4 ns	**Tdd**
Minimum driver delay	0.8 ns	0.8 ns	**Tmdd**

Output pin timing is specified for lumped 40-pF and 10-pF loads for the normal driver. In some cases, the circuit may have loads higher than 40 pF. The 21164PC can safely drive higher loads provided the average charging or discharging current from each pin is 11 mA or less for normal output drivers. The following equation can be used to determine the maximum capacitance that can be safely driven by each pin:

- For normal output drivers: C_{max} (in pF) = 5t, where t is the waveform period (measured from rising to rising or falling to falling edge), in nanoseconds.

For example, if the waveform appearing on a given normal I/O pin has a 12.0-ns period, it can safely drive up to and including 60 pF.

9.4.2.2 Accumulated CPU Phase Error

When the 21164PC is operating in PLL mode (that is, **clk_mode_h[2:0]** = 00x), the onchip PLL accumulates 50 ps of phase error for each successive CPU cycle. The analysis of the Bcache loop timing must account for the accumulated phase error, since the Bcache timing is based on CPU cycle granularity. The accumulated phase error, however, will never exceed the maximum PLL-induced clock skew (±1 ns), regardless of the number of CPU cycles between signal events.

The Bcache control signals are driven **bc_clk_delay** CPU cycles *before* the **st_clk*x*_h** is driven, and the accumulated phase error (**bc_clk_delay** × 50 ps) must be accounted for when analyzing Bcache control setup timing. In Figure 9–5, the **bc_clk_delay** is two CPU cycles, which means that 100 ps of additional skew uncertainty must be accounted for when analyzing the Bcache setup timing to **st_clk*x*_h**.

When determining the appropriate bc_rd_latency parameter, the number of CPU cycles of accumulated phase error relative to the **st_clk*x*_h** that delivered the data must also be counted. Recall that for reg-flow SSRAMs, the first clock delivers the data, while for reg-reg SSRAMs, the second clock delivers the data.

For example:

Reg-Reg SSRAMs:

Accumulated Phase Error (ns) = (bc_rd_latency – (bc_clk_ratio + bc_clk_delay)) × 50 ps

Example:

bc_rd_latency = 8 CPU cycles

bc_clk_ratio = 3 CPU cycles

bc_clk_delay = 2 CPU cycles

Accumulated Phase Error (ns) = (8 – (3 + 2)) × 50 ps = 150 ps

Reg-Flow SSRAMs:

Accumulated Phase Error (ns) = (bc_rd_latency – bc_clk_delay) × 50 ps

Example:

bc_rd_latency = 8 CPU cycles

bc_clk_ratio = 3 CPU cycles

bc_clk_delay = 2 CPU cycles

Accumulated Phase Error (ns) = (8 – 2) × 50 ps = 300 ps

Figure 9–5 Bcache Read Timing

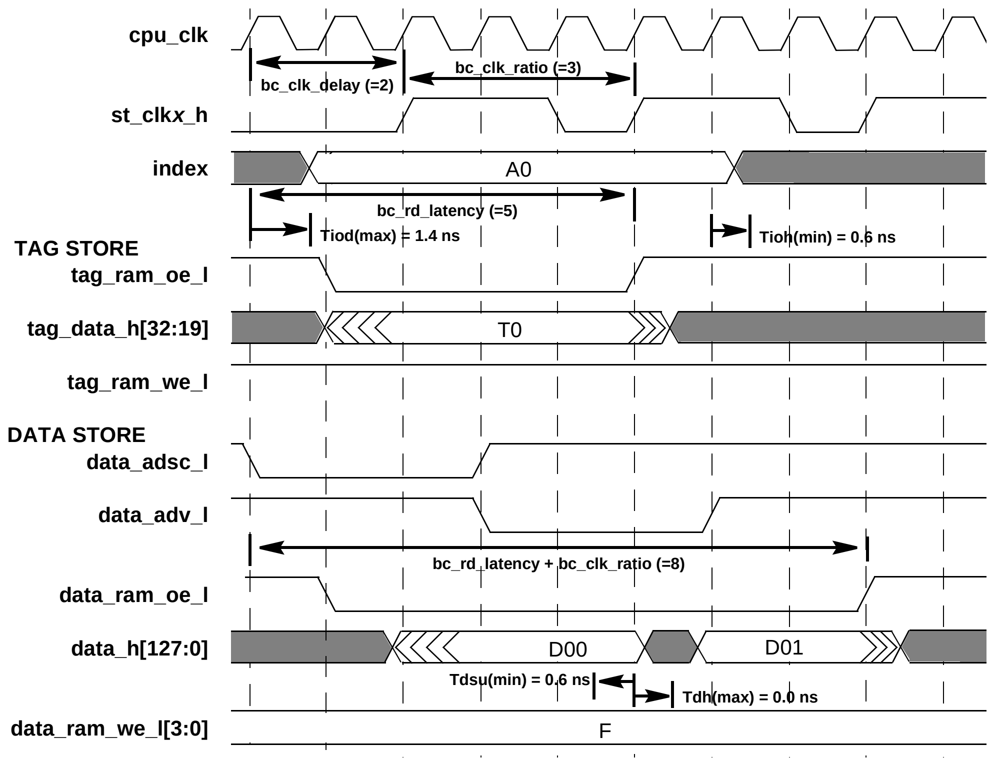
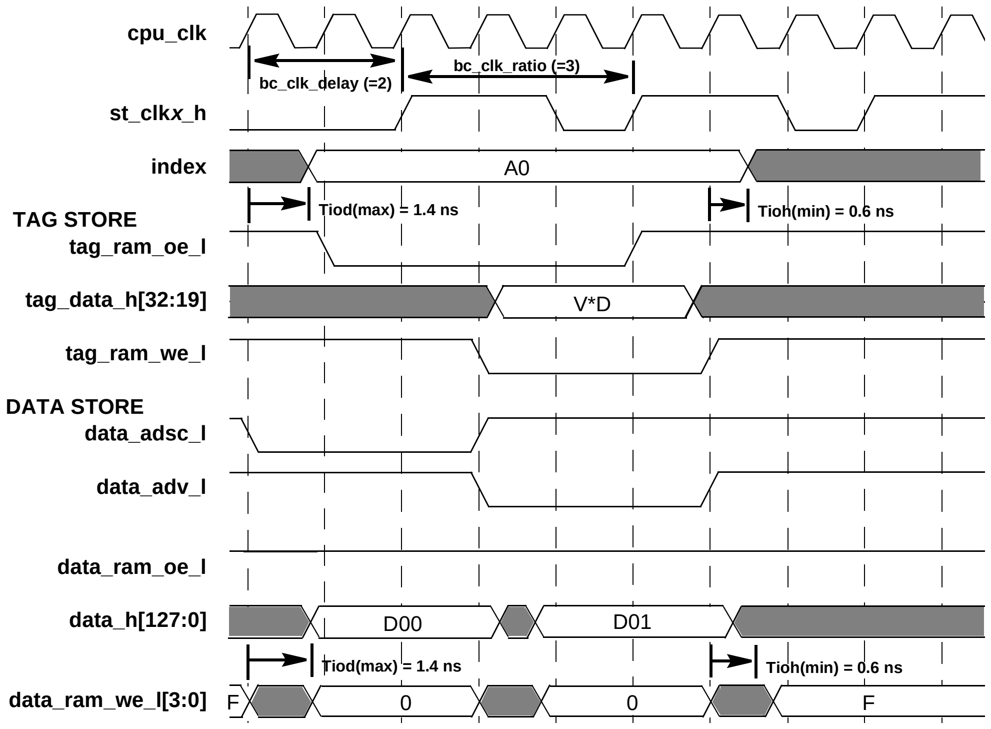


Figure 9–6 shows the timing for a Bcache private data-write hit clean operation.

Figure 9–6 Bcache Private Data-Write Hit Clean



9.4.2.3 sys_clk_out-Based Systems

All timing is specified relative to the rising edge of the internal CPU clock.

Table 9–7 shows 21164PC system clock **sys_clk_out1_h** output timing. Setup and hold times are specified independent of the relative capacitive loading of **sys_clk_out1_h**, **addr_h[39:4]**, **data_h[127:0]**, and **cmd_h[3:0]** signals.

Table 9–7 21164PC System Clock Output Timing (sysclk=$T_ø$) *(Sheet 1 of 2)*

Signal	Specification	Value	Name
sys_clk_out1_h	Output delay	**Tdd**	**Tsysd**
sys_clk_out1_h	Minimum output delay	**Tmdd**	**Tsysdm**
data_bus_ req_h, data_h[127:0], addr_h[39:4]	Input setup	0.6 ns	**Tdsu**
data_bus_ req_h, data_h[127:0], addr_h[39:4]	Input hold	0 ns	**Tdh**

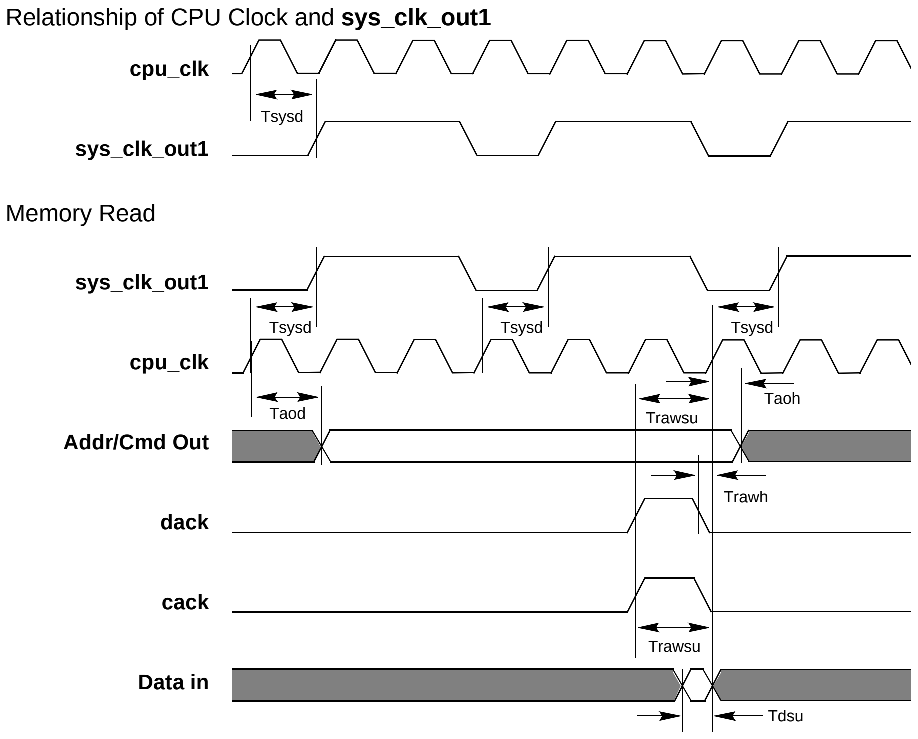
Table 9–7 21164PC System Clock Output Timing (sysclk=$T_ø$) *(Sheet 2 of 2)*

Signal	Specification	Value	Name
addr_h[39:4]	Output delay	**Tdd** + 0.2 ns[1]	**Taod**
addr_h[39:4]	Output hold time	**Tmdd**	**Taoh**
data_h[127:0]	Output delay	**Tdd** [+ **Tcycle**][2] + 0.2 ns[1]	**Tdod**[2]
data_h[127:0]	Output hold time	**Tmdd** [+ **Tcycle**][2]	**Tdoh**[2]
addr_bus_req_h, dack_h, cack_h	Input setup	**Tcycle**	**Trawsu**
addr_bus_req_h, dack_h, cack_h	Input hold	0.0 ns	**Trawh**

[1] The value 0.2 ns accounts for onchip driver and clock skew.
[2] For all system write transactions initiated by the 21164PC, data is driven **Tcycle** (= 1 **cpu_clk**) after the **sys_clk_out1_h** pin. For all private write transactions, data is driven coincident with **Tcycle** (= 0 **cpu_clk**) the driving of **index_h[21:4].**

Figure 9–7 shows **sys_clk** system timing.

Figure 9–7 sys_clk System Timing



9.4.3 Timing—Additional Signals

This section lists timing for all other signals.

Asynchronous Input Signals

The following is a list of the asynchronous input signals:

clk_mode_h[2:0]	**dc_ok_h**	**sys_reset_l**	**irq_h[3:0]**
mch_hlt_irq_h[1]	**pwr_fail_irq_h**[1]	**sys_mch_chk_irq_h**[1]	

[1] These signals can also be used synchronously.

Miscellaneous Signals

Table 9–8 and Table 9–9 list the timing for miscellaneous input-only and output-only signals. All timing is expressed in nanoseconds.

Table 9–8 Input Timing for sys_clk_out-Based Systems

Signal	Specification	Value	Name
fill_h, fill_error_h, fill_id_h, idle_bc_h **irq_h[3:0], mch_hlt_irq_h, pwr_fail_irq_h, sys_mch_chk_irq_h** Testability pins: **port_mode_h, srom_data_h, srom_present_l**	Input setup	0.6 ns	**Tdsu**
fill_h, fill_error_h, fill_id_h, idle_bc_h **irq_h[3:0], mch_hlt_irq_h, pwr_fail_irq_h, sys_mch_chk_irq_h** **sys_reset_l** Testability pins: **port_mode_h, srom_data_h, srom_present_l**	Input hold	0 ns	**Tdh**

Table 9–9 Output Timing for sys_clk_out-Based Systems

Signal	Specification	Value	Name
Unidirectional Signals			
addr_res_h, **int4_valid_h**,[1] **srom_clk_h**, **srom_oe_l**, **victim_pending_h**	Output delay	**Tdd** + 0.2 ns	**Taod**
addr_res_h, **int4_valid_h**,[1] **srom_clk_h**, **srom_oe_l**, **victim_pending_h**	Output hold	**Tmdd**	**Taoh**
int4_valid_h[2]	Output delay	**Tdd** + **Tcycle** + 0.2 ns	**Tdod**
int4_valid_h[2]	Output hold	**Tmdd** + **Tcycle**	**Tdoh**
Bidirectional Signals			
Input mode:			
cmd_h, **lw_parity_h**,[1] **tag_dirty_h**[3]	Input setup	0.6 ns	**Tdsu**
cmd_h, **lw_parity_h**,[1] **tag_dirty_h**[3]	Input hold	0 ns	**Tdh**
Output mode:			
cmd_h, **tag_dirty_h**,[4] **tag_valid_h**[4]	Output delay	**Tdd** + 0.2 ns	**Taod**
lw_parity_h[2]	Output delay	**Tdd** + **Tcycle** + 0.2 ns	**Tdod**
cmd_h, **tag_dirty_h**,[4] **tag_valid_h**[4]	Output hold	**Tmdd**	**Taoh**
lw_parity_h[2]	Output hold	**Tmdd** + **Tcycle**	**Tdoh**

[1] Read transaction
[2] Write transaction
[3] Fills from memory
[4] Only for write broadcasts and system transactions

Signals in Table 9–10 are used to control Bcache data transfers. These signals are driven off the CPU clock. The timing of these signals does not change when switching over to the **sys_clk_out** timing domain.

Table 9–10 Bcache Control Signal Timing

Signal	Specification	Value	Name
Input mode:			
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Input setup	0.6 ns	**Tdsu**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Input hold	0 ns	**Tdh**
Output mode:			
data_ram_oe_l, **data_ram_we_l[3:0]**, **tag_ram_oe_l**, **tag_ram_we_l**	Output delay	**Tbddd** + 0.2 ns[1]	**Taod**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Output delay	**Tdd** + 0.2 ns[1]	**Taod**
data_ram_oe_l, **data_ram_we_l[3:0]**, **tag_ram_oe_l**, **tag_ram_we_l**	Output hold	**Tmdd**	**Taoh**
tag_data_h, **tag_data_par_h**, **tag_valid_h**	Output hold	**Tmdd**	**Taoh**

[1] The value 0.2 ns accounts for onchip driver and clock skew.

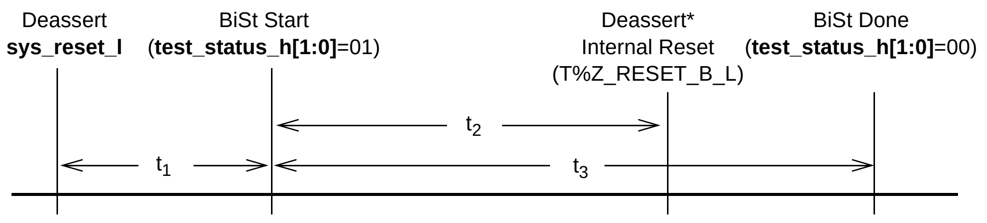
9.4.4 Timing of Test Features

Timing of 21164PC testability features depends on the system clock rate and the test port's operating mode. This section provides timing information that may be needed for most common operations.

9.4.4.1 Icache BiSt Operation Timing

The Icache BiSt is invoked by deasserting the external reset signal **sys_reset_l**. Figure 9–8 shows the timing between various events relevant to BiSt operations.

Figure 9–8 BiSt Timing Event —Timeline



The timing for deassertion of internal reset (time t_2, see asterisk) is valid only if an SROM is not present (indicated by keeping signal **srom_present_l** deasserted). If an SROM is present, the SROM load is performed once the BiSt completes. The internal reset signal T%Z_RESET_B_L is extended until the end of the SROM load (Section 9.4.4.2). In this case, the end of the timeline shown in Figure 9–8 connects to the beginning of the timeline shown in Figure 9–9.

Table 9–11 and Table 9–12 list timing shown in Figure 9–8 for some of the system clock ratios. Time t_1 is measured starting from the rising edge of **sysclk** following the deassertion of the **sys_reset_l** signal.

Table 9–11 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (System Cycles)

	System Cycles		
Sysclk Ratio	t_1	t_2	t_3
4	7	28569 + 3½	28570
15	7	15749 + 14½	15750

Table 9–12 BiSt Timing for Some System Clock Ratios, Port Mode=Normal (CPU Cycles)

	CPU Cycles		
Sysclk Ratio	t_1	t_2	t_3
4	28	114279½	114280
15	105	236249½	236250

9.4.4.2 Automatic SROM Load Timing

The SROM load is triggered by the conclusion of BiSt if **srom_present_l** is asserted. The SROM load occurs at the internal cycle time of approximately 126 CPU cycles for **srom_clk_h**, but the behavior at the pins may shift slightly. Refer to Chapter 7 for more information on input signals, booting, and the SROM interface port.

