DECchip 21164-AA (EV5 CPU) Functional Specification

The DECchip 21164-AA CPU Chip is a high-performance, single-chip implementation of the Alpha Architecture.

Revision/Update Information: This is Revision 1.9 of this specification. Relative to Revision 1.5, all chapters are updated. This is the last release expected prior to tape-out.

Product Manager: John Fortune, RICKS::FORTUNE
Engineering Manager: Paul Rubinfeld, ROCK::RUBINFELD

Semiconductor Engineering Group
Digital Equipment Corporation, Hudson, Massachusetts

updated to Oct. 93

December 1992

Contents

FIGURES

TABLES

Table 1: REVISION History

Revision	Date	Description
1.0	2-March-1992	Initial release.
1.5	8-May-1992	Updates including some significant changes.
1.9	21-December-1992	Further updates.

Chapter 1

Introduction

1.1 Scope

This document describes the DECchip 21164-AA chip, a microprocessor that implements the Alpha architecture. This specification describes the external interface and programming information specific to the actual implementation. It does not describe the detailed implementation of the chip nor the Alpha architecture. The reader is referred to the Alpha System Reference Manual for the architectural specification.

1.2 Chip Features

The DECchip 21164-AA microprocessor is a CMOS-5 (.5 micron) super-scalar super-pipelined implementation of the Alpha architecture. It will be the basis of a family of Alpha products. The DECchip 21164-AA chip is designed to meet the requirements of a wide variety of systems, ranging from uni-processor workstations to multiprocessors. DECchip 21164-AA is intended to integrate well into a certain style of system environment, one with a particular kind of cache coherence protocol and a pipelined or lock-step style of bus and memory subsystem operation. A number of configuration options allow its use in a range of system designs ranging from extremely simple systems with minimum component count to high-performance systems with very high cache and memory bandwidth. DECchip 21164-AA design compromises are made with the intention of achieving maximum performance in high-performance systems while offering competitive performance and reasonable implementation constraints in lower cost systems.

DECchip 21164-AA features:

- Alpha instructions to support byte, word, longword, quadword, DEC F_floating, G_floating and IEEE S_floating and T_floating data types. Limited support is provided for DEC D_ floating operations. Partial implementation of the architecturally optional instructions: FETCH and FETCH_M.
- Demand paged memory management unit which in conjunction with properly written PALcode fully implements the Alpha memory management architecture appropriate to the operating system running on the processor. The translation buffer can be used with alternative PALcode to implement a variety of page table structures and translation algorithms.
- On-chip 48-entry I-stream TB and 64-entry D-stream TB in which each entry maps one 8Kbyte page or a group of 8, 64, or 512 8Kbyte pages, with the size of each TB entry's group specified by hint bits stored in the entry.

- World class performance.
- Low average cycles per instructions (CPI). The DECchip 21164-AA chip can issue four Alpha instructions in a single cycle, thereby minimizing the average CPI. A number of low-latency and/or high-throughput features in the instruction issue unit and the on-chip components of the memory subsystem further reduce the average CPI.
- On-chip high-throughput floating point units, capable of executing both DEC and IEEE floating point data types.
- On-chip 8Kbyte virtual instruction cache with seven-bit ASNs (MAX_ASN=127).
- On-chip dual-read-ported 8Kbyte data cache (implemented as two 8Kbyte data caches containing identical data).
- On-chip write buffer with six 32-byte entries.
- On-chip 96Kbyte 3-way set associative writeback second level cache.
- Bus interface unit, which contains logic to directly access an optional external third-level cache without CPU module action. The size and access time of the external third-level cache is programmable.
- On-chip performance counters to measure and analyze CPU and system performance.
- An instruction cache diagnostic interface to support chip and module level testing.
- An internal clock generator which generates both a high-speed clock needed by the chip itself, and a pair of system clocks for use by the CPU module.
- The DECchip 21164-AA chip is packaged in 503 pin IPGA packages. The heat sinks are separable and application specific.