Timing events are shown in Figure 9–9 and are listed in Table 9–13 and Table 9–14.

Figure 9–9 SROM Load Timing Event—Timeline

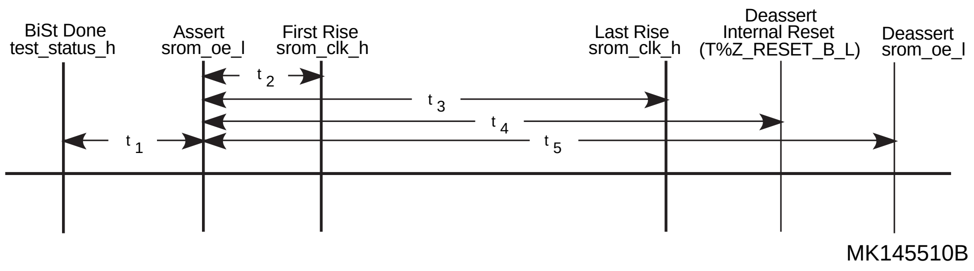


Table 9–13 SROM Load Timing for Some System Clock Ratios (System Cycles)

	System Cycles[1]				
Sysclk Ratio	t_1	t_2	t_3	t_4	t_5
4	3	48	4209267	4209361 + 3½	4209362
15	3	13	1122472	1122496 + 14½	1122497

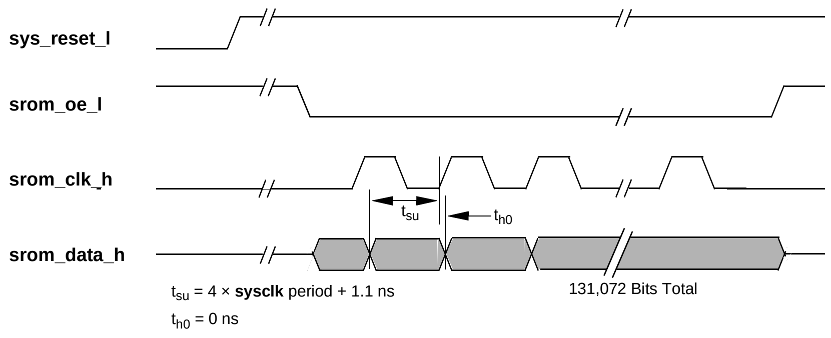
[1] Measured in **sysclk** cycles, where "+ *n*" refers to an additional *n* CPU cycles

Table 9–14 SROM Load Timing for Some System Clock Ratios (CPU Cycles)

	CPU Cycles				
Sysclk Ratio	t_1	t_2	t_3	t_4	t_5
4	12	192	16837068	16837447½	16837448
15	45	195	16837080	16837454½	16837455

Figure 9–10 is a timing diagram of an SROM load sequence.

Figure 9–10 Serial ROM Load Timing



The minimum **srom_clk_h** cycle = (126 **–** **sysclk** ratio) × (CPU cycle time).

The maximum **srom_clk_h** to **srom_data_h** delay allowable (in order to meet the required setup time) = [126 **–** (5 × **sysclk** ratio)] × (CPU cycle time).

9.4.5 Clock Test Modes

This section describes the 21164PC clock test modes.

9.4.5.1 PLL Mode

When **clk_mode_h[2:0] =** 00x, the **clk_in_h**/**clk_in_l** frequency is driven at the system bus frequency. The 21164PC onchip PLL frequency multiplies the **clk_in_h**/ **clk_in_l** to generate the internal CPU clock. The frequency multiplier is determined by the value sampled at the IRQ pins at the deassertion of reset (**sysclk** ratio).

When operating in PLL mode, the maximum clock skew from **clk_in_h**/**clk_in_l** to **sys_clk_out1_h** is ±1 ns and is referred to as PLL-induced clock skew. When you are using a **ref_clk**-based clock distribution scheme, you must add this PLL-induced clock skew to the ac components referenced in **sys_clk_out**-based ac timing tables.

9.4.5.2 Normal (1× Clock) Mode

When **clk_mode_h[2:1]** = 10, the **clk_in_h**/**clk_in_l** frequency is used to drive the input clock frequency. The **clk_mode_h[0]** signal is used to enable/disable a clock equalizing circuit, called a **symmetrator**. The symmetrator equalizes the duty-cycle of the input clock for use onchip. The **clk_in_h**/**clk_in_l** signals must have a duty cycle of at least 60/40 for the symmetrator to work properly.

9.4.5.3 Clock Test Reset Mode

When **clk_mode_h[1]** is asserted, the **sys_clk_out** generator circuit is forced to reset to a known state. This allows the chip manufacturing tester to synchronize the chip to the tester cycle. This mode can be used with the symmetrator either enabled or disabled.

Table 9–15 lists the clock test modes.

Table 9–15 Clock Test Modes

	clk_mode_h			
Mode	**[2]**	**[1]**	**[0]**	**Notes**
Normal (1×) clock mode	1	0	0	
Normal (1×) clock mode	1	0	1	Symmetrator is enabled.
Clock reset	x	1	0	
Clock reset	x	1	1	Symmetrator is enabled.
PLL mode	0	0	x	System bus frequency.

9.4.6 IEEE 1149.1 (JTAG) Performance

Table 9–16 lists the standard mandated performance specifications for the IEEE 1149.1 circuits.

Table 9–16 IEEE 1149.1 Circuit Performance Specifications

Item	Specification
trst_l is asynchronous. Minimum pulse width.	4 ns
trst_l setup time for deassertion before a transition on **tck_h**.	4 ns
Maximum acceptable **tck_h** clock frequency.	16.6 MHz
tdi_h/**tms_h** setup time (referenced to **tck_h** rising edge).	4 ns
tdi_h/**tms_h** hold time (referenced to **tck_h** rising edge).	4 ns
Maximum propagation delay at pin **tdo_h** (referenced to **tck_h** falling edge).	14 ns
Maximum propagation delay at system output pins (referenced to **tck_h** falling edge).	20 ns

9.5 Power Supply Considerations

For correct operation of the 21164PC, all of the **Vss** pins must be connected to ground, all of the **Vdd** pins must be connected to a 2.5-V ±5% power source, and all of the **Vddi** pins must be connected to a 2.0-V ±0.1 V power source. This source voltage should be guaranteed (even under transient conditions) at the 21164PC pins, and not just at the PCB edge.

Plus 5 V is not used in the 21164PC. The voltage difference between the **Vdd** pins and **Vss** pins must never be greater than 2.625 V, and the voltage difference between the **Vddi** pins and **Vss** pins must never be greater than 2.1 V. If the differentials exceed these limits, the 21164PC chip will be damaged.

9.5.1 Decoupling

The effectiveness of decoupling capacitors depends on the amount of inductance placed in series with them. The inductance depends both on the capacitor style (construction) and on the module design. In general, the use of small, high-frequency capacitors placed close to the chip package's power and ground pins with very short module etch will give best results. Depending on the user's power supply and power supply distribution system, bulk decoupling may also be required on the module.

The 21164PC requires two sets of decoupling capacitors: one for **Vdd** and one for **Vddi**.

9.5.1.1 Vdd Decoupling

The amount of decoupling capacitance connected between **Vdd** and **Vss** should be roughly equal to 10 times the amount of capacitive load that 21164PC is required to drive at any one time. This should guarantee a voltage drop of no more than 10% on **Vdd** during heavy drive conditions.

Use capacitors that are as physically small as possible. Connect the capacitors directly to the 21164PC **Vdd** and **Vss** pins by short surface etch (0.64 cm [0.25 in] or less). The small capacitors generally have better electrical characteristics than the larger units and will more readily fit close to the IPGA pin field.

When designing the placement of decoupling capacitors, **Vdd** decoupling capacitors should be favored over **Vddi** decoupling capacitors (that is, **Vdd** capacitors should be placed closer to the 21164PC than the **Vddi** capacitors).

9.5.1.2 Vddi Decoupling

Each individual case must be separately analyzed, but generally designers should plan to use at least 4 µF of capacitance connected between **Vddi** and **Vss**. Typically, 30 to 40 small, high-frequency 0.1-µF capacitors are placed near the chip's **Vddi** and **Vss** pins. Actually placing the capacitors in the pin field is the best approach. Several tens of µF of bulk decoupling (comprised of tantalum and ceramic capacitors) should be positioned near the 21164PC chip.

Use capacitors that are as physically small as possible. Connect the capacitors directly to the 21164PC **Vddi** and **Vss** pins by short surface etch (0.64 cm [0.25 in] or less). The small capacitors generally have better electrical characteristics than the larger units, and will more readily fit close to the IPGA pin field.

9.5.2 Power Supply Sequencing

When applying or removing power to the 21164PC, **Vdd** (the 2.5-V supply voltage) must be no less than **Vddi** (the 2.0-V supply voltage).

The following rules must be followed when either applying or removing the supply voltages:

1. **Vdd** must always be at the same or a higher voltage than **Vddi** during normal operation.
2. The *signal voltage* must not exceed **Vclamp**.
3. The *signal voltage* must not be more than 1.5 V higher than **Vddi** after **Vddi** has reached 2.0 V.

Rule 1 means that either **Vdd** and **Vddi** can be brought up and down in unison or **Vddi** can be applied after and removed before **Vdd**.

Rule 2 means that the signal voltage must not be allowed to exceed **Vclamp** during the application or removal of power. Refer to Table 9–3 for the value of **Vclamp**. Note that it is acceptable for the signal voltage either to be held at zero or to follow **Vdd** during the application or removal of power.

Rule 3 means that, if the signal voltage follows **Vdd**, the signal voltage must never be greater than 1.5 V above the value of **Vddi** after **Vddi** has reached 2.0 V. If the value of **Vddi** is 0 V while the signal voltage is following **Vdd**, the signal voltage must not exceed 2.5 V. This applies equally during the application or the removal of power.

Note that if the signal voltage is held at 0 V during power-up reset (that is, the ASICs and SRAMs are set to drive 0 V during reset), **Vdd** and **Vddi** can be brought up together. In a similar manner, the power-down situation can be managed if the signal voltages are forced to 0 V when the loss of **Vddi** is detected.

During power-up, **Vddi** can momentarily exceed the maximum steady-state value under the following conditions:

- The transient voltage is 200 mV or less.
- The transient period lasts for 200 µs or less.

The transient voltage is defined as the voltage that rises above the maximum-allowed steady-state value. The transient period is defined as the time beginning when the transient voltage exceeds the steady-state value and ending when it falls back to it.

There is no derating for shorter transient periods or lower transient voltages (for example, a 400-mV transient voltage lasting for 100 µs is not acceptable).

All input and bidirectional signals are diode-clamped to **Vdd** and **Vss**. A current greater than **Iclamp** on an individual pin could damage the 21164PC. Designers must take care that currents greater than **Iclamp** will not be achieved during power-supply sequencing. While currents less than **Iclamp** will not damage the 21164PC, other source drivers connected to the 21164PC could be damaged by the clamp. Designers must verify that the source drivers will not be damaged by currents up to **Iclamp**.

10
Power Management

This chapter describes the 21164PC power management.

The 21164PC implements two different power states as defined by the ACPI 1.0 specification: C0 and C3. This capability is only possible when the low-frequency external-clock mode is used: that is, none of these power management capabilities can be used with the high-frequency mode.

Power management is provided by carefully changing the internal 21164PC clock frequency. Since power dissipation is proportionate with operating speed, controlling the clock provides a way to change the power utilization of the 21164PC based on computational needs.

Full operation is maintained in the C0 state. While in the C0 state, the 21164PC allows software to throttle the CPU speed by changing the **sysclk** ratio. The C0 state is maintained as long as the internal operating frequency is set to greater than or equal to four times the external clock frequency.

The C3 state provides a way for users to further reduce the power dissipation on the 21164PC during idle periods. The internal 21164PC clock is reduced to one-eighth or one-sixteenth of full-speed operation in this mode. User instructions cannot be executed in the C3 state.

10.1 Clock Generation and Power Management

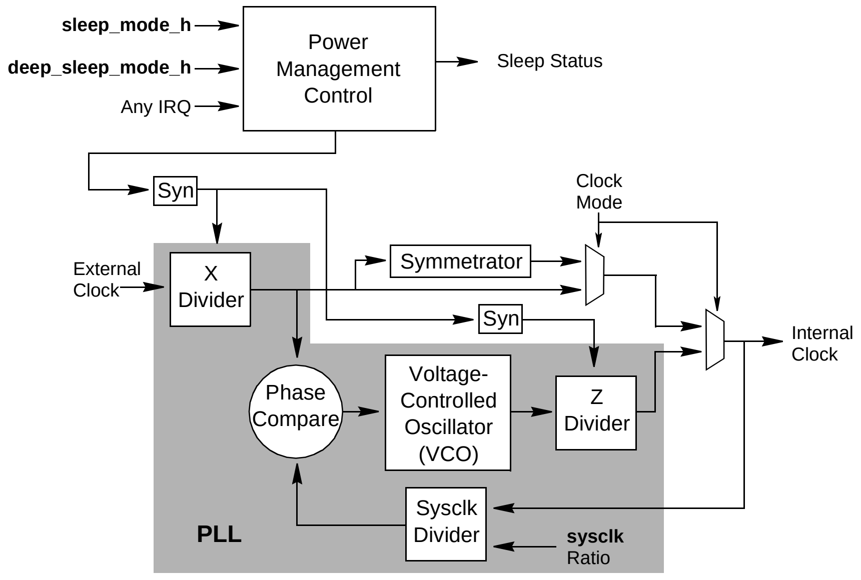
A detailed understanding of how internal clock generation is implemented is necessary in order to understand how power management operates in the 21164PC.

As shown in Figure 10–1, the main components of the clock generation logic consist of a phase-locked loop (PLL) and power management control logic. The PLL consists of a voltage-controlled oscillator (VCO), phase compare logic, and several dividers. The VCO generates the main high-frequency clock that eventually drives the internal 21164PC clock (assuming a low-frequency external clocking is used). In steady state, the phase compare logic controls the VCO in such a way as to keep the internal clock in phase with the external clock. Whenever the value in any of the dividers changes (that is, changing frequency within the C0 state or changing ACPI states), phase-lock is lost between the internal and external clock for a specific period of time.

Because the VCO has a relatively slow response time, it continually operates near the highest operating frequency that will be used in the system, irrespective of the power management state. The power management is provided by carefully changing the divider values in the X, Z, and **sysclk** divider logic in order to change the internal 21164PC clock rate.

Changing the values in the X and Z dividers controls transitions between the C0 and C3 states. These dividers are not directly accessible to software. Instead, they are controlled by the power management control logic in response to setting the SLEEP_MODE and DEEP_SLEEP_MODE bits in the ICSR (start the transition to C3 state), or receiving any external interrupt (start the transition back to C0 state).

Figure 10–1 Power Management Block Diagram



The transition into C3 state is initiated by setting **sleep_mode_h** and, optionally, **deep_sleep_mode_h**. The power management control logic begins a sequence of activities that throttles down the internal clock frequency to either one-eighth the full clock rate (if **deep_sleep_mode_h** is not set) or one-sixteenth the full clock rate (if **deep_sleep_mode_h** is set). The first step in the sequence throttles down the internal frequency to half the full-speed rate for 16,384 (16K) cycles. For reference, at a full frequency of 500 MHz, this provides 30 µs for the power supply to adjust to the partial frequency change. The power management control logic keeps the frequency at the half-speed rate for the full 16K cycles. If an external interrupt occurs during this 30-µs period (signaling a return to C0 state), the frequency is restored to full-speed operation after the 16K-cycle delay.

After this delay, if no external interrupts are seen, the power management control logic completes the transition to either one-eighth or one-sixteenth the full-frequency rate.

A similar sequence occurs when transitioning from the one-eighth or one-sixteenth frequency C3 state to C0 state. The power control logic bumps the frequency to half the full-speed rate for 16K cycles and afterwards moves to the full-frequency rate.

Changing the value in the **sysclk** divider provides a mechanism to throttle the internal 21164PC frequency within the C0 state. This divider is directly accessible to software using the SR[43:40] bits in the ICSR (see Section 5.1.17).

The **sysclk** divider is programmed to define what is meant by full-speed operation. This provides some degree of control for the final C3 state frequency.

Table 10–1 shows the values that are loaded into the X and Z dividers by the power management control logic during the different phases of the transition between ACPI states.

Table 10–1 X and Z Divider Values

X Divider	Z Divider	Internal Clock Rate
1	2	Full_speed/1
2	4	Full_speed/2
8	16	Full_speed/8
16	32	Full_speed/16

Whenever the internal clock rate is in the Full_speed/1 state, the SS bit (ICSR [44]) is cleared. When it is not in the Full_speed/1 state, the SS bit is cleared.

A specific code sequence must be followed when either changing operating frequency within the C0 state or when changing between ACPI states. Whenever the value in any PLL divider changes, phase-lock is lost between the internal and external clocks for a specific period of time. The 21164PC cannot communicate with the rest of the system when the external and internal clocks are out of phase. This means that any code executed during the time when the two clocks are not phase locked can not access main memory or the external L2 cache.

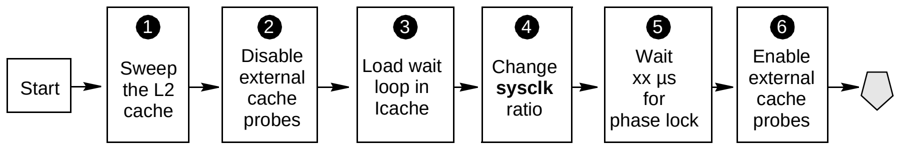
Conversely, the system will not be able to probe the L2 or onchip caches during this time. Obviously, great care must be taken when writing the power management code to adhere to these limitations.

The next three sections include several flowcharts that provide a general guideline for power management firmware writers.

10.2 Changing Frequency Within C0 State

The flowchart in Figure 10–2 provides a general guide for the code sequence that must be followed to change the frequency within the C0 state.

Figure 10–2 Changing Frequency Within the C0 State


Start
1
Sweep the L2 cache
2
Disable external cache probes
3
Load wait loop in Icache
4
Change **sysclk** ratio
5
Wait xx µs for phase lock
6
Enable external cache probes

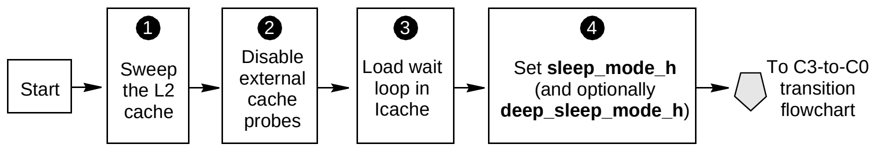
1. All dirty data in the L2 cache must be written back to memory. A simple loop that loads a contiguous block of data the size of the L2 cache can be used to sweep dirty data from the cache.
2. The core logic chipset must disable cache probes because phase lock will be lost for a period of time. Since the L2 cache has been flushed of dirty data (and the rest of this code sequence will not create dirty cache entries), DMA references do not need to probe the cache.
3. The code loop that waits the necessary time to ensure that phase lock is restored must be put into the onchip Icache, since offchip references do not work once phase lock is lost.
4. The appropriate value is loaded into the SR[43:40] bits of the ICSR.
5. The 21164PC must wait until phase lock is restored. This delay is 50 µs.

6. Once phase lock is restored, cache probes are enabled and normal operation restarts.

10.3 Entering C3 State

The flowchart in Figure 10–3 provides a general guide for the code sequence that must be followed to enter the C3 state.

Figure 10–3 Entering the C3 State


Start
1
Sweep the L2 cache
2
Disable external cache probes
3
Load wait loop in Icache
4
Set **sleep_mode_h** (and optionally **deep_sleep_mode_h**)
To C3-to-C0 transition flowchart

1. All dirty data in the L2 cache must be written back to memory. A simple loop that loads a contiguous block of data the size of the L2 cache can be used to sweep dirty data from the cache.
2. The core logic chipset must disable cache probes because phase lock will be lost for a period of time. Since the L2 cache has been flushed of dirty data (and the rest of this code sequence will not create dirty cache entries), DMA references do not need to probe the cache.
3. The code loop that will wait the necessary time to ensure that phase lock is restored in the C3-to-C0 state transition must be put into the onchip Icache since offchip references do not work once phase lock is lost.
4. **sleep_mode_h** and optionally **deep_sleep_mode_h** are set to start the transition to either one-eighth or one-sixteenth full-speed operation.

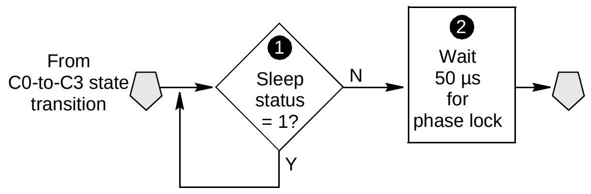
After **sleep_mode_h** (and, optionally, **deep_sleep_mode_h**) are set, the internal frequency is throttled down to half the full-speed rate for 16,384 (16K) cycles. The power management control logic keeps the frequency at the half-speed rate for the full 16K cycles. If an external interrupt occurs during this 30-µs time period (signaling go back to C0 state), the frequency is restored to full-speed operation after the 16K cycle delay.

After this delay, if no external interrupts are seen, the power management control logic completes the transition to either one-eighth or one-sixteenth the full-frequency rate.

10.4 C3-to-C0 State Transition

The C3-to-C0 state transition is initiated whenever an external interrupt is asserted when the 21164PC is either in the C3 state or in the 16K-cycle delay period going to the C3 state. The flowchart in Figure 10–4 provides a general guide for the code sequence that must be followed to transition from the C3 state to the C0 state.

Figure 10–4 C3-to-C0-State Transition



1. Loop until **sleep_status_h** equals zero, indicating that the X and Z dividers were set to full frequency operation. At this time, the internal and external clocks are still out of phase.
2. The code must loop until phase lock is restored. A delay of 50 µs is needed to reestablish phase lock.

11 Thermal Management

This chapter describes the 21164PC thermal management and thermal design considerations.

11.1 Operating Temperature

The 21164PC is specified to operate when the temperature at the center of the heat sink (T_c) is 66°C for 533 MHz, 65°C for 600 MHz, or 63.5°C for 666 MHz. Temperature (T_c) should be measured at the center of the heat sink (between the two package studs). The GRAFOIL pad is the interface material between the package and the heat sink.

Table 11–1 lists the values for the center of heat-sink-to-ambient ($\Theta_c a$) for the 413-pin grid array. Table 11–2 shows the allowable T_a (without exceeding T_c) at various airflows.

Note: COMPAQ recommends using the heat sink because it greatly improves the ambient temperature requirement.

Table 11–1 $\Theta_c a$ at Various Airflows

Frequency: 533 MHz, 600 MHz, and 666 MHz						
	Airflow (linear ft/min)					
	100	**200**	**400**	**600**	**800**	**1000**
$\Theta_c a$ with heat sink 1 (°C/W)	3.2	1.7	0.95	0.75	0.65	0.55
$\Theta_c a$ with heat sink 2 (°C/W) (includes 52×10 mm fan)	◄── 0.75 ──►					

Operating Temperature

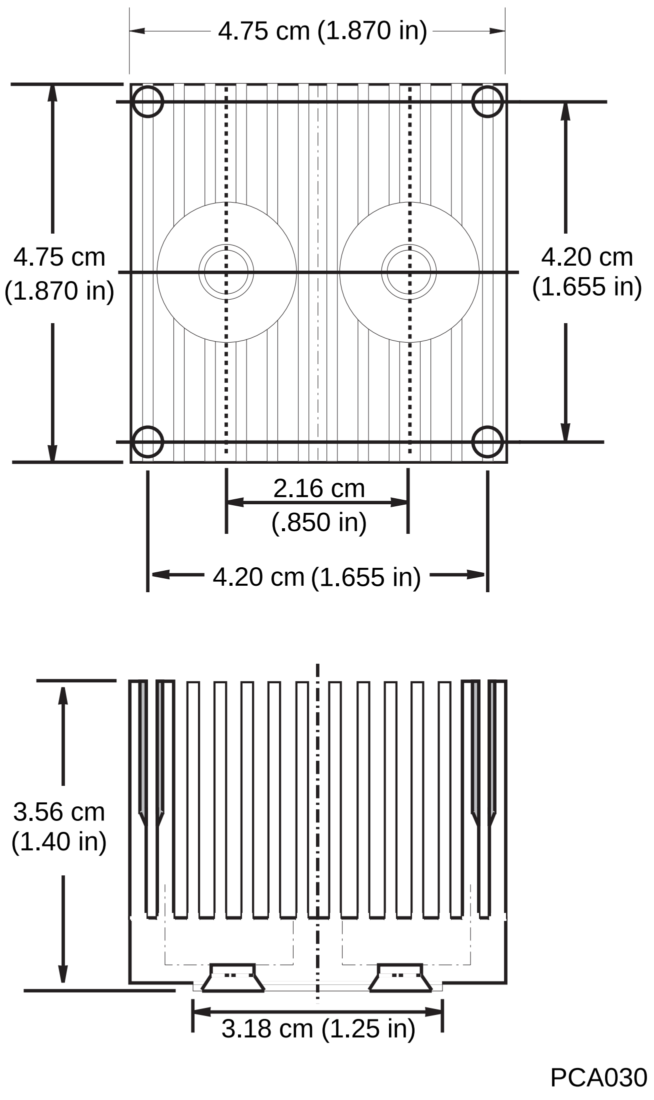
Table 11–2 Maximum T_a at Various Airflows

	Airflow (linear ft/min)					
	100	**200**	**400**	**600**	**800**	**1000**
Frequency: 533 MHz, Power: 18.0 W @Vdd = 2.5 V and @Vddi = 2.0 V						
T_a with heat sink 1 (°C)	—	35.4	48.9	52.5	54.3	56.1
T_a with heat sink 2 (°C) (includes 52×10 mm fan)	◄——— 52.5 ———►					
Frequency: 600 MHz, Power: 20.0 W @Vdd = 2.5 V and @Vddi = 2.0 V						
T_a with heat sink 1 (°C)	—	31.0	46.0	50.0	52.0	54.0
T_a with heat sink 2 (°C) (includes 52×10 mm fan)	◄——— 50.0 ———►					
Frequency: 666 MHz, Power: 23.0 W @Vdd = 2.5 V and @Vddi = 2.0 V						
T_a with heat sink 1 (°C)	—	24.4	41.7	46.3	48.6	50.9
T_a with heat sink 2 (°C) (includes 52×10 mm fan)	◄——— 46.3 ———►					

11.2 Heat-Sink Specifications

Figure 11–1 describes the specifications of heat sink 1. Heat sink 2 has the exact same specifications, plus an added 52×10 mm fan.

Figure 11–1 Heat Sink 1



11.3 Thermal Design Considerations

Follow these guidelines for printed circuit board (PCB) component placement:

- Orient the 21164PC on the PCB with the heat-sink fins aligned with the airflow direction.
- Avoid preheating ambient air. Place the 21164PC 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 21164PC.
- Do not restrict the airflow across the 21164PC heat sink. Placement of other devices must allow for maximum system airflow in order to maximize the performance of the heat sink.

12
Mechanical Packaging Information

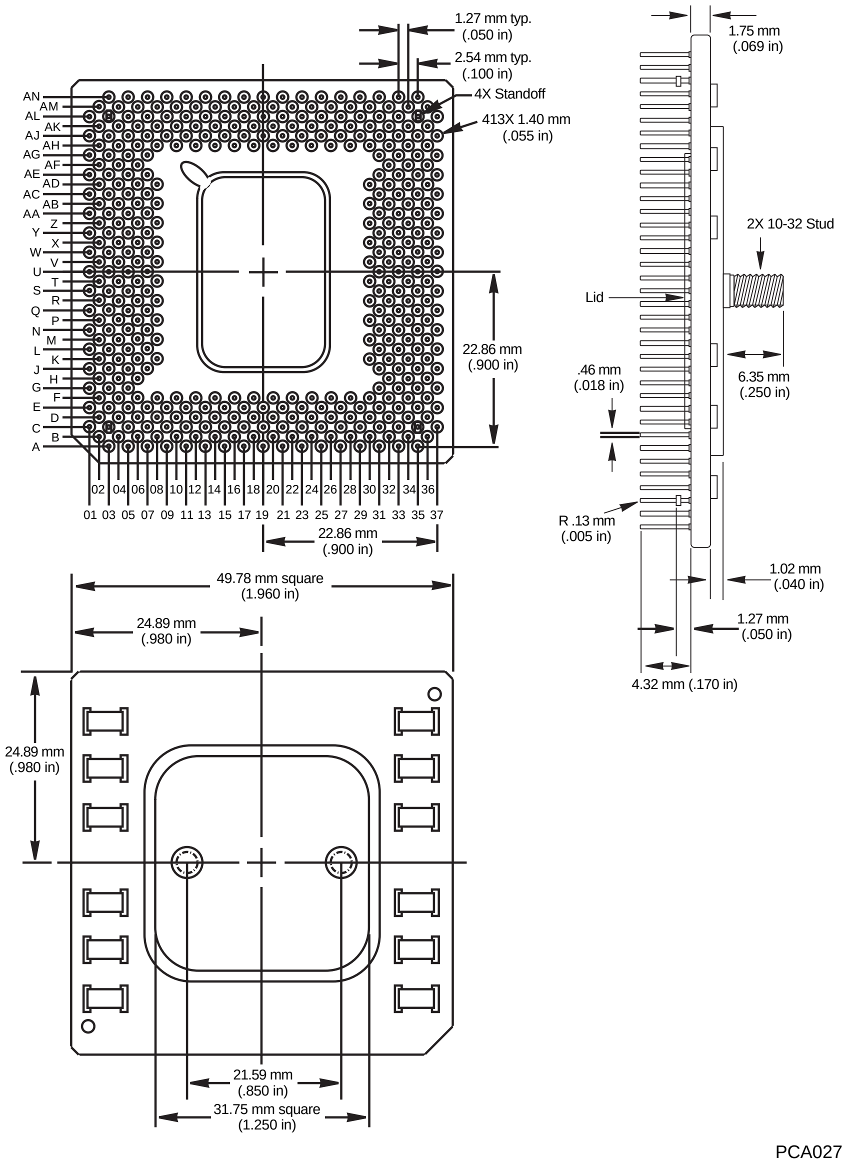
This chapter describes the 21164PC mechanical packaging including chip package physical specifications and a signal/pin list. For heat-sink dimensions, refer to Chapter 11.

12.1 Mechanical Specifications

Figure 12–1 shows the package physical dimensions without a heat sink.

Mechanical Specifications

Figure 12–1 Package Dimensions



12.2 Signal Descriptions and Pin Assignment

This section provides detailed information about the 21164PC pinout. The 21164PC has 413 pins aligned in an interstitial pin grid array (IPGA) design.

12.2.1 Signal Pin Lists

Table 12–1 lists the 21164PC signal pins and their corresponding pin grid array (PGA) locations in alphabetic order; Table 12–2 lists the voltage reference, power, and ground pins. There are 283 functional signal pins, 8 spare signal pins (unused), 4 voltage reference pins, 23 external power (**Vdd**) pins, 34 internal power (**Vddi**) pins, 1 **Vdd_PLL** pin, and 60 ground (**Vss**) pins, for a total of 413 pins in the array.

Table 12–1 Alphabetic Signal Pin List *(Sheet 1 of 5)*

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
addr_h[4]	AK8	**addr_h[5]**	AJ13	**addr_h[6]**	AL11
addr_h[7]	AL5	**addr_h[8]**	AK6	**addr_h[9]**	AH12
addr_h[10]	AJ11	**addr_h[11]**	AL3	**addr_h[12]**	AG7
addr_h[13]	AJ9	**addr_h[14]**	AJ7	**addr_h[15]**	AK4
addr_h[16]	AJ1	**addr_h[17]**	AH6	**addr_h[18]**	AJ3
addr_h[19]	AH4	**addr_h[20]**	AF6	**addr_h[21]**	AG1
addr_h[22]	AH34	**addr_h[23]**	AA31	**addr_h[24]**	AE35
addr_h[25]	AC31	**addr_h[26]**	AG35	**addr_h[27]**	AG31
addr_h[28]	AJ37	**addr_h[29]**	AH32	**addr_h[30]**	AH36
addr_h[31]	AJ35	**addr_h[32]**	AK36	**addr_h[33]**	AJ31
addr_h[34]	AK34	**addr_h[35]**	AK32	**addr_h[36]**	AK30
addr_h[37]	AJ29	**addr_h[38]**	AL33	**addr_h[39]**	AN29
addr_bus_req_h	E21	**addr_res_h[0]**	F32	**addr_res_h[1]**	A35
cack_h	A17	**clk_in_h**	AN19	**clk_in_l**	AM20
clk_mode_h[0]	AL23	**clk_mode_h[1]**	AJ25	**clk_mode_h[2]**	AL25
cmd_h[0]	D2	**cmd_h[1]**	D8	**cmd_h[2]**	C7
cmd_h[3]	E3	**cpu_clk_out_h**	AJ19	**dack_h**	F20

Table 12–1 Alphabetic Signal Pin List *(Sheet 2 of 5)*

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
data_h[0]	E37	**data_h[1]**	K34	**data_h[2]**	M34
data_h[3]	M32	**data_h[4]**	F36	**data_h[5]**	N31
data_h[6]	G35	**data_h[7]**	M30	**data_h[8]**	L37
data_h[9]	P32	**data_h[10]**	P34	**data_h[11]**	N37
data_h[12]	Q33	**data_h[13]**	Q35	**data_h[14]**	H36
data_h[15]	J35	**data_h[16]**	R32	**data_h[17]**	R34
data_h[18]	Q37	**data_h[19]**	S33	**data_h[20]**	R36
data_h[21]	L35	**data_h[22]**	Q31	**data_h[23]**	T32
data_h[24]	S35	**data_h[25]**	T34	**data_h[26]**	S37
data_h[27]	T36	**data_h[28]**	M36	**data_h[29]**	N35
data_h[30]	U35	**data_h[31]**	U37	**data_h[32]**	V36
data_h[33]	S31	**data_h[34]**	W35	**data_h[35]**	T30
data_h[36]	U31	**data_h[37]**	W37	**data_h[38]**	Y37
data_h[39]	X34	**data_h[40]**	X32	**data_h[41]**	Y35
data_h[42]	Y33	**data_h[43]**	V30	**data_h[44]**	V32
data_h[45]	Z34	**data_h[46]**	AA37	**data_h[47]**	Z32
data_h[48]	V34	**data_h[49]**	AA35	**data_h[50]**	AC37
data_h[51]	W31	**data_h[52]**	AB36	**data_h[53]**	AB34
data_h[54]	AD34	**data_h[55]**	AB32	**data_h[56]**	AE37
data_h[57]	AD32	**data_h[58]**	Y31	**data_h[59]**	AC35
data_h[60]	AE33	**data_h[61]**	AE31	**data_h[62]**	AF34
data_h[63]	AF32	**data_h[64]**	J5	**data_h[65]**	L1
data_h[66]	P6	**data_h[67]**	Q7	**data_h[68]**	L7
data_h[69]	M6	**data_h[70]**	L5	**data_h[71]**	K4
data_h[72]	M2	**data_h[73]**	M4	**data_h[74]**	P4
data_h[75]	Q5	**data_h[76]**	Q1	**data_h[77]**	R6