1.3 Terminology and Conventions

1.3.1 Numbering

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

1.3.2 UNPREDICTABLE And UNDEFINED

Throughout this specification, the terms UNPREDICTABLE and UNDEFINED are used. Their meanings are quite different and must be carefully distinguished. One key difference is that only privileged software (that is, software running in kernel mode) may trigger UNDEFINED operations, whereas either privileged or unprivileged software may trigger UNPREDICTABLE results or occurrences. A second key difference is that UNPREDICTABLE results and occurrences do not disrupt the basic operation of the processor; the processor continues to execute instructions in its normal manner. In contrast, UNDEFINED operation may halt the processor or cause it to lose information.

A result specified as UNPREDICTABLE may acquire an arbitrary value subject to a few constraints. Such a result may be an arbitrary function of the input operands or of any state information that is accessible to the process in its current access mode. UNPREDICTABLE results may be unchanged from their previous values. UNPREDICTABLE results must not be security holes. Specifically, UNPREDICTABLE results must not do any of the following:

- Depend on or be a function of the contents of memory locations or registers which are inaccessible to the current process in the current access mode.
- 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.

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

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

Operations specified as UNDEFINED may vary from moment to moment, implementation to implementation, and instruction to instruction within implementations. The operation may vary in effect from nothing to stopping system operation. UNDEFINED operations must not cause the processor to hang, i.e., 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.

1.3.3 Data Field Size

The term INTnn, where nn is one of 2, 4, 8, 16, 32, or 64, refers to a data field of nn contiguous naturally aligned bytes. INT4 refers to a naturally aligned longword, for example.

1.3.4 Ranges And Extents

Ranges are specified by a pair of numbers separated by a ".." and are inclusive, e.g., a range of integers 0..4 includes the integers 0, 1, 2, 3, and 4.

Extents are specified by a pair of numbers in angle brackets separated by a colon and are inclusive, e.g., bits <7:3> specify an extent of bits including bits 7, 6, 5, 4, and 3.

1.3.5 Register Format Notation

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

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

Table 1–1: Register Field Type Notation

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

In addition to named fields in registers, other bits of the register may be labeled with one of the four symbols listed in Table 1–2. These symbols denote the type of the unnamed fields in the register.

Table 1–2: Register Field Notation

Notation	Description
RAZ	Fields specified as Read As Zero (RAZ) return a zero when read.
RAO	Fields specified as Read As One (RAO) return a one when read.
IGN	Fields specified as Ignore (IGN) are ignored when written and UNPREDICTABLE when read if not otherwise specified.
MBZ	Fields specified as Must Be Zero (MBZ) must never be filled by software with a non-zero value. If the processor encounters a non-zero value in a field specified as MBZ, a Reserved Operand exception occurs.
SBZ	Fields specified as Should Be Zero (SBZ) should be filled by software with a zero value. These fields may be used at some future time. Non-zero values in SBZ fields produce UNPREDICTABLE results.

1.4 Chip Summary

Table 1–3: DECchip 21164-AA Chip Summary and Micro-architecture

Feature	Description
Estimated Cycle Time Range	4.4ns to 3.2ns†
Product Speed Bin Points	**To Be Determined**
Process Technology	CMOS5 (0.5 micron CMOS) and CMOS5S (TBD micron CMOS)
Transistor count	
Die Size	
Package	503 pin IPGA (interstitial pin grid array)
No. Chip Pads	581
No. Signal Pins	289
Typ Maximum Power Dissipation	approx. 60W @ 3.5ns cycles, Vdd=3.45V‡
Clocking input	two times the internal clock speed. E.g., 571.4 Mhz at a 3.5ns cycle time.
Virtual address size	43 bits
Physical address size	40 bits
Page size	8Kbytes
Issue rate	4 instructions per cycle
Integer Pipeline	7 stage
Floating Pipeline	9 stage
On-chip Dcache	8Kbyte, physical, direct-mapped, write-thru, 32-byte block, 32-byte fill
On-chip Icache	8Kbyte, virtual, direct-mapped, 32-byte block, 32-byte fill, 128 ASNs (MAX_ASN=127)
On-chip Scache	96Kbyte, physical, 3-way set associative, writeback, 32 or 64-byte block, 32 or 64-byte fill
On-chip DTB	64-entry, fully-associative, NLU replacement, 8K pages, 128 ASNs (MAX_ASN=127), full granularity hint support
On-chip ITB	48-entry, fully-associative, NLU replacement, 128 ASNs (MAX_ASN=127), full granularity hint support
FPU	On-chip FPU supports both IEEE and DEC floating point
Bus	Separate data and address bus. 128-bit
Serial ROM Interface	Allows the chip to access a serial ROM

†This range should not be interpreted as implying any particular production speed bin point. Speed bin ranges will not be known until characterization of production CMOS5 parts has been completed. The highest performance system designs should be designed to accept 3.2ns DECchip 21164-AA parts, though it is not known if or when production parts that fast will be available.