Table 12–1 Alphabetic Signal Pin List *(Sheet 3 of 5)*

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
data_h[78]	N3	**data_h[79]**	N1	**data_h[80]**	S7
data_h[81]	Q3	**data_h[82]**	T8	**data_h[83]**	R4
data_h[84]	R2	**data_h[85]**	T6	**data_h[86]**	T4
data_h[87]	S3	**data_h[88]**	U7	**data_h[89]**	S1
data_h[90]	T2	**data_h[91]**	U3	**data_h[92]**	V6
data_h[93]	U1	**data_h[94]**	V8	**data_h[95]**	W7
data_h[96]	V2	**data_h[97]**	W3	**data_h[98]**	X8
data_h[99]	V4	**data_h[100]**	W1	**data_h[101]**	W5
data_h[102]	X4	**data_h[103]**	X6	**data_h[104]**	Y1
data_h[105]	Y5	**data_h[106]**	Z4	**data_h[107]**	Y3
data_h[108]	Y7	**data_h[109]**	Z6	**data_h[110]**	AA1
data_h[111]	AA3	**data_h[112]**	AB4	**data_h[113]**	AB6
data_h[114]	AD4	**data_h[115]**	AC7	**data_h[116]**	AB2
data_h[117]	AC1	**data_h[118]**	AD6	**data_h[119]**	AE5
data_h[120]	AC3	**data_h[121]**	AE3	**data_h[122]**	AF4
data_h[123]	AE1	**data_h[124]**	AF2	**data_h[125]**	AG3
data_h[126]	AH2	**data_h[127]**	AE7	**data_adsc_l**	C33
data_adv_l	E31	**data_bus_req_h**	A21	**data_ram_oe_l**	F18
data_ram_we_l[0]	B18	**data_ram_we_l[1]**	A19	**data_ram_we_l[2]**	B20
data_ram_we_l[3]	D20	**dc_ok_h**	AH18	**fill_h**	F6
fill_dirty_h	AN9	**fill_error_h**	B4	**fill_id_h**	C1
idle_bc_h	E9	**index_h[4]**	E23	**index_h[5]**	C25
index_h[6]	C21	**index_h[7]**	B26	**index_h[8]**	A23
index_h[9]	C23	**index_h[10]**	A25	**index_h[11]**	A27
index_h[12]	F26	**index_h[13]**	E25	**index_h[14]**	C27
index_h[15]	C29	**index_h[16]**	E27	**index_h[17]**	E29

Table 12–1 Alphabetic Signal Pin List *(Sheet 4 of 5)*

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
index_h[18]	D30	**index_h[19]**	A29	**index_h[20]**	A31
index_h[21]	C31	**int4_valid_h[0]**	K32	**int4_valid_h[1]**	L31
int4_valid_h[2]	N7	**int4_valid_h[3]**	H4	**irq_h[0]**	AL31
irq_h[1]	AL29	**irq_h[2]**	AL27	**irq_h[3]**	AN27
lw_parity_h[0]	E35	**lw_parity_h[1]**	D36	**lw_parity_h[2]**	L3
lw_parity_h[3]	P8	**mch_hlt_irq_h**	AJ27	**port_mode_h[0]**	AH20
port_mode_h[1]	AJ23	**pwr_fail_irq_h**	AM26	**srom_clk_h**	AL17
srom_data_h	AN17	**srom_oe_l**	AK16	**srom_present_l**	AJ17
st_clk1_h	C9	**st_clk2_h**	B30	**st_clk3_h**	D32
sys_clk_out1_h	AM18	**sys_clk_out2_h**	AK22	**sys_mch_chk_irq_h**	AN25
sys_reset_l	AN23	**tag_data_h[19]**	A11	**tag_data_h[20]**	E11
tag_data_h[21]	F8	**tag_data_h[22]**	C11	**tag_data_h[23]**	E13
tag_data_h[24]	A3	**tag_data_h[25]**	E15	**tag_data_h[26]**	F10
tag_data_h[27]	B6	**tag_data_h[28]**	F12	**tag_data_h[29]**	B8
tag_data_h[30]	A9	**tag_data_h[31]**	C13	**tag_data_h[32]**	C15
tag_data_par_h	B12	**tag_dirty_h**	A13	**tag_ram_oe_l**	C17
tag_ram_we_l	D18	**tag_valid_h**	E17	**tck_h**	AN13
tdi_h	AL15	**tdo_h**	AN15	**temp_sense**	AM12
test_status_h[1]	AL13	**tms_h**	AJ15	**trst_l**	AN11
victim_pending_h	A15	**VDD_PLL**	AH24	**spare1**	AK18
spare2	AH26	**spare3**	J33	**spare4**	C5
spare5	D4	**spare6**	E1	**spare7**	H6
spare8	J31	**spare9**	J7	**spare10**	F2
spare11	E7	**spare12**	D6	**spare13**	G3
spare14	G1	**spare15**	J3	**spare16**	J1
spare17	F4	**spare18**	B34	**spare19**	G31

Table 12–1 Alphabetic Signal Pin List *(Sheet 5 of 5)*

Signal	PGA Location	Signal	PGA Location	Signal	PGA Location
spare20	F34	**spare21**	H32	**spare22**	C37
spare23	D34	**spare24**	H34	**spare25**	K30
spare26	G37				

Table 12–2 lists the voltage reference, power, and ground pins.

Table 12–2 Voltage Reference, Power, and Ground Pins

Signal	PGA Location
Vss Metal planes 2, 6	A5, A7, A33, B2, B14, B24, B36, D12, D16, D22, D26, E5, E19, E33, F16, F22, F28, H2, K8, L33, P2, P30, P36, S5, W33, Z2, Z8, Z30, Z36, AC5, AC33, AD8, AD30, AF36, AH10, AH16, AH22, AH28, AJ5, AJ33, AK12, AK20, AK26, AL1, AL19, AL21, AL37, AM2, AM8, AM14, AM24, AM30, AM36, AN3, AN5, AN7, AN21, AN31, AN33, AN35
Vdd Metal plane 7	B32, C19, D10, D14, D24, D28, G5, G33, N5, N33, U5, U33, AA5, AA33, AG5, AG33, AJ21, AK10, AK14, AK24, AK28, AL7, AL9
Vddi Metal plane 4	AB8, AB30, AD2, AD36, AH8, AH14, AH30, AK2, AL35, AM4, AM6, AM10, AM16, AM22, AM28, AM32, AM34, B10, B16, B22, B28, C3, C35, F14, F24, F30, K2, K36, M8, R8, R30, X2, X30, X36
Vddref	J37, K6, AA7, AG37

12.2.2 Pin Assignment

Figure 12–2 shows the 21164PC pinout from the top view with pins facing down.

Figure 12–2 21164PC Top View (Pin Down)

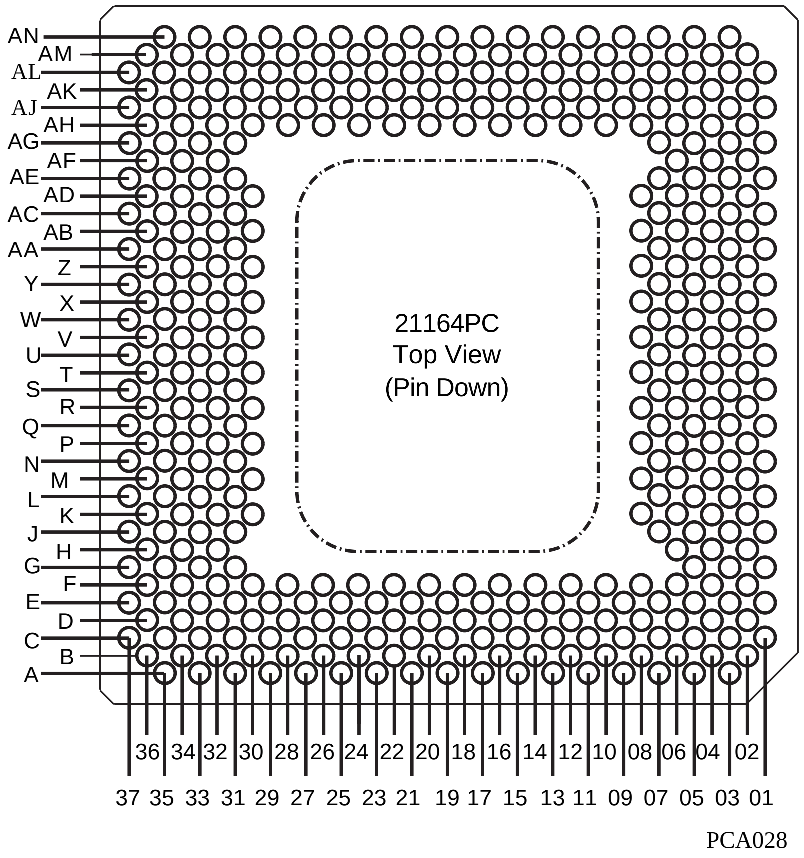
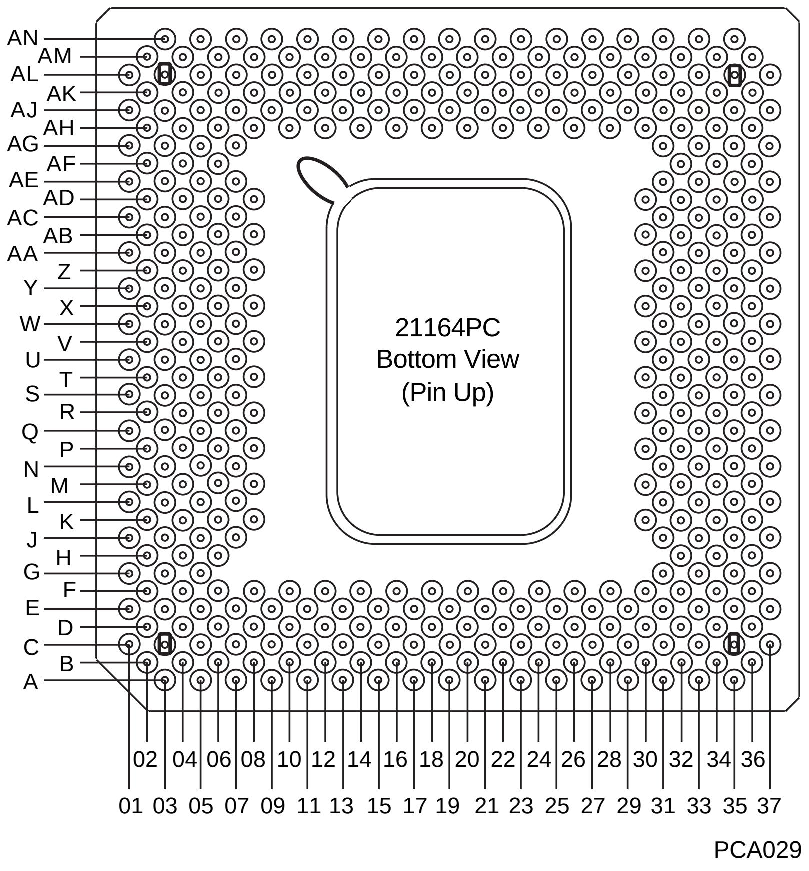


Figure 12–3 shows the 21164PC pinout from the bottom view with pins facing up.

Figure 12–3 21164PC Bottom View (Pin Up)



13
Testability and Diagnostics

This chapter describes the 21164PC user-oriented testability features. The 21164PC also has several internal testability features that are implemented for factory use only. These features are beyond the scope of this document.

13.1 Test Port Pins

Table 13–1 summarizes the test port pins and their functions.

Table 13–1 21164PC Test Port Pins

Pin Name	Type	Function
port_mode_h[1]	I	Must be false.
port_mode_h[0]	I	Must be false.
srom_clk_h/Tx	O	Supplies clock to SROMs or transmits serial terminal data.
srom_data_h/Rx	I	Receives SROM or serial terminal data.
srom_oe_l	O	SROM enable.
srom_present_l	I	Tied low if serial ROMs (SROMs) are present in system.
tck_h	I	IEEE 1149.1 TCK port.
tdi_h	I	IEEE 1149.1 TDI port.
tdo_h	O	IEEE 1149.1 TDO port.
test_status_h[1]	O	Outputs an IPR-written value and timeout reset.
tms_h	I	IEEE 1149.1 TMS port.
trst_l	I	IEEE 1149.1 optional TRST port.

13.2 Test Interface

The 21164PC test interface supports a serial ROM interface, a serial diagnostic terminal interface, and an IEEE 1149.1 test access port. These ports are available and set to normal test interface mode when **port_mode_h[1:0]**=00. Driving these pins to a value of anything other than 00 redefines all other test interface pins and invokes special factory test modes not covered in this document.

The SROM port is described in Section 7.4 and the serial terminal port is described in Section 7.5.

13.2.1 IEEE 1149.1 Test Access Port

Pins **tdi_h**, **tdo_h**, **tck_h**, **tms_h**, and **trst_l** constitute the IEEE 1149.1 test access port. This port accesses the 21164PC chip's boundary-scan register and chip tristate functions for board-level manufacturing test. The port also allows access to factory manufacturing features not described in this document. The port is compliant with most requirements of IEEE 1149.1 test access port.

Compliance Enable Inputs

Table 13–2 shows the compliance enable inputs and the pattern that must be driven to those inputs in order to activate the 21164PC IEEE 1149.1 circuits.

Table 13–2 Compliance Enable Inputs

Input	Compliance Enable Pattern
port_mode_h[1:0]	00
dc_ok_h	1

Exceptions to Compliance

The 21164PC is compliant with IEEE Standard 1149.1—1993, with two exceptions. Both exceptions provide enhanced value to the user.

1. **trst_l** pin

   The optional **trst_l** pin has an internal pull-down, instead of a pull-up, as required by IEEE 1149.1 (non-complied spec 3.6.1(b) in IEEE 1149.1–1993). The **trst_l** pull-down allows the chip to automatically force reset to the IEEE 1149.1 circuits in a system in which the IEEE 1149.1 port is unconnected. This may be considered a feature for most system designs that use IEEE 1149.1 circuits solely during module manufacturing.

Note: COMPAQ recommends that the **trst_l** pin be driven low (asserted) when the JTAG (IEEE 1149.1) logic is not in use.

2. Coverage of oscillator differential input pins

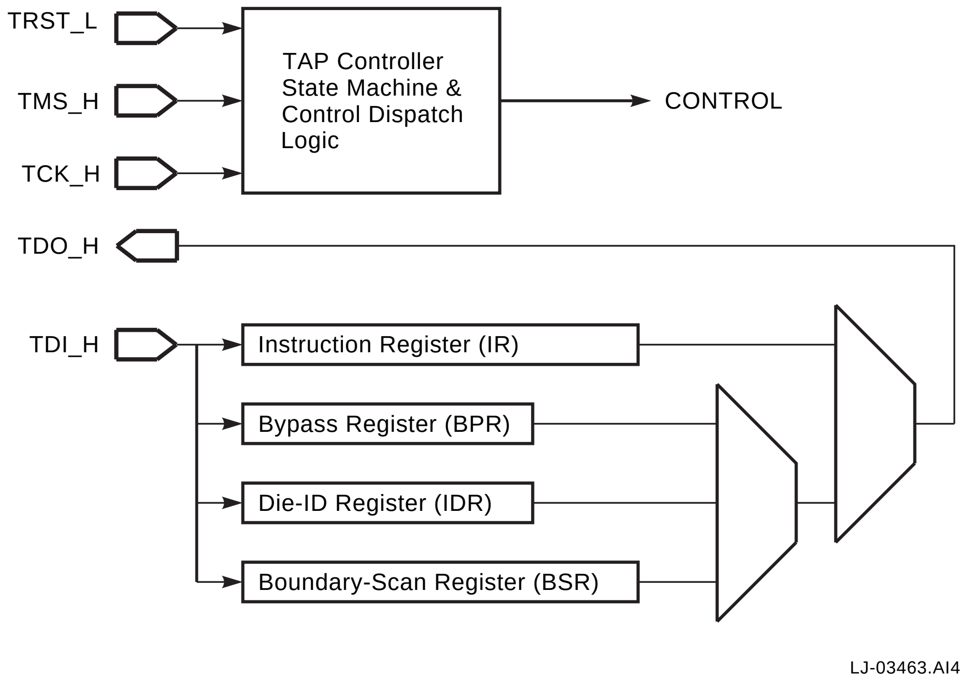
   The two differential clock input pins, **clk_in_h** and **clk_in_l**, do not have any boundary-scan cells associated with them (noncompliant spec 10.4.1(b) in IEEE 1149.1–1993). Instead, there is an extra input BSR cell in the boundary-scan register in bit position 268 (at pin **dc_ok_h**). This cell captures the output of a "clock sniffer" circuit. It captures a 1 when the oscillator is connected, and captures a 0 if the chip's oscillator connections are broken.

   This exception to the standard is made to permit a meaningful test of the oscillator input pins.

Refer to IEEE Standard 1149.1–1993 *A Test Access Port and Boundary Scan Architecture* for a full description of the specification.

Figure 13–1 shows the user-visible features from this port.

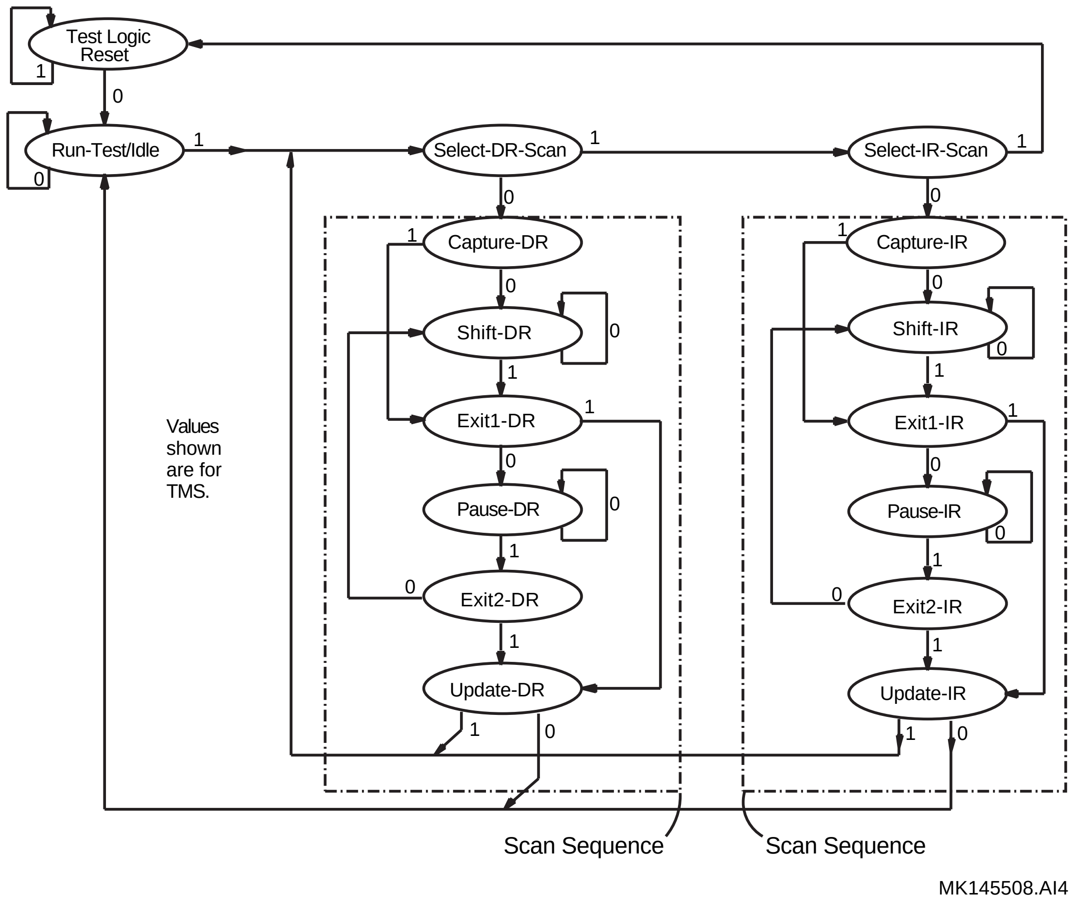
Figure 13–1 IEEE 1149.1 Test Access Port


TRST_L
TMS_H
TCK_H
TDO_H
TDI_H
TAP Controller State Machine & Control Dispatch Logic
CONTROL
Instruction Register (IR)
Bypass Register (BPR)
Die-ID Register (IDR)
Boundary-Scan Register (BSR)
LJ-03463.AI4

TAP Controller

The TAP controller contains a state machine. It interprets IEEE 1149.1 protocols received on signal **tms_h** and generates appropriate clocks and control signals for the testability features under its jurisdiction. The state machine is shown in Figure 13–2.

Figure 13–2 TAP Controller State Machine



Instruction Register

The 5-bit-wide instruction register (IR) supports IEEE 1149.1 mandated public instructions (EXTEST, SAMPLE, BYPASS, HIGHZ) and a number of optional instructions for public and private factory use. Table 13–3 summarizes the public instructions and their functions.

During the capture operation, the shift register stage of IR is loaded with the value 00001. This automatic load feature is useful for testing the integrity of the IEEE 1149.1 scan chain on the module.

Table 13–3 Instruction Register

IR[4:0]	Name	Selected Scan Register	Operation
00000	EXTEST	BSR	BSR drives pins. Interconnect test mode.
00010	SAMPLE/PRELOAD	BSR	Preloads BSR.
00010	Private	BSR	Private.
00011	Private	BSR	Private.
00100	CLAMP	BPR	BSR drives pins.
00101	HIGHZ	BPR	Tristate all output and I/O pins.
00110	Private	IDR	Private.
00111	Private	IDR	Private.
01000 through 11110	Private	BPR	Private.
11111	BYPASS	BPR	Default.

Bypass Register

The bypass register is a 1-bit shift register. It provides a short single-bit scan path through the port (chip).

Boundary-Scan Register

The 286-bit boundary-scan register is accessed during SAMPLE, EXTEST, and CLAMP instructions. Refer to Section 13.3 for the organization of this register.

13.2.2 Test Status Pin

One test status signal **test_status_h[1]** pin is used for extracting test status information from the chip. System reset drives the test status pin low. The default operation for **test_status_h[1]** is to output the IPR-written value.

- IPR read and write operations to the test status pin

  PALcode can write to the **test_status_h[1]** signal pin through hardware IPR access. Refer to Chapter 6.

- Timeout reset

  The 21164PC generates a timeout reset if an instruction is not retired within 1 billion cycles. The CPU signals the timeout reset event by outputting a 256 CPU cycle wide pulse on the **test_status_h[1]** pin. The pulse on the **test_status_h[1]** pin is clocked by **sysclk** and therefore appears as an approximately 256 CPU cycle pulse that rises and falls on system clock rising edges.

13.3 Boundary-Scan Register

The 21164PC boundary-scan register (BSR) is 286 bits long. Table 13–4 provides the boundary-scan register organization. The BSR is connected between the **tdi_h** and **tdo_h** pins whenever an instruction selects it (Table 13–3). The scan register runs clockwise beginning at the upper-left corner of the chip.

There are six groups of bidirectional pins, each group controlled from a group control cell. Loading a value of 1 in the control cell tristates the output drivers, and all bidirectional pins in the group are configured as input pins. The bidirectional pin groups are identified as groups gr_1 through gr_6 in the Control Group column in Table 13–4.

Information on Boundary Scan Description Language (BSDL) as it applies to the 21164PC boundary-scan register is available through your local sales office (see Appendix D).

Notes: The following notes apply to Table 13–4:

- The direction of shift is from top to bottom, and from left to right.
- The bottom-most signals appear first at the **tdo_h** pin when shifting.
- Given an arrayed signal of the form signal[a:b], signal[b] appears at the **tdo_h** pin prior to signal[a].

Table 13–4 Boundary-Scan Register Organization *(Sheet 1 of 4)*

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
addr_h[10:4]	B	285:279	io_bcell	gr_6	—
fill_dirty_h	I	278	in_bcell	—	—
temp_sense	O	—	None	—	Analog pin.
test_status_h	O	277	io_bcell	—	—
trst_l	B	—	None	—	—
tck_h	B	—	None	—	—
tms_h	B	—	None	—	—
tdo_h	O	—	None	—	—
tdi_h	B	—	None	—	—
srom_oe_l	O	276	io_bcell	—	—
srom_clk_h	O	275	io_bcell	—	—
srom_data_h	I	274	in_bcell	—	—
srom_present_l	I	273	in_bcell	—	—
port_mode_h[0:1]	I	—	None	—	Compliance enable pins.
pca57_clkout_l	O	—	None	—	—
pca57_clkin_l,h	I	—	None	—	—
spare1	B	272	io_bcell	—	Tied off as input.
sys_clk_out1_h	O	271	io_bcell	—	—
sys_clk_out2_h	O	270	io_bcell	—	—
sys_reset_l	I	269	io_bcell	—	—
dc_ok_h	I	—	None	—	Compliance enable pin.
OSC_SNIFFER_H	Internal	268	in_bcell	—	Captures 1 if osc is connected, otherwise, captures 0.
cpu_clk_out_h	O	—	None	—	For chip test.

Boundary-Scan Register

Table 13–4 Boundary-Scan Register Organization *(Sheet 2 of 4)*

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
pca57_clkout_h	O	—	None	—	—
clk_mode_h[0]	I	267	in_bcell	—	—
clk_in_h,l	I	—	None	—	—
clk_mode_h[1:2]	I	266:265	in_bcell	—	—
sys_mch_chk_irq_h	I	264	in_bcell	—	—
mch_hlt_irq_h	I	263	in_bcell	—	—
pwr_fail_irq_h	I	262	in_bcell	—	—
spare2	B	261	io_bcell	—	Tied off as input.
irq_h[3:0]	I	260:257	in_bcell	—	Upper-right corner.
TR_ADR	Control	256	io_bcell	gr_1	—
addr_h[39:22]	B	255:238	io_bcell	gr_1	—
TR_DDR	Control	237	io_bcell	gr_2	—
data_h[63:0]	B	236:173	io_bcell	gr_2	—
lw_parity_h[0:1]	B	172:171	io_bcell	gr_2	—
int4_valid_h[1:0]	O	170:169	io_bcell	—	—
spare3	B	168	io_bcell	—	Tied off as input.
spare26	B	167	io_bcell	—	BSR read as zero.
spare25	B	166	io_bcell	—	Mask out in BSR traces.
spare8	B	165	io_bcell	—	Mask out in BSR traces.
spare24	O	164	io_bcell	—	Mask out in BSR traces.
spare23	I	163	in_bcell	—	Mask out in BSR traces.
spare22	O	162	io_bcell	—	BSR read as one.
spare21	O	161	io_bcell	—	BSR read as one.
spare20	O	160	io_bcell	—	BSR read as one.
spare19	O	159	io_bcell	—	BSR read as zero.
spare18	O	158	io_bcell	—	BSR read as one.

Table 13–4 Boundary-Scan Register Organization *(Sheet 3 of 4)*

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
addr_res_h[1:0]	O	157:156	io_bcell	—	—
st_clk3_h	O	155	io_bcell	—	—
data_adsc_l	O	154	io_bcell	—	—
data_adv_l	O	153	io_bcell	—	—
st_clk2_h	O	152	io_bcell	—	Lower-right corner.
index_h[21:4]	O	151:134	io_bcell	—	—
data_bus_req_h	I	133	in_bcell	—	—
dack_h	I	132	in_bcell	—	—
addr_bus_req_h	I	131	in_bcell	—	—
data_ram_we_l[3:0]	O	130:127	io_bcell	—	—
data_ram_oe_l	O	126	io_bcell	—	—
cack_h	I	125	in_bcell	—	—
tag_ram_we_l	O	124	io_bcell	—	—
tag_ram_oe_l	O	123	io_bcell	—	—
victim_pending_h	O	122	io_bcell	—	—
TMIS1	Control	121	io_bcell	gr_3	—
tag_dirty_h	B	120	io_bcell	gr_3	—
tag_data_par_h	B	119	io_bcell	gr_3	—
tag_valid_h	B	118	io_bcell	gr_3	—
tag_data_h[19]	B	117	io_bcell	gr_3	—
tag_data_h[32:20]	B	116:104	io_bcell	gr_3	—
st_clk1_h	O	103	io_bcell	—	Lower-left corner.
idle_bc_h	I	102	in_bcell	—	—
fill_error_h	I	101	in_bcell	—	—
fill_id_h	I	100	in_bcell	—	—
fill_h	I	99	in_bcell	—	—

Table 13–4 Boundary-Scan Register Organization *(Sheet 4 of 4)*

Signal Name	Pin Type	BSR Count	BSR Cell Type	Control Group	Remarks
TTAG1	Control	98	io_bcell	gr_4	—
cmd_h[0:3]	B	97:94	io_bcell	gr_4	—
spare6	B	93	io_bcell	—	Mask out in BSR traces.
spare10	B	92	io_bcell	—	Mask out in BSR traces.
spare11	O	91	io_bcell	—	BSR read as zero.
spare12	O	90	io_bcell	—	BSR read as one.
spare13	O	89	io_bcell	—	BSR read as one.
spare14	O	88	io_bcell	—	BSR read as one.
spare4	B	87	io_bcell	—	Tied off as input.
spare5	B	86	io_bcell	—	Tied off as input.
spare15	I	85	in_bcell	—	Mask out in BSR traces.
spare16	O	84	io_bcell	—	BSR read as zero.
spare7	B	83	io_bcell	—	Mask out in BSR traces.
spare9	B	82	io_bcell	—	Mask out in BSR traces.
spare17	I	81	in_bcell	—	Mask out in BSR traces.
int4_valid_h[2:3]	O	80:79	io_bcell	—	—
TTAG2	Control	78	io_bcell	gr_5	—
lw_parity_h[3:2]	B	77:76	io_bcell	gr_5	—
data_h[64:127]	B	75:12	io_bcell	gr_5	—
TR_DDL	Control	11	io_bcell	gr_6	—
addr_h[21:11]	B	10:00	io_bcell	gr_6	—

A
Alpha Instruction Set

A.1 Alpha Instruction Summary

This appendix contains a summary of all Alpha architecture instructions. All values are in hexadecimal radix. Table A–1 describes the contents of the Format and Opcode columns that are in Table A–2.

Table A–1 Instruction Format and Opcode Notation

Instruction Format	Format Symbol	Opcode Notation	Meaning
Branch	Bra	oo	oo is the 6-bit opcode field.
Floating-point	F-P	oo.fff	oo is the 6-bit opcode field. fff is the 11-bit function code field.
Memory	Mem	oo	oo is the 6-bit opcode field.
Memory/ function code	Mfc	oo.ffff	oo is the 6-bit opcode field. ffff is the 16-bit function code in the displacement field.
Memory/ branch	Mbr	oo.h	oo is the 6-bit opcode field. h is the high-order 2 bits of the displacement field.
Operate	Opr	oo.ff	oo is the 6-bit opcode field. ff is the 7-bit function code field.
PALcode	Pcd	oo	oo is the 6-bit opcode field; the particular PALcode instruction is specified in the 26-bit function code field.

Alpha Instruction Summary

Qualifiers for operate instructions are shown in Table A–2. Qualifiers for IEEE and VAX floating-point instructions are shown in Tables A–5 and A–6, respectively.

Table A–2 Architecture Instructions *(Sheet 1 of 8)*

Mnemonic	Format	Opcode	Description
ADDF	F-P	15.080	Add F_floating
ADDG	F-P	15.0A0	Add G_floating
ADDL	Opr	10.00	Add longword
ADDL/V	Opr	10.40	Add longword
ADDQ	Opr	10.20	Add quadword
ADDQ/V	Opr	10.60	Add quadword
ADDS	F-P	16.080	Add S_floating
ADDT	F-P	16.0A0	Add T_floating
AMASK	Opr	11.61	Architectural mask
AND	Opr	11.00	Logical product
BEQ	Bra	39	Branch if = zero
BGE	Bra	3E	Branch if ≥ 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 ≤ zero
BLT	Bra	3A	Branch if < zero
BNE	Bra	3D	Branch if ≠ 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

Table A–2 Architecture Instructions *(Sheet 2 of 8)*

Mnemonic	Format	Opcode	Description
CMOVGE	Opr	11.46	CMOVE if ≥ 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 ≤ zero
CMOVLT	Opr	11.44	CMOVE if < zero
CMOVNE	Opr	11.26	CMOVE if ≠ zero
CMPBGE	Opr	10.0F	Compare byte
CMPEQ	Opr	10.2D	Compare signed quadword equal
CMPGEQ	F-P	15.0A5	Compare G_floating equal
CMPGLE	F-P	15.0A7	Compare G_floating less than or equal
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	15.0AD	Convert G_floating to F_floating

Table A–2 Architecture Instructions *(Sheet 3 of 8)*

Mnemonic	Format	Opcode	Description
CVTGQ	F-P	15.0AF	Convert G_floating to quadword
CVTLQ	F-P	17.010	Convert longword to quadword
CVTQF	F-P	15.0BC	Convert quadword to F_floating
CVTQG	F-P	15.0BE	Convert quadword to G_floating
CVTQL	F-P	17.030	Convert quadword to longword
CVTQL/SV	F-P	17.530	Convert quadword to longword
CVTQL/V	F-P	17.130	Convert quadword to longword
CVTQS	F-P	16.0BC	Convert quadword to S_floating
CVTQT	F-P	16.0BE	Convert quadword to T_floating
CVTST	F-P	16.2AC	Convert S_floating to T_floating
CVTTQ	F-P	16.0AF	Convert T_floating to quadword
CVTTS	F-P	16.0AC	Convert T_floating to S_floating
DIVF	F-P	15.083	Divide F_floating
DIVG	F-P	15.0A3	Divide G_floating
DIVS	F-P	16.083	Divide S_floating
DIVT	F-P	16.0A3	Divide T_floating
EQV	Opr	11.48	Logical equivalence
EXCB	Mfc	18.0400	Exception barrier
EXTBL	Opr	12.06	Extract byte low
EXTLH	Opr	12.6A	Extract longword high
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

Table A–2 Architecture Instructions *(Sheet 4 of 8)*

Mnemonic	Format	Opcode	Description
FBGE	Bra	36	Floating branch if ≥ zero
FBGT	Bra	37	Floating branch if > zero
FBLE	Bra	33	Floating branch if ≤ zero
FBLT	Bra	32	Floating branch if < zero
FBNE	Bra	35	Floating branch if ≠ zero
FCMOVEQ	F-P	17.02A	FCMOVE if = zero
FCMOVGE	F-P	17.02D	FCMOVE if ≥ zero
FCMOVGT	F-P	17.02F	FCMOVE if > zero
FCMOVLE	F-P	17.02E	FCMOVE if ≤ zero
FCMOVLT	F-P	17.02C	FCMOVE if < zero
FCMOVNE	F-P	17.02B	FCMOVE if ≠ zero
FETCH	Mfc	18.80	Prefetch data
FETCH_M	Mfc	18.A0	Prefetch data, modify intent
IMPLVER	Opr	11.6C	Implementation version
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

Table A–2 Architecture Instructions *(Sheet 5 of 8)*

Mnemonic	Format	Opcode	Description
LDBU	Mem	0A	Load zero-extended byte
LDF	Mem	20	Load F_floating
LDG	Mem	21	Load G_floating
LDL	Mem	28	Load sign-extended longword
LDL_L	Mem	2A	Load sign-extended longword locked
LDQ	Mem	29	Load quadword
LDQ_L	Mem	2B	Load quadword locked
LDQ_U	Mem	0B	Load unaligned quadword
LDS	Mem	22	Load S_floating
LDT	Mem	23	Load T_floating
LDWU	Mem	0C	Load zero-extended word
MAXSB8	Opr	1C.3E	Vector signed byte maximum
MAXSW4	Opr	1C.3F	Vector signed word maximum
MAXUB8	Opr	1C.3C	Vector unsigned byte maximum
MAXUW4	Opr	1C.3D	Vector unsigned word maximum
MB	Mfc	18.4000	Memory barrier
MF_FPCR	F-P	17.025	Move from floating-point control register
MINSB8	Opr	1C.3E	Vector signed byte minimum
MINSW4	Opr	1C.3F	Vector signed word minimum
MINUB8	Opr	1C.3C	Vector unsigned byte minimum
MINUW4	Opr	1C.3D	Vector unsigned word minimum
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

Table A–2 Architecture Instructions *(Sheet 6 of 8)*

Mnemonic	Format	Opcode	Description
MSKWH	Opr	12.52	Mask word high
MSKWL	Opr	12.12	Mask word low
MT_FPCR	F-P	17.024	Move to floating-point control register
MULF	F-P	15.082	Multiply F_floating
MULG	F-P	15.0A2	Multiply G_floating
MULL	Opr	13.00	Multiply longword
MULL/V	Opr	13.40	Multiply longword
MULQ	Opr	13.20	Multiply quadword
MULQ/V	Opr	13.60	Multiply quadword
MULS	F-P	16.082	Multiply S_floating
MULT	F-P	16.0A2	Multiply T_floating
ORNOT	Opr	11.28	Logical sum with complement
PERR	Opr	1C.31	Pixel error
PKLB	Opr	1C.37	Pack longwords to bytes
PKWB	Opr	1C.36	Pack words to bytes
RC	Mfc	18.E0	Read and clear
RET	Mbr	1A.2	Return from subroutine
RPCC	Mfc	18.C0	Read process cycle counter
RS	Mfc	18.F000	Read and set
S4ADDL	Opr	10.02	Scaled add longword by 4
S4ADDQ	Opr	10.22	Scaled add quadword by 4
S4SUBL	Opr	10.0B	Scaled subtract longword by 4
S4SUBQ	Opr	10.2B	Scaled subtract quadword by 4
S8ADDL	Opr	10.12	Scaled add longword by 8
S8ADDQ	Opr	10.32	Scaled add quadword by 8
S8SUBL	Opr	10.1B	Scaled subtract longword by 8

Table A–2 Architecture Instructions *(Sheet 7 of 8)*

Mnemonic	Format	Opcode	Description
S8SUBQ	Opr	10.3B	Scaled subtract quadword by 8
SEXTB	Opr	1C.00	Store byte
SEXTW	Opr	1C.01	Store word
SLL	Opr	12.39	Shift left logical
SRA	Opr	12.3C	Shift right arithmetic
SRL	Opr	12.34	Shift right logical
STB	Mem	0E	Store byte
STF	Mem	24	Store F_floating
STG	Mem	25	Store G_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
STS	Mem	26	Store S_floating
STT	Mem	27	Store T_floating
STW	Mem	0D	Store word
SUBF	F-P	15.081	Subtract F_floating
SUBG	F-P	15.0A1	Subtract G_floating
SUBL	Opr	10.09	Subtract longword
SUBL/V		10.49	
SUBQ	Opr	10.29	Subtract quadword
SUBQ/V		10.69	
SUBS	F-P	16.081	Subtract S_floating
SUBT	F-P	16.0A1	Subtract T_floating
TRAPB	Mfc	18.00	Trap barrier

Table A–2 Architecture Instructions *(Sheet 8 of 8)*

Mnemonic	Format	Opcode	Description
UMULH	Opr	13.30	Unsigned multiply quadword high
UNPKBL	Opr	1C.35	Unpack bytes to longwords
UNPKBW	Opr	1C.34	Unpack bytes to words
WMB	Mfc	18.44	Write memory barrier
XOR	Opr	11.40	Logical difference
ZAP	Opr	12.30	Zero bytes
ZAPNOT	Opr	12.31	Zero bytes not

A.1.1 Opcodes Reserved for COMPAQ

Table A–3 lists opcodes reserved for COMPAQ.