‡Power consumption scales linearly wih frequency over the frequency range 225Mhz to 312Mhz.

1.5 Revision History

Table 1–4: Revision History

Who	When	Description of change
JHE	9-Feb-1992	Initial version.
JHE	1-March-1992	Add chip summary. Initial release.
JHE	29-November-1992	Updates for new revision.

Chapter 2

DECchip 21164-AA Micro-Architecture

2.1 Introduction

This chapter gives a programmer and system designer a view of the DECchip 21164-AA micro-architecture. It is intended to be sufficient for almost all purposes. More detailed hardware descriptions of the chip exist in the internal specification and the behavioral model.

DECchip 21164-AA can issue four instructions in a single cycle. Scheduling and issue rules are given at the end of the chapter. DECchip 21164-AA is a pipelined CPU with 4 Ibox[1] stages, 3 integer operate stages and 4 floating point operate stages. The pipeline is presented later in this chapter.

The combination of DECchip 21164-AA and its PALcode implements the Alpha architecture. Parts of the hardware design assume specific PAL functionality. This functionality is described in the next chapter. If a certain piece of hardware is "architecturally incomplete", the missing functionality must be implemented in PALcode.

2.2 Overview

The DECchip 21164-AA microprocessor consists of five functional units:

- The Ibox fetches, decodes, and issues instructions. It manages the pipelines (data bypassing), the PC, instruction caching (Icache), prefetching, and instruction stream memory management. It also contains interrupt and trap handling hardware.
- The Ebox contains the two integer execution units which execute all integer instructions. It also partially executes all memory instructions by calculating the effective address, if there is one.
- The Mbox processes all load and store operations after the Ebox produces the address. It implements data stream memory management, executes loads, stores, the memory barrier instruction, and some other instructions. It manages outstanding load misses, the write buffer, and the data cache (Dcache). It enforces any reference ordering required for correct operation or by the Alpha shared memory model. It also buffers physical instruction stream requests sent by the Ibox.