Table A–3 Opcodes Reserved for COMPAQ

Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic	Opcode
OPC01	01	OPC05	05	OPC0C	0C[1]
OPC02	02	OPC06	06	OPC0D	0D[1]
OPC03	03	OPC07	07	OPC0E	0E[1]
OPC04	04	OPC0A	0A[1]		

[1] Reserved when byte/word instructions are not enabled.

A.1.2 Opcodes Reserved for PALcode

Table A–4 lists the 21164-specific instructions. For more information, refer to Section 6.6.

Table A–4 Opcodes Reserved for PALcode

21164 Mnemonic	Opcode	Architecture Mnemonic	Function
HW_LD	1B	PAL1B	Performs Dstream load instructions.
HW_ST	1F	PAL1F	Performs Dstream store instructions.
HW_REI	1E	PAL1E	Returns instruction flow to the program counter (PC) pointed to by the EXC_ADDR internal processor register (IPR).
HW_MFPR	19	PAL19	Accesses the IDU, MTU, and Dcache IPRs.
HW_MTPR	1D	PAL1D	Accesses the IDU, MTU, and Dcache IPRs.

A.2 IEEE Floating-Point Instructions

Table A–5 lists the hexadecimal value of the 11-bit function code field for the IEEE floating-point instructions, with and without qualifiers. The opcode for these instructions is 16_{16}.

Table A–5 IEEE Floating-Point Instruction Function Codes *(Sheet 1 of 3)*

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	—	—	—	—	—	—	—
CMPTLE	0A7	—	—	—	—	—	—	—
CMPTLT	0A6	—	—	—	—	—	—	—
CMPTUN	0A4	—	—	—	—	—	—	—
CVTQS	0BC	03C	07C	0FC	—	—	—	—
CVTQT	0BE	03E	07E	0FE	—	—	—	—
CVTTS	0AC	02C	06C	0EC	1AC	12C	16C	1EC
DIVS	083	003	043	0C3	183	103	143	1C3

Table A–5 IEEE Floating-Point Instruction Function Codes *(Sheet 2 of 3)*

Mnemonic	None	/C	/M	/D	/U	/UC	/UM	/UD
DIVT	0A3	023	063	0E3	1A3	123	163	1E3
MULS	082	002	042	0C2	182	102	142	1C2
MULT	0A2	022	062	0E2	1A2	122	162	1E2
SUBS	081	001	041	0C1	181	101	141	1C1
SUBT	0A1	021	061	0E1	1A1	121	161	1E1

Mnemonic	/SU	/SUC	/SUM	/SUD	/SUI	/SUIC	/SUIM	/SUID
ADDS	580	500	540	5C0	780	700	740	7C0
ADDT	5A0	520	560	5E0	7A0	720	760	7E0
CMPTEQ	5A5	—	—	—	—	—	—	—
CMPTLE	5A7	—	—	—	—	—	—	—
CMPTLT	5A6	—	—	—	—	—	—	—
CMPTUN	5A4	—	—	—	—	—	—	—
CVTQS	—	—	—	—	7BC	73C	77C	7FC
CVTQT	—	—	—	—	7BE	73E	77E	7F3
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

Table A–5 IEEE Floating-Point Instruction Function Codes *(Sheet 3 of 3)*

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

Note: Because underflow cannot occur for CMPT*xx*, there is no difference in function or performance between CMPT*xx*/S and CMPT*xx*/SU. It is intended that software generate CMPT*xx*/SU in place of CMPT*xx*/S.

In the same manner, CVTQS and CVTQT can take an inexact result trap, but not an underflow. Because there is no encoding for a CVTQ*x*/SI instruction, it is intended that software generate CVTQ*x*/SUI in place of CVTQ*x*/SI.

A.3 VAX Floating-Point Instructions

Table A–6 lists the hexadecimal value of the 11-bit function code field for the VAX floating-point instructions. The opcode for these instructions is 15_{16}.

Table A–6 VAX Floating-Point Instruction Function Codes *(Sheet 1 of 2)*

Mnemonic	None	/C	/U	/UC	/S	/SC	/SU	/SUC
ADDF	080	000	180	100	480	400	580	500
ADDG	0A0	020	1A0	120	4A0	420	5A0	520
CMPGEQ	0A5	—	—	—	4A5	—	—	—
CMPGLE	0A7	—	—	—	4A7	—	—	—
CMPGLT	0A6	—	—	—	4A6	—	—	—
CVTDG	09E	01E	19E	11E	49E	41E	59E	51E
CVTGD	0AD	02D	1AD	12D	4AD	42D	5AD	52D
CVTGF	0AC	02C	1AC	12C	4AC	42C	5AC	52C
CVTQF	0BC	03C	—	—	—	—	—	—
CVTQG	0BE	03E	—	—	—	—	—	—
DIVF	083	003	183	103	483	403	583	503

Table A–6 VAX Floating-Point Instruction Function Codes *(Sheet 2 of 2)*

Mnemonic	None	/C	/U	/UC	/S	/SC	/SU	/SUC
DIVG	0A3	023	1A3	123	4A3	423	5A3	523
MULF	082	002	182	102	482	402	582	502
MULG	0A2	022	1A2	122	4A2	422	5A2	522
SUBF	081	001	181	101	481	401	581	501
SUBG	0A1	021	1A1	121	4A1	421	5A1	521
Mnemonic	**None**	**/C**	**/V**	**/VC**	**/S**	**/SC**	**/SV**	**/SVC**
CVTGQ	0AF	02F	1AF	12F	4AF	42F	5AF	52F

A.4 Opcode Summary

Table A–7 lists all Alpha opcodes from 00 (CALL_PAL) through 3F (BGT). In the table, the column headings that appear over the instructions have a granularity of 8_{16}. The rows beneath the Offset column supply the individual hexadecimal number to resolve that granularity.

If an instruction column has a 0 in the right (low) hexadecimal digit, replace that 0 with the number to the left of the slash in the Offset column on the instruction's row. If an instruction column has an 8 in the right (low) hexadecimal digit, replace that 8 with the number to the right of the slash in the Offset column.

For example, the third row (2/A) under the 10_{16} column contains the symbol INTS*, representing the all-integer shift instructions. The opcode for those instructions would then be 12_{16} because the 0 in 10 is replaced by the 2 in the Offset column. Likewise, the third row under the 18_{16} column contains the symbol JSR*, representing all jump instructions. The opcode for those instructions is 1A because the 8 in the heading is replaced by the number to the right of the slash in the Offset column.

Opcode Summary

The instruction format is listed under the instruction symbol.

Table A–7 Opcode Summary

Offset	00	08	10	18	20	28	30	38
0/8	PAL* (pal)	LDA (mem)	INTA* (op)	MISC* (mem)	LDF (mem)	LDL (mem)	BR (br)	BLBC (br)
1/9	Res	LDAH (mem)	INTL* (op)	\PAL\	LDG (mem)	LDQ (mem)	FBEQ (br)	BEQ (br)
2/A	Res	LDBU (mem)	INTS* (op)	JSR* (mem)	LDS (mem)	LDL_L (mem)	FBLT (br)	BLT (br)
3/B	Res	LDQ_U (mem)	INTM* (op)	\PAL\	LDT (mem)	LDQ_L (mem)	FBLE (br)	BLE (br)
4/C	Res	LDWU (mem)	Res	SEXT/ MVI* (op)	STF (mem)	STL (mem)	BSR (br)	BLBS (br)
5/D	Res	STW (mem)	FLTV* (op)	\PAL\	STG (mem)	STQ (mem)	FBNE (br)	BNE (br)
6/E	Res	STB (mem)	FLTI* (op)	\PAL\	STS (mem)	STL_C (mem)	FBGE (br)	BGE (br)
7/F	Res	STQ_U (mem)	FLTL* (op)	\PAL\	STT (mem)	STQ_C (mem)	FBGT (br)	BGT (br)

Symbol	**Meaning**
FLTI*	IEEE floating-point instruction opcodes
FLTL*	Floating-point operate instruction opcodes
FLTV*	VAX floating-point instruction opcodes
INTA*	Integer arithmetic instruction opcodes
INTL*	Integer logical instruction opcodes
INTM*	Integer multiply instruction opcodes
INTS*	Integer shift instruction opcodes
JSR*	Jump instruction opcodes
MISC*	Miscellaneous instruction opcodes
PAL*	PALcode instruction (CALL_PAL) opcodes
\PAL\	Reserved for PALcode
Res	Reserved for COMPAQ
SEXT/MVI*	Sign extend and motion video instruction set opcodes

A.5 Required PALcode Function Codes

The opcodes listed in Table A–8 are required for all Alpha implementations. The notation used is oo.ffff, where oo is the hexadecimal 6-bit opcode and ffff is the hexadecimal 26-bit function code.

Table A–8 Required PALcode Function Codes

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

A.6 21164PC Microprocessor IEEE Floating-Point Conformance

The 21164PC supports the IEEE floating-point operations as defined by the Alpha architecture. Support for a complete implementation of the IEEE *Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Standard 754 1985) is provided by a combination of hardware and software as described in the *Alpha Architecture Reference Manual*.

Additional information about writing code to support precise exception handling (necessary for complete conformance to the standard) is in the *Alpha Architecture Reference Manual.*

The following information is specific to the 21164PC:

- Invalid operation (INV)

  The invalid operation trap can be disabled for IEEE instructions. If the trap occurs, then the destination register is UNPREDICTABLE. This exception is signaled if any VAX architecture operand is nonfinite (reserved operand or dirty zero) and the operation can take an exception (that is, certain instructions, such as CPYS, never take an exception). For IEEE operations with the /S qualifier, nonfinite operations can be handled without taking an exception. This trap may be masked by setting the INVD bit and using the /S qualifier in the instruction. In this case, the register result will not be UNPREDICTABLE, but rather will contain a canonical quiet NAN (CQNAN).

- Divide-by-zero (DZE)

  The divide-by-zero trap can be disabled for IEEE instructions. If the trap occurs, then the destination register is UNPREDICTABLE. For VAX architecture format, this exception is signaled whenever the numerator is valid and the denominator is zero. For IEEE format, this exception is signaled whenever the numerator is valid and nonzero, with a denominator of ±0. If the exception occurs, then FPCR[DZE] is set and the trap is signaled to the IDU. If the instruction has the /S qualifier and DZED = 1, then the trap is suppressed and FPCR[DZE] is set and the destination register contains the appropriate signed infinity.

- Floating overflow (OVF)

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

- Underflow (UNF)

  The underflow trap can be disabled. If underflow occurs, then the destination register is forced to a true zero, consisting of a full 64 bits of zero. This is done even if the proper IEEE result would have been -0. The exception is signaled if the rounded result is smaller in magnitude than the smallest finite number that can be represented by the destination format. If the exception occurs, then FPCR[UNF] is set. If the trap is enabled, then the trap is signaled to the IDU. The 21164PC never produces a denormal number; underflow occurs instead.

- Inexact (INE)

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

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

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

- Integer overflow (IOV)

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

- Software completion (SWC)

  The software completion signal is not recorded in the FPCR. The state of this signal is always sent to the IDU. If the IDU detects the assertion of any of the listed exceptions concurrent with the assertion of the SWC signal, then it sets EXC_SUM[SWC].

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

B
21164PC Microprocessor Specifications

Table B–1 lists specifications for the 21164PC.

Table B–1 21164PC Microprocessor Specifications *(Sheet 1 of 2)*

Feature	Description
Cycle time range	1.67 ns (600 MHz) to 1.50 ns (666 MHz)
Process technology	0.28-µm CMOS
Transistor count	5.7 million
Die size	6.7 × 15 mm
Package	413-pin IPGA (interstitial pin grid array)
Number of signal pins	283
Typical worst-case power @**Vdd** = 2.5 V @**Vddi** = 2.0 V	18.5 W (int.) and 1.5 W (ext.) @ 1.67 ns cycle time (600 MHz) 20.5 W (int.) and 2.5 W (ext.) @ 1.50 ns cycle time (666 MHz)
Power supply	2.5 V dc, 2.0 V dc
Clocking input	One times the internal clock speed or 4 to 15 times the input clock, based on clock mode
Virtual address size	43 bits
Physical address size	33 bits
Page size	8KB
Issue rate	2 integer instructions and 2 floating-point instructions per cycle
Integer instruction pipeline	7 stage
Floating instruction pipeline	9 stage

Table B–1 21164PC Microprocessor Specifications *(Sheet 2 of 2)*

Feature	Description
Onchip L1 Dcache	16KB, virtually-indexed, physically-tagged, direct-mapped, write-through, 32-byte block, 32-byte fill
Onchip L1 Icache	16KB, virtually-indexed, virtually-tagged, 2-way set-associative, 64-byte block, 32-byte fill, 128 address space numbers (ASNs) (MAX_ASN=127)
Onchip data translation buffer	64-entry, fully associative, not-last-used replacement, 8K pages, 128 ASNs (MAX_ASN=127), full granularity hint support
Onchip instruction translation buffer	48-entry, fully associative, not-last-used replacement, 128 ASNs (MAX_ASN=127), full granularity hint support
Floating-point unit	Onchip FPU supports both IEEE and COMPAQ floating point
Bus	Separate data and address bus, 128-bit/64-bit data bus
Serial ROM interface	Allows microprocessor to access a serial ROM

C
Serial Icache Load Predecode Values

The following C code calculates the predecode values of a serial Icache load. A software tool called the SROM Packer converts a binary image into a format suitable for Icache serial loading. This tool is available from COMPAQ.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  DATA_BYTES_PER_REC 32
#define  MAX_INSTR 2048

int eparity(int) ;
int instrpredecode(int);
void build_vector();

/* Everything below is off by 54 bits. I'll add in 54 in main() */
/* fillmap [0 - 127] maps data 127:0, etc. */
/* fillmap[n] is bit position in output vector.  bit 0 of this vector is first-in;
 bit 201 is last */

int dfillmap [128] = {
                                     /* data 0:127 -- fillmap[0:127]*/
  44,46,48,50,52,54,56,58,           /* 0:7 */
  60,62,64,66,68,70,72,74,           /* 8:15 */
  76,78,80,82,84,86,88,90,           /* 16:23 */
  92,94,96,98,100,102,104,106,       /* 24:31 */
  45,47,49,51,53,55,57,59,           /* 32:39 */
  61,63,65,67,69,71,73,75,           /* 40:47 */
  77,79,81,83,85,87,89,91,           /* 48:55 */
  93,95,97,99,101,103,105,107,       /* 56:63 */
  130,132,134,136,138,140,142,144,   /* 64:71 */
  146,148,150,152,154,156,158,160,   /* 72:79 */
  162,164,166,168,170,172,174,176,   /* 80:87 */
  178,180,182,184,186,188,190,192,   /* 88:95 */
  131,133,135,137,139,141,143,145,   /* 96:103 */
  147,149,151,153,155,157,159,161,   /* 104:111 */
  163,165,167,169,171,173,175,177,   /* 112:119 */
  179,181,183,185,187,189,191,193    /* 120:127 */
  };
```