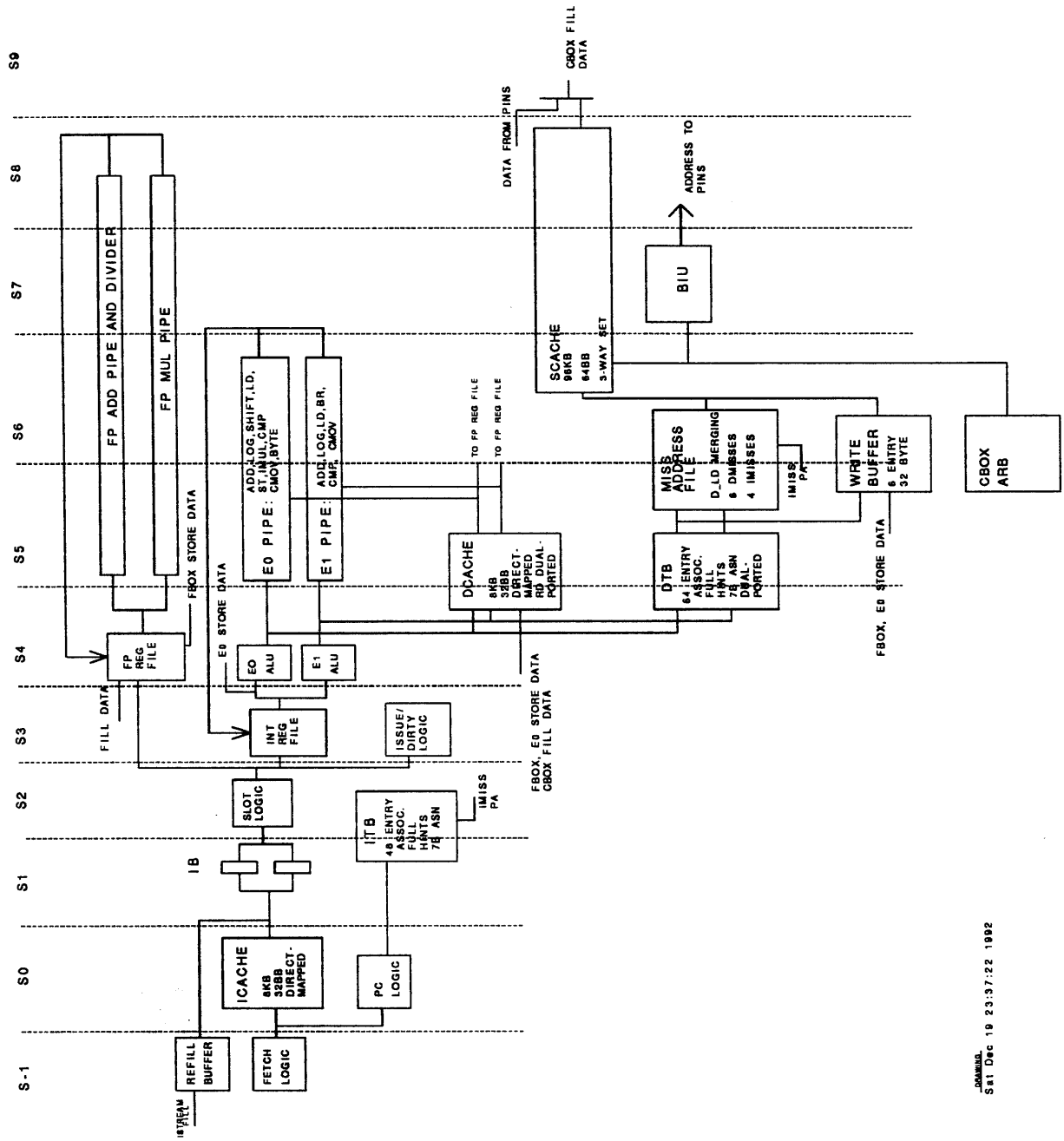
[1] The Ibox is the unit which fetches, decodes and issues instructions.

- The Cbox processes all accesses sent by the Mbox and implements all memory related external interface functions, particularly the coherence protocol functions for writeback caching. It controls the Scache, a 96 Kbyte, 3 way set-associative, writeback, data and instruction cache. The Cbox also manages the optional external direct mapped Bcache. It handles all instruction and data primary cache read misses, performs the function of writing data from the write buffer into the shared coherent memory subsystem, and has a major role in executing the memory barrier instruction.
- The Fbox contains the two floating point execution units, one which basically executes floating multiply instructions and another which executes all other floating point instructions, particularly floating point add and subtract. Both units execute the CPYS instruction.

The Ebox and Fbox can each accept one or two instructions per cycle. If code is properly scheduled, DECchip 21164-AA can issue up to four instructions per cycle.

Figure 2–1 is a block diagram of DECchip 21164-AA showing the major functional elements and their positions in the pipeline.

Figure 2–1: Abstract CPU Block Diagram



2.3 Ibox

The primary function of the Ibox is to issue instructions to the Ebox, Mbox and Fbox. In order to provide those instructions, the Ibox also contains the prefetcher, PC pipeline, 48-entry ITB, abort logic, register conflict or dirty logic, and interrupt and exception logic. The Ibox decodes four instructions in parallel and checks that the required resources are available for each instruction. If resources are available and multiple issue is possible, then up to four instructions may be issued. Section 2.10 give the detailed rules governing multiple instruction issue. The Ibox issues only the instructions for which all required resources are available. The Ibox does NOT issue instructions out of order, even if the resources are available for a later instruction and not for an earlier one.

The Ibox controls the primary instruction cache, the Icache. See Section 2.8.2 for more detail.

The Ibox does not advance to a new group of four instructions until all instructions in the current group have been issued. The Ibox only handles naturally aligned groups of four instructions (INT16). If a branch to the middle of such a group occurs, the Ibox attempts issuing the instructions from the branch target to the end of the INT16, proceeding to the next INT16 of instructions only when all the instructions in the target INT16 have been issued. This implies that achieving maximum issue rate requires that code be be scheduled properly and NOPs (floating or integer) be used to fill empty slots in the schedule.

2.3.1 Instruction Prefetch

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

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

Once an Icache miss occurs, the Icache enters fill mode. When it is both in fill mode and awaiting a fill, the Icache is checked for hit. If the instruction data is found in the Icache, the Icache returns to access mode and the prefetcher stops sending fetches to the Mbox. When a new PC is loaded (e.g., taken branches) the Icache returns to access mode until the first miss. The refill buffer receives and holds instruction data from prefetches initiated before before the Icache returned to access mode.

2.3.2 Branch Execution

When a branch or jump instruction is fetched from the Icache, the Ibox takes 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 two-bit history state provided for each instruction location in the cache. This information is used as the prediction for the next execution of the branch instruction. The history status is not initialized on Icache fill, so it may "remember" a branch which is evicted from the Icache and subsequently reloaded.

DECchip 21164-AA 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 stages 3 and 4 plus up to four in stage 2.

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

DECchip 21164-AA provides a twelve-entry subroutine return stack which is controlled by decoding the opcode (BSR, HW_REI and JMP/JSR/RET/JSR_COROUTINE), and disp<15:14> in JMP/JSR/RET/JSR_COROUTINE. The stack stores an Icache index in each entry. (Note that the stack is implemented as a circular queue which wraps around in the overflow and underflow cases.)

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

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

Note that changes from PAL mode to native mode and vice versa are predicted on all PC predictions that use the subroutine return stack. If the opcode isn't HW_REI, this might not seem to make sense, but if the PC prediction is correct, the mode prediction will be as well.

As noted above, Istream prefetching is disabled when a PC prediction is outstanding.

2.3.3 ITB

The Ibox contains a 48-entry fully associative translation buffer to cache recently used instruction-stream address translations and protection information for pages ranging from 8 Kbytes to 512 Kbytes in size. The ITB uses a not-last-used replacement algorithm. The ITB is filled and maintained by PALcode. Each entry supports all four granularity hint bit combinations permitting translation for up to 512 contiguously mapped 8 Kbyte pages using a single ITB entry.

The operating system, via PALcode, must ensure that virtual addresses can only be mapped through a single ITB entry or super page mapping at one time. Multiple simultaneous mapping can cause UNDEFINED results.

While not executing in PAL mode, the 43-bit virtual program counter (PC) is presented each cycle to the ITB. If the PTE associated with the PC is cached in the ITB, the protection bits for the page which contains the PC are used by the Ibox to do the necessary access checks. If there is an Icache miss and the PC is cached in the ITB, the PFN and protection bits for the page which contains the PC are used by the Ibox to do the address translation and access checks.

The DECchip 21164-AA ITB supports 128 ASNs (MAX_ASN=127) via a seven-bit ASN field in each ITB entry. PALcode which supports writes to the architecturally-defined TBIAP register does so by using the hardware-specific HW_MTPR instruction to write to a specific hardware register. This has the effect of invalidating ITB entries which do not have their ASM bit set.

DECchip 21164-AA provides two optional translation extensions referred to as super pages. They are enabled via ICSR<SPE>. One super page mapping maps virtual address bits <39:13> one-to-one to physical address bits <39:13> when virtual address bits <42:41> = 2. This maps the entire physical address space four times over to the quadrant of the virtual address space with virtual address bits <42:41> = 2. The second super page mapping maps virtual address bits <29:13> one-to-one to physical address bits <29:13> with physical address bits <39:30> set to 0. This mapping occurs for virtual addresses with bits <42:30> = 1FFE(Hex), mapping 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(Hex). Access to either super page mapping is only allowed while executing in kernel mode.