```
int BHTfillmap[8] = {          /* BHT vector 0:7 -- BHTfillmap[0:7] */
 201,200,199,198,197,196,195,194     /* 0:7 */
 };

int predfillmap[20] = {        /* predecodes 0:19 -- predfillmap[0:19] */
 108,110,112,114,116,               /* 0:4 */
 109,111,113,115,117,               /* 5:9 */
 120,122,124,126,128,               /* 10:14 */
 121,123,125,127,129                /* 15:19 */
 };

int octawpfillmap =            /* octaword parity */
 119;

int predpfillmap =             /* predecode parity */
 118;


int tagfillmap[29] = {         /* tag bits 13:42 -- tagfillmap[0:29] */
 29,28,27,26,25,24,23,22,21,      /* 13:22 */
 20,19,18,17,16,15,14,13,12,11,      /* 23:32 */
 10, 9, 8, 7, 6, 5, 4, 3, 2, 1       /* 33:42 */
 };

int asnfillmap[7] = {          /* asn 0:6 -- asnfillmap[0:6] */
 37,36,35,34,33,32,31               /* 0:6 */
 };

int asmfillmap =               /* asm -- asmfillmap */
 30;

int tagphysfillmap =           /* tagphysical address -- tagphysfillmap */
 38;

int tagvalfillmap[4] = {       /* tag valid bits 0:3 -- tagvalfillmap */
 42,41,40,39                        /* 0:3 */
 };

int tagparfillmap =            /* tag parity -- tagparfillmap */
 43;


/*
**   global variables
*/
char filename[256],ofilename[256],hfilename[256];
FILE *infile, *outfile, *hexfile;
int pdparity,
 tparity,
 tvalids,
 tphysical,
 bhtvector,
 offset;
 int base,
 asm,
 asn,
  tag,
  predecodes,
  owparity;
int device_size;

/*
**  define the ROM size in bits to determine the maximum number of instructions allowed
**  define the number of bits per instruction for 21164PC ICache
*/
#define ROMSIZE   262144
#define B_PER_INST 64

main(int argc, char *argv[])
{

  int i, j;
  int instatus, instr_count;
  int lines_written;
  char *charptr;
  int chksum;
  int instr[4], outvector[DATA_BYTES_PER_REC/4];

  strcpy (filename ,"loadfile.dxe");          /* default file names */
  strcpy (ofilename,"loadfile.srom");
     strcpy (hfilename,"loadfile.hex");
  base = 0;
  tag = 0;
  asn = 0;
  asm = 1;
  tphysical= 1;
  bhtvector = 0;
  offset = 0;

/* for PCA I added 55 bits of padding. One of those bits is reflected in the above
numbers. */

  for (i=0; i<128; i++)
     dfillmap[i]+=54;
  for (i=0; i<8; i++)
     BHTfillmap[i]+=54;
  for (i=0; i<20; i++)
     predfillmap[i]+=54;
  octawpfillmap+=54;
  predpfillmap+=54;
  for (i=0; i<29; i++)
     tagfillmap[i]+=54;
  for (i=0; i<7; i++)
     asnfillmap[i]+=54;
  asmfillmap+=54;
  tagphysfillmap+=54;
  for (i=0; i<4; i++)
     tagvalfillmap[i]+=54;
  tagparfillmap+=54;
```

```
 if (argc>1)
    strcpy(filename,argv[1]);
 if (argc>2)
    strcpy(ofilename,argv[2]);
 if (argc>3)
    strcpy(hfilename,argv[3]);
 if (NULL == (infile = fopen(filename,"rb")))
    {
       printf("input file open error: %s\n", filename);
       exit(0);
    }
 if (NULL == (outfile = fopen(ofilename, "wb")))
    {
       printf("binary output file open error: %s\n", ofilename);
       exit(0);
    }
 if (NULL == (hexfile = fopen(hfilename, "w")))
    {
       printf("hex output file open error: %s\n", hfilename);
       exit(0);
    }
 fprintf(hexfile,":020000020000FC\n");         /* extended segment addr record */
 tparity = eparity(tag) ^ eparity(tphysical) ^ eparity(asn);
 tvalids = 15;
 instatus = 0;
 instr_count = 0;
    /* there are 512 full 32 byte records (MAX_INSTR instructions) */
 for (lines_written = 0; lines_written < 512; lines_written++)
    {
       build_vector(instr, outvector, &instatus, &instr_count); /* build the vector */
       fwrite(&outvector[0],1,DATA_BYTES_PER_REC,outfile);      /* print it to a
       binary file */
       fprintf(hexfile,":19%04X00",offset);                     /* print it to the hex
       file */
       chksum = (offset & 0xff) + (offset >> 8) + 0x19;
       for (j=0; j<DATA_BYTES_PER_REC; j++)
{
 charptr = ((char*) &outvector[0]) + j;
 fprintf(hexfile,"%02X", (0xff& *charptr));
 chksum += *charptr;
}
    offset += DATA_BYTES_PER_REC;
    fprintf(hexfile,"%02X\n", (-chksum) & 0xff);
 }
if (instatus == 0)  /* there’s more data in the file to read, oops... */
 {
    while (instatus == 0){
```

```
build_vector(instr, outvector, &instatus, &instr_count); /* build the vector */
    }
    if (instr_count > MAX_INSTR){
printf("\nev5fmt Warning: input file too long.\n");
printf("\tThere are %d instructions in the input file\n", instr_count);
printf("\tTruncated after %d instructions\n\n", instr_count, MAX_INSTR);
    }
 }

fprintf(hexfile,":15%04X00",offset);
chksum = (offset & 0xff) + (offset >> 8) + 0x15;
for (j=0; j < 17; j++)                                  /* special case, last record */
 {
    fprintf(hexfile,"%02X", (0x00));
    chksum += *charptr;
 }
for (j=0; j < 4; j++)  /* Put 4 bytes of FF at the end... */
 {
    fprintf(hexfile, "%02X", 0xff);
    chksum += 0xff;
 }

fprintf(hexfile,"%02X\n", (-chksum) & 0xff);

fprintf(hexfile,":00000001FF\n");          /* end-of-file record */
printf("Total intructions processed = %d\t(%d free)\n",
 instr_count, MAX_INSTR - instr_count);
 fclose(infile);
 fclose(outfile);
 fclose(hexfile);
 exit(0);
}

void build_vector(int instr[], int outvector[], int *instatus, int *instr_count)
{
    int j, k, t;
    int status;

       for (j=0;j<4;j++) instr[j] = 0;
       for (j=0;j<DATA_BYTES_PER_REC/4;j++) outvector[j]=0;

       if (*instatus == 0)              /* read until the file's done */
 {
    /* read-in 4 instructions */
    if (16 > (status = fread(&instr[0],1,16,infile)))
       *instatus = 1;             /* we're done now             */
    *instr_count += status/4;
 }

       predecodes=0;
       owparity = 0;
       for (j=0;j<4;j++)
 {
    predecodes |= (4 ^ instrpredecode(instr[j])) << (j*5);
    /* invert bit 2 to match fill scan chain attribute */
    owparity ^= eparity(instr[j]);
}
       pdparity = eparity(predecodes);
       /* bhtvector */
       for (j=0;j<7;j++)
{
    t = BHTfillmap[j];
    outvector[t>>5] |= ((bhtvector >> j) & 1) << (t&0x1f);
}
    /* instructions */
    for (k=0;k<4;k++)
{
 for (j=0;j<32;j++)
  {
    t = dfillmap[j+k*32];
    outvector[t>>5] |= ((instr[k] >> j) & 1) << (t&0x1f);
  }
}
    /* predecodes */
    for (j=0;j<20;j++)
{
 t = predfillmap[j];
 outvector[t>>5] |= ((predecodes >> j) & 1) << (t&0x1f);
}
    /* owparity */
    outvector[octawpfillmap>>5] |= owparity << (octawpfillmap&0x1f);
    /* pdparity */
    outvector[predpfillmap>>5] |= pdparity << (predpfillmap&0x1f);
    /* tparity */
    outvector[tagparfillmap>>5] |= tparity << (tagparfillmap&0x1f);
    /* tvalids */
    for (j=0;j<4;j++)
{
 t =tagvalfillmap[j];
 outvector[t>>5] |= ((tvalids >> j) & 1) << (t&0x1f);
}
    /* tphysical */
    outvector[tagphysfillmap>>5] |=  tphysical << (tagphysfillmap&0x1f);
    /* asn */
    for (j=0;j<7;j++)
{
 t = asnfillmap[j];
 outvector[t>>5] |= ((asn >> j) & 1) << (t&0x1f);
}
```

```
    /* asm */
    outvector[asmfillmap>>5] |=  asm << (asmfillmap&0x1f);
    /* tag */
    for (j=0;j<29;j++)
{
 t = tagfillmap[j];
 outvector[t>>5] |= ((tag >> j) & 1) << (t&0x1f);
}
}
int eparity(int x)
{
 x = x ^ (x >> 16);
 x = x ^ (x >> 8);
 x = x ^ (x >> 4);
 x = x ^ (x >> 2);
 x = x ^ (x >> 1);
 return (x&1);
}
#define EXT(data, bit)\
    (((data) & ((unsigned) 1 << (bit))) != 0)
#define EXTV(data, hbit, lbit)\
    (((data) >> (lbit)) & \
       ((((hbit) - (lbit) + 1) == 32) ? ((unsigned)0xffffffff) :
       (~((unsigned)0xffffffff << ((hbit) - (lbit) + 1)))))
#define INS(name, bit, data)\
 (name) = (((name) & ~((unsigned) 1 << (bit))) | \
    (((unsigned) (data) << (bit)) & ((unsigned) 1 << (bit))))


int instrpredecode(int inst)
{
int result;
int opcode;
int func;
int jsr_type;
int ra;
int out0;
int out1;
int out2;
int out3;
int out4;
int e0_only;
int  e1_only;
int  ee;
int lnoop;
int  fadd;
int  fmul;
int  fe;
int  br_type;
int  ld;
int  store;
int  br;
int  call_pal;
int  bsr;
int  ret_rei;
int  jmp;
int  jsr_cor;
int  jsr;
int  cond_br;

 opcode = EXTV(inst, 31, 26 );
 func   = EXTV(inst, 12, 5);
 jsr_type = EXTV(inst, 15,14);
 ra = EXTV(inst,25,21);


e0_only  = (opcode == 0x24) ||      /* STF */
    (opcode == 0x25) ||      /* STG */
    (opcode == 0x26) ||      /* STS */
    (opcode == 0x27) ||      /* STT */
    (opcode == 0x0F) ||      /* STQ_U */
    (opcode == 0x2A) ||      /* LDL_L */
    (opcode == 0x2B) ||      /* LDQ_L */
    (opcode == 0x2C) ||      /* STL */
    (opcode == 0x2D) ||      /* STQ */
    (opcode == 0x2E) ||      /* STL_C */
    (opcode == 0x2F) ||      /* STQ_C */
    (opcode == 0x1F) ||      /* HW_ST*/
    (opcode == 0x18) ||      /* MISC mem format: FETCH/_M, RS, RC, RPCC, TRAPB, MB) */
    (opcode == 0x12) ||      /* EXT,MSK,INS,SRX,SLX,ZAP*/
    (opcode == 0x13) ||      /* MULX */
    ((opcode == 0x1D) && (EXT(inst,8) == 0)) || /* MBOX HW_MTPR */
    ((opcode == 0x19) && (EXT(inst,8) == 0)) || /* MBOX HW_MFPR */
    (opcode == 0x01) ||             /* VR::: might change this later RESDEC's */
    (opcode == 0x02) ||             /* RESDEC's */
    (opcode == 0x03) ||             /* RESDEC's */
    (opcode == 0x04) ||             /* RESDEC's */
    (opcode == 0x05) ||             /* RESDEC's */
    (opcode == 0x06) ||             /* RESDEC's */
    (opcode == 0x07) ||             /* RESDEC's */
    (opcode == 0x0a) ||             /* RESDEC's */
    (opcode == 0x0c) ||             /* RESDEC's */
    (opcode == 0x0d) ||      /* RESDEC's */
    (opcode == 0x0e) ||      /* RESDEC's */
    (opcode == 0x14) ||      /* RESDEC's */
    (opcode == 0x1c);      /* RESDEC's */

e1_only =  (opcode == 0x30) ||      /* BR */
    (opcode == 0x34) ||      /* BSR */
    (opcode == 0x38) ||      /* BLBC */
    (opcode == 0x39) ||      /* BEQ */
    (opcode == 0x3A) ||      /* BLT */
    (opcode == 0x3B) ||      /* BLE */
    (opcode == 0x3C) ||      /* BLBS */
    (opcode == 0x3D) ||      /* BNE */
    (opcode == 0x3E) ||      /* BGE */
    (opcode == 0x3F) ||      /* BGT */
    (opcode == 0x1A) ||      /* JMP,JSR,RET,JSR_COROT */
    (opcode == 0x1E) ||      /* HW_REI */
    (opcode == 0x00) ||      /* CALL_PAL */
    ((opcode == 0x1D) && (EXT(inst,8) == 1)) || /* IBOX HW_MTPR */
    ((opcode == 0x19) && (EXT(inst,8) == 1));   /* IBOX HW_MTPR */

ee =       (opcode == 0x10) ||      /* ADD, SUB, CMP */
    (opcode == 0x11) ||      /* AND, BIC etc. logicals */
    (opcode == 0x28) ||      /* LDL */
    (opcode == 0x29) ||      /* LDQ */
    (opcode == 0x0B)&(ra != 0x1F) ||  /* LDQ_U */
    (opcode == 0x08) ||      /* LDA */
    (opcode == 0x09) ||      /* LDAH */
    (opcode == 0x20) ||      /* LDF */
    (opcode == 0x21) ||      /* LDG */
    (opcode == 0x22) ||      /* LDS */
    (opcode == 0x23) ||      /* LDT */
    (opcode == 0x1B);      /* HW_LD */

lnoop =    (opcode == 0x0B)&(ra == 0x1F);    /* LDQ_U R31, x(y) - NOOP*/
fadd =    ((opcode == 0x17) && (func != 0x20)) ||    /* Flt, datatype indep excl CPYS*/
    ((opcode == 0x15) && ((func & 0xf) != 0x2)) ||   /* VAX excl MUL's */
    ((opcode == 0x16) && ((func & 0xf) != 0x2)) ||   /* IEEE excl MUL's */
    (opcode == 0x31) ||      /* FBEQ */
    (opcode == 0x32) ||      /* FBLT */
    (opcode == 0x33) ||      /* FBLE */
    (opcode == 0x35) ||      /* FBNE */
    (opcode == 0x36) ||      /* FBGE */
    (opcode == 0x37);      /* FBGT */

fmul =     ((opcode == 0x15) && ((func & 0xf) == 0x2)) ||   /* VAX MUL's */
    ((opcode == 0x16) && ((func & 0xf) == 0x2));     /* IEEE MUL's */
fe =       ((opcode == 0x17) && (func == 0x20));    /*  CPYS */
br_type =  ((opcode & 0x30) == 0x30) ||   /* all branches */
    (opcode == 0x1A) ||    /* JMP's */
    (opcode == 0x00) ||    /* CALL PAL */
    (opcode == 0x1E);    /* HW_REI */

ld =    (opcode == 0x28) ||    /* LDL */
    (opcode == 0x29) ||    /* LDQ */
    (opcode == 0x0B) ||    /* LDQ_U */
    (opcode == 0x20) ||    /* LDF */
    (opcode == 0x21) ||    /* LDG */
    (opcode == 0x22) ||    /* LDS */
    (opcode == 0x23) ||    /* LDT */
    (opcode == 0x1B);    /* HW_LD */

store =    (opcode == 0x24) ||    /* STF */
    (opcode == 0x25) ||    /* STG */
    (opcode == 0x26) ||    /* STS */
    (opcode == 0x27) ||    /* STT */
    (opcode == 0x0F) ||    /* STQ_U */
    (opcode == 0x2C) ||    /* STL */
    (opcode == 0x2D) ||    /* STQ */
    (opcode == 0x2E) ||    /* STL_C */
    (opcode == 0x2F) ||    /* STQ_C */
    (opcode == 0x18) ||    /* Misc: TRAPB, MB, RS, RC, RPCC etc. */
          (opcode == 0x1F) ||    /* HW_ST */
    (opcode == 0x2A) ||    /* LDL_L */
    (opcode == 0x2B);    /* LDQ_L */

br =    (opcode == 0x30);    /* all branches */

call_pal = (opcode == 0x00);    /* call PAL */

bsr =    (opcode == 0x34);

ret_rei =  ((opcode == 0x1A) && (jsr_type == 0x2)) ||
    ((opcode == 0x1E) && (jsr_type != 0x3));

jmp = ((opcode == 0x1A) && (jsr_type == 0x0));

jsr_cor = ((opcode == 0x1A) && (jsr_type == 0x3));

jsr = ((opcode == 0x1A) && (jsr_type == 0x1));

cond_br = (opcode == 0x31) ||
    (opcode == 0x32) ||
    (opcode == 0x33) ||
    (opcode == 0x35) ||
    (opcode == 0x36) ||
    (opcode == 0x37) ||
    (opcode == 0x38) ||
    (opcode == 0x39) ||
    (opcode == 0x3A) ||
    (opcode == 0x3B) ||
    (opcode == 0x3C) ||
    (opcode == 0x3D) ||
    (opcode == 0x3E) ||
    (opcode == 0x3F);
```

```
out0 = br || bsr || jmp || jsr || (ee && !ld) || (e0_only && !store);
out1 = ret_rei ||(e1_only && !br_type)|| jmp ||jsr_cor|| jsr || lnoop || (fadd &&
!br_type) || fe;;
out2 = call_pal || bsr || jsr_cor || e0_only ||jsr ||fmul || fe;
out3 = (e1_only && cond_br) || (e1_only && !br_type) || fadd || fmul || fe;
out4 = ee || lnoop || e0_only || fadd || fmul || fe;

 result = 0;
 INS( result, 0, out0 );
 INS( result, 1, out1 );
 INS( result, 2, out2 );
 INS( result, 3, out3 );
 INS( result, 4, out4 );

 return (result);

 }
```

D

Support, Products, and Documentation

D.1 Customer Support

Alpha OEM provides the following web page resources for customer support.

URL	Description
http://www.digital.com/alphaoem	Contains the following links: • **Developers' Area:** Development tools, code examples, driver developers' information, and technical white papers • **Motherboard Products:** Motherboard details and performance information • **Microprocessor Products:** Microprocessor details and performance information • **News:** Press releases • **Technical Information:** Motherboard firmware and drivers, hardware compatibility lists, and product documentation library • **Customer Support:** Feedback form

D.2 Alpha Products

To order the Alpha 21164PC microprocessor, contact your sales office. The following table lists some of the Alpha products available.

Note: The following products and order numbers might have been revised. For the latest versions, contact your sales office.

Chips	Order Number
Alpha 21164PC 533-MHz microprocessor	211PC–13
Alpha 21164PC 600-MHz microprocessor	211PC–15
Alpha 21164PC 666-MHz microprocessor	211PC–17

D.3 Alpha Documentation

The following table lists some of the available Alpha documentation. You can download Alpha documentation from the Alpha OEM World Wide Web Internet site:

http://www.digital.com/alphaoem

Title	Order Number
AlphaArchitecture Reference Manual[1]	EY–W938E–DP
Alpha Architecture Handbook	EC–QD2KB–TE
Alpha 21164PC Microprocessor Product Brief	EC–R2W2B–TE
21172 Core Logic Chipset Product Brief	EC–QUQHA–TE
21172 Core Logic Chipset Technical Reference Manual	EC–QUQJA–TE
Answers to Common Questions about PALcode for Alpha AXP Systems	EC–N0647–72
PALcode for Alpha Microprocessors System Design Guide	EC–QFGLC–TE
Alpha Microprocessors Motherboard Windows NT 3.51 and 4.0 Installation Guide	EC–QLUAH–TE
SPICE Models for Alpha Microprocessors: An Application Note	EC–QA4XG–TE

Title	Order Number
Alpha Microprocessors SROM Mini-Debugger User’s Guide	EC–QHUXC–TE
Alpha Microprocessors Motherboard Debug Monitor User’s Guide	EC–QHUVF–TE
Alpha Microprocessors Motherboard Software Design Tools User’s Guide	EC–QHUWD–TE

[1] To purchase the *Alpha Architecture Reference Manual*, contact your sales office or call Butterworth-Heinemann (Digital Press) at 1-800-366-2665.

If you have feedback about the Alpha technical documentation, please send your comments to **alpha.techdoc@compaq.com**.

D.4 Third–Party Documentation

You can order the following third-party documentation directly from the vendor.

Title	Vendor
PCI Local Bus Specification, Revision 2.1 PCI System Design Guide	PCI Special Interest Group U.S. 1–800–433–5177 International 1–503–797–4207 Fax 1–503–234–6762
IEEE Standard 754, Standard for Binary Floating-Point Arithmetic IEEE Standard 1149.1, A Test Access Port and Boundary Scan Architecture	The Institute of Electrical and Electronics Engineers, Inc. U.S. 1–800–701–4333 International 1–908–981–0060 Fax 1–908–981–9667

Glossary

The glossary defines terms and spells out acronyms associated with the Alpha 21164PC microprocessor and chips in general.

abort

The unit stops the operation it is performing, without saving status, to perform some other operation.

ABT

Advanced bipolar/CMOS technology.

address space number (ASN)

An optionally implemented register used to reduce the need for invalidation of cached address translations for process-specific addresses when a context switch occurs. ASNs are processor specific; the hardware makes no attempt to maintain coherency across multiple processors.

address translation

The process of mapping addresses from one address space to another.

ALIGNED

A datum of size 2^N is stored in memory at a byte address that is a multiple of 2^N (that is, one that has N low-order zeros).

ALU

Arithmetic logic unit.

ANSI

American National Standards Institute. An organization that develops and publishes standards for the computer industry.

ASIC

Application-specific integrated circuit.

ASN

See address space number.

assert

To cause a signal to change to its logical true state.

AST

See asynchronous system trap.

asynchronous system trap (AST)

A software-simulated interrupt to a user-defined routine. ASTs enable a user process to be notified asynchronously, with respect to that process, of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes execution of the process at the point where it was interrupted.

backmap

A memory unit that is used to note addresses of valid entries within a cache.

bandwidth

Bandwidth is often used to express “high rate of data transfer” in a bus or an I/O channel. This usage assumes that a wide bandwidth may contain a high frequency, which can accommodate a high rate of data transfer.

barrier transaction

A transaction on the external interface as a result of an MB (memory barrier) instruction.

Bcache

See external cache.

BCT

Bipolar/CMOS technology.

BiCMOS

Bipolar/CMOS. The combination of bipolar and MOSFET transistors in a common integrated circuit.

bidirectional

Flowing in two directions. The buses are bidirectional; they carry both input and output signals.

BiSr

Built-in self-repair.

BiSt

Built-in self-test.

bit

Binary digit. The smallest unit of data in a binary notation system, designated as 0 or 1.

BIU

Bus interface unit. *See* CBU.

block exchange

Memory feature that improves bus bandwidth by paralleling a cache victim write-back with a cache miss fill.

board-level cache

See external cache.

boot

Short for bootstrap. Loading an operating system into memory is called booting.

BSR

Boundary-scan register.

buffer

An internal memory area used for temporary storage of data records during input or output operations.

bugcheck

A software condition, usually the response to software's detection of an "internal inconsistency," which results in the execution of the system bugcheck code.

bus

A group of signals that consists of many transmission lines or wires. It interconnects computer system components to provide communications paths for addresses, data, and control information.

byte

Eight contiguous bits starting on an addressable byte boundary. The bits are numbered right to left, 0 through 7.

byte granularity

Memory systems are said to have byte granularity if adjacent bytes can be written concurrently and independently by different processes or processors.

cache

See cache memory.

cache block

The smallest unit of storage that can be allocated or manipulated in a cache. Also known as a cache line.

cache coherence

Maintaining cache coherence requires that when a processor accesses data cached in another processor, it must not receive incorrect data and when cached data is modified, all other processors that access that data receive modified data. Schemes for maintaining consistency can be implemented in hardware or software. Also called cache consistency.

cache fill

An operation that loads an entire cache block by using multiple read cycles from main memory.

cache flush

An operation that marks all cache blocks as invalid.

cache hit

The status returned when a logic unit probes a cache memory and finds a valid cache entry at the probed address.

cache interference

The result of an operation that adversely affects the mechanisms and procedures used to keep frequently used items in a cache. Such interference may cause frequently used items to be removed from a cache or incur significant overhead operations to ensure correct results. Either action hampers performance.

cache line

See cache block.

cache line buffer

A buffer used to store a block of cache memory.

cache memory

A small, high-speed memory placed between slower main memory and the processor. A cache increases effective memory transfer rates and processor speed. It contains copies of data recently used by the processor and fetches several bytes of data from memory in anticipation that the processor will access the next sequential series of bytes. The Alpha 21164PC microprocessor contains two onchip internal caches. *See also* write-through cache and write-back cache.

cache miss

The status returned when cache memory is probed with no valid cache entry at the probed address.

CALL_PAL Instructions

Special instructions used to invoke PALcode.

CBU

Cache control and bus interface unit. The logic unit within the 21164PC microprocessor that provides an interface to the external data bus and board-level Bcache.

central processing unit (CPU)

The unit of the computer that is responsible for interpreting and executing instructions.

CISC

Complex instruction set computing. An instruction set consisting of a large number of complex instructions that are managed by microcode. *Contrast with* RISC.

clean

In the cache of a system bus node, refers to a cache line that is valid but has not been written.

clock

A signal used to synchronize the circuits in a computer.

CMOS

Complementary metal-oxide semiconductor. A silicon device formed by a process that combines PMOS and NMOS semiconductor material.

conditional 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.

CPLD

Complex programmable logic device.

CPU

See central processing unit.

CSR

See control and status register.

cycle

One clock interval.

data bus

The bus used to carry data between the 21164PC and external devices. Also called the pin bus.

Dcache

Data cache. A cache reserved for storage of data. The Dcache does not contain instructions.

DIP

Dual inline package.

direct-mapping cache

A cache organization in which only one address comparison is needed to locate any data in the cache, because any block of main memory data can be placed in only one possible position in the cache.

direct memory access (DMA)

Access to memory by an I/O device that does not require processor intervention.

dirty

One status item for a cache block. The cache block is valid and has been written so that it may differ from the copy in system main memory.

dirty victim

Used in reference to a cache block in the cache of a system bus node. The cache block is valid but is about to be replaced due to a cache block resource conflict. The data must therefore be written to memory.

DRAM

Dynamic random-access memory. Read/write memory that must be refreshed (read from or written to) periodically to maintain the storage of information.

DTL

Diode-transistor logic.

dual issue

Two instructions are issued, in parallel, during the same microprocessor cycle. The instructions use different resources and so do not conflict.

ECC

Error correction code. Code and algorithms used by logic to facilitate error detection and correction. *See also* ECC error.

ECC error

An error detected by ECC logic, to indicate that data (or the protected “entity”) has been corrupted. The error may be correctable (soft error) or uncorrectable (hard error).

ECL

Emitter-coupled logic.

EEPROM

Electrically erasable programmable read-only memory. A memory device that can be byte-erased, written to, and read from. *Contrast with* FEPROM.

EPLD

Erasable programmable logic device.

external cache

A cache memory provided outside of the microprocessor chip, usually located on the same module. Also called board-level or module-level cache.

FEPROM

Flash-erasable programmable read-only memory. FEPROMs can be bank- or bulk-erased. *Contrast with* EEPROM.

FET

Field-effect transistor.

firmware

Machine instructions stored in hardware.

floating point

A number system in which the position of the radix point is indicated by the exponent part and another part represents the significant digits or fractional part.

flush

See cache flush.

FPGA

Field-programmable gate array.

FPLA

Field-programmable logic array.

FPU

Floating-point execution unit. The logic unit within the 21164PC microprocessor that performs floating-point calculations.

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.

Icache

Instruction cache. A cache reserved for storage of instructions. One of the two areas of primary cache (located on the 21164PC) used to store instructions. The Icache contains 16KB of memory space. It is a direct-mapped cache. Icache blocks, or lines, contain 64 bytes of instruction stream data with associated tag as well as a 6-bit ASM field and an 8-bit branch history field per block. Icache does not contain hardware for maintaining cache coherency with memory and is unaffected by the invalidate bus.

IDU

Instruction fetch/decode unit. The logic unit within the 21164PC microprocessor that fetches, decodes, and issues instructions. It also controls the microprocessor pipeline.

IEEE Standard 754

A set of formats and operations that apply to floating-point numbers. The formats cover 32-, 64-, and 80-bit operand sizes.

IEEE Standard 1149.1

A standard for the Test Access Port and Boundary Scan Architecture used in board-level manufacturing test procedures. Commonly referred to as the Joint Test Action Group (JTAG) standard.

IEU

Integer execution unit. The logic unit within the 21164PC microprocessor that contains the 64-bit integer execution data path.

INT*nn*

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

internal processor register (IPR)

One of many registers internal to the Alpha 21164PC microprocessor.

IPGA

Interstitial pin grid array.

JFET

Junction field-effect transistor.

latency

The amount of time it takes the system to respond to an event.

LCC

Leadless chip carrier.

LFSR

Linear feedback shift register.

load/store architecture

A characteristic of a machine architecture where data items are first loaded into a processor register, operated on, and then stored back to memory. No operations on memory other than load and store are provided by the instruction set.

longword

Four contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 31.

LSB

Least significant bit.

LSI

Large-scale integration.

machine check

An operating system action triggered by certain system hardware-detected errors that can be fatal to system operation. Once triggered, machine check handler software analyzes the error.

MAF

Miss address file.

main memory

The large memory, external to the microprocessor, used for holding most instruction code and data. Usually built from cost-effective DRAM memory chips. May be used in connection with the microprocessor's internal caches and an optional external cache.

masked write

A write cycle that only updates a subset of a nominal data block.

MBO

See must be one.

MBZ

See must be zero.

MESI protocol

A cache consistency protocol with full support for multiprocessing. The MESI protocol consists of four states that define whether a block is modified (M), exclusive (E), shared (S), or invalid (I).

MIPS

Millions of instructions per second.

miss

See cache miss.

module

A board on which logic devices (such as transistors, resistors, and memory chips) are mounted and connected to perform a specific system function.

module-level cache

See external cache.

MOS

Metal-oxide semiconductor.

MOSFET

Metal-oxide semiconductor field-effect transistor.

MSI

Medium-scale integration.

MTU

Memory address translation unit. The logic unit within the 21164PC microprocessor that performs address translation, interfaces to the Dcache, and performs several other functions.

multiprocessing

A processing method that replicates the sequential computer and interconnects the collection so that each processor can execute the same or a different program at the same time.

Must be one (MBO)

A field that must be supplied as one.

Must be zero (MBZ)

A field that is reserved and must be supplied as zero. If examined, it must be assumed to be UNDEFINED.

NATURALLY ALIGNED

See ALIGNED.

NATURALLY ALIGNED data

Data stored in memory such that the address of the data is evenly divisible by the size of the data in bytes. For example, an ALIGNED longword is stored such that the address of the longword is evenly divisible by 4.

NMOS

N-type metal-oxide semiconductor.

NVRAM

Nonvolatile random-access memory.

OBL

Observability linear feedback shift register.

octaword

Sixteen contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 127.

OpenVMS Alpha operating system

The open version of the COMPAQ VMS operating system, which runs on Alpha platforms.

operand

The data or register upon which an operation is performed.

PAL

Privileged architecture library (software). *See also* PALcode.

Programmable array logic (hardware). *See* programmable array logic.

PALcode

Alpha privileged architecture library code, written to support Alpha microprocessors. PALcode implements architecturally defined behavior.

PALmode

A special environment for running PALcode routines.

parameter

A variable that is given a specific value that is passed to a program before execution.

parity

A method for checking the accuracy of data by calculating the sum of the number of ones in a piece of binary data. Even parity requires the correct sum to be an even number. Odd parity requires the correct sum to be an odd number.

PGA

Pin grid array.

pipeline

A CPU design technique whereby multiple instructions are simultaneously overlapped in execution.

PLA

Programmable logic array.

PLCC

Plastic leadless chip carrier or plastic-leaded chip carrier.

PLD

Programmable logic device.

PLL

Phase-locked loop.

PMOS

P-type metal-oxide semiconductor.

PQFP

Plastic quad flat pack.

primary cache

The cache that is the fastest and closest to the processor. The first-level caches, located on the CPU chip, composed of the Dcache and Icache.

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.

programmable array logic (PAL)

A device that can be programmed by a process that blows individual fuses to create a circuit.

PROM

Programmable read-only memory.

pull-down resistor

A resistor placed between a signal line and a negative voltage.

pull-up resistor

A resistor placed between a signal line to a positive voltage.

quad issue

Four instructions are issued, in parallel, during the same microprocessor cycle. The instructions use different resources and so do not conflict.

quadword

Eight contiguous bytes starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 63.

RAM

Random-access memory.

READ BLOCK

A transaction where the 21164PC 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 21164PC and external hardware.

read stream buffers

Arrangement whereby each memory module independently prefetches DRAM data prior to an actual read request for that data. Reduces average memory latency while improving total memory bandwidth.

register

A temporary storage or control location in hardware logic.

reliability

The probability a device or system will not fail to perform its intended functions during a specified time interval when operated under stated conditions.

reset

An action that causes a logic unit to interrupt the task it is performing and go to its initialized state.

RISC

Reduced instruction set computing. A computer with an instruction set that is paired down and reduced in complexity so that most instructions can be performed in a single processor cycle. High-level compilers synthesize the more complex, least frequently-used instructions by breaking them down into simpler instructions. This approach allows the RISC architecture to implement a small, hardware-assisted instruction set, thus eliminating the need for microcode.

ROM

Read-only memory.

RTL

Register-transfer logic.

SAM

Serial access memory.

SBO

Should be one.

SBZ

Should be zero.

scheduling

The process of ordering instruction execution to obtain optimum performance.

set-associative

A form of cache organization in which the location of a data block in main memory constrains, but does not completely determine, its location in the cache. Set-associative organization is a compromise between direct-mapped organization, in which data from a given address in main memory has only one possible cache location, and fully associative organization, in which data from anywhere in main memory can be put anywhere in the cache. An “*n*-way set-associative” cache allows data from a given address in main memory to be cached in any of *n* locations.

SIMM

Single inline memory module.

SIP

Single inline package.

SIPP

Single inline pin package.

SMD

Surface mount device.

SRAM

Static random-access memory.

SROM

Serial read-only memory.

SSI

Small-scale integration.

SSRAM

Synchronous static random-access memory.

stack

An area of memory set aside for temporary data storage or for procedure and interrupt service linkages. A stack uses the last-in/first-out concept. As items are added to (pushed on) the stack, the stack pointer decrements. As items are retrieved from (popped off) the stack, the stack pointer increments.

STRAM

Self-timed random-access memory.

superpipelined

Describes a pipelined machine that has a larger number of pipe stages and more complex scheduling and control. *See also* pipeline.

superscalar

Describes a machine architecture that allows multiple independent instructions to be issued in parallel during a given clock cycle.

tag

The part of a cache block that holds the address information used to determine if a memory operation is a hit or a miss on that cache block.

TB

Translation buffer.

tristate

Refers to a bused line that has three states: high, low, and high-impedance.

TTL

Transistor-transistor logic.

UART

Universal asynchronous receiver-transmitter.

UNALIGNED

A datum of size 2^N stored at a byte address that is not a multiple of 2^N.

unconditional branch instructions

Instructions that write a return address into a register.

UNDEFINED

An operation that may halt the processor or cause it to lose information. Only privileged software (that is, software running in kernel mode) can trigger an UNDEFINED operation.

UNPREDICTABLE

Results or occurrences that do not disrupt the basic operation of the processor; the processor continues to execute instructions in its normal manner. Privileged or unprivileged software can trigger UNPREDICTABLE results or occurrences.

UVPROM

Ultraviolet (erasable) programmable read-only memory.

valid

Allocated. Valid cache blocks have been loaded with data and may return cache hits when accessed.

VHSIC

Very-high-speed integrated circuit.

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.

VLSI

Very-large-scale integration.

VRAM

Video random-access memory.

word

Two contiguous bytes (16 bits) starting on an arbitrary byte boundary. The bits are numbered from right to left, 0 through 15.

write-back

A cache management technique in which write operation data is written into cache but is not written into main memory in the same operation. This may result in temporary differences between cache data and main memory data. Some logic unit must maintain coherency between cache and main memory.

write-back cache

Copies are kept of any data in the region; read and write operations may use the copies, and write operations use additional state to determine whether there are other copies to invalidate or update.

WRITE BLOCK

A transaction in which the 21164PC requests that an external logic unit process write data.

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 21164PC 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 during the same operation.

write-through cache

Copies are kept of any data in the region; read operations may use the copies, but write operations update the actual data location and either update or invalidate all copies.

Index