2.3.4 Interrupt Logic

The DECchip 21164-AA chip supports three sources of interrupts: hardware, software and asynchronous system trap (AST). There are seven level-sensitive hardware interrupts sourced by pins, 15 software interrupts sourced by an on-chip IPR (SIRR), and 4 AST interrupts sourced by a second on-chip IPR (ASTRR). Interrupts are masked by the hardware interrupt priority level register (IPL). In addition, AST interrupts are qualified by the current processor mode and the performance counter interrupts, the serial line interrupt, and the internally-detected correctable error interrupt are all maskable by bits in the IPR, ICSR (see Chapter 3). All interrupts are disabled when the processor is executing PALcode.

Table 2–1 shows which interrupts are enabled for a given IPL. An interrupt is enabled if the current IPL is less than the target IPL of the interrupt.

Table 2–1: Interrupt Priority Level Effect

Interrupt Source	Target IPL (decimal)
Software Interrupt Request 1	1
Software Interrupt Request 2	2
Software Interrupt Request 3	3
Software Interrupt Request 4	4
Software Interrupt Request 5	5

Table 2–1 (Cont.): Interrupt Priority Level Effect

Interrupt Source	Target IPL (decimal)
Software Interrupt Request 6	6
Software Interrupt Request 7	7
Software Interrupt Request 8	8
Software Interrupt Request 9	9
Software Interrupt Request 10	10
Software Interrupt Request 11	11
Software Interrupt Request 12	12
Software Interrupt Request 13	13
Software Interrupt Request 14	14
Software Interrupt Request 15	15
AST pending (for current or more privileged mode)	2
Performance counter interrupt	29
Power fail interrupt§	30
System machine check interrupt§; Internally detected correctable error interrupt pending	31
External interrupt 20§ (I/O interrupt at IPL 20; corrected system error interrupt)	20
External interrupt 21§ (I/O interrupt at IPL 21)	21
External interrupt 22§ (I/O interrupt at IPL 22; interprocessor interrupt; timer interrupt)	22
External interrupt 23§ (I/O interrupt at IPL 23)	23
Halt§	Masked only by executing in PAL mode.

§These interrupts are from external sources. In some cases, the system environment provides the logic-or of multiple interrupt sources at the same IPL.

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 can not trap (implied DRAINT).

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

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

2.3.5 Performance Counters

TBD FUNCTIONALITY

We have yet to define our performance monitoring features completely.

2.4 Ebox

The Ebox contains two 64-bit integer execution pipelines, a total of 2 adders, 2 logic boxes, 1 barrel shifter, 1 byte zapper, and 1 integer multiplier. Almost all useful bypass paths are implemented; the result of any completed integer operation is available for use by instructions other than integer multiply issuing into either pipeline. (The integer multiplier is unable to recieve data from certain bypass paths. This is reflected in the latency specification at the end of this chapter.) The integer multiplier retires 8 bits per cycle. Table 2–9 lists all instruction latencies. The Ebox also contains the 40-entry 64-bit integer register file containing 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 which 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. Arbitration implemented by the Ibox reserves the write ports for fills from the Mbox when appropriate.

2.5 Mbox

The Mbox contains the address translation buffer for all loads and stores, the write buffer address file, the miss address file, the Dcache interface, and Mbox IPRs. It executes up to two loads per cycle, though a load can not be issued simultaneously with a store or certain other Mbox instructions (see Section 2.10 for detailed issue rules). The address translation datapath receives a virtual address every cycle from each adder in the Ebox. A translation buffer with two read ports generates the corresponding physical addresses and access control information.

2.5.1 Big Endian Support

DECchip 21164-AA provides limited support for big endian data formats via MCSR<BIG_ ENDIAN>. When this bit is set, physical address bit <2> is inverted for all longword D-stream references. It is intended that this mode be set during initialization PALcode and not changed during operation.

2.5.2 DTB

DECchip 21164-AA contains a 64-entry fully associative dual read-ported translation buffer which caches recently used data-stream page table entries for 8 Kbyte pages. Each entry supports all four granularity hint bit combinations permitting translation for up to 512 contiguously mapped 8 Kbyte pages using a single DTB entry. The translation buffer uses a not-last-used replacement algorithm.

The DECchip 21164-AA DTB supports 128 ASNs (MAX_ASN=127) via a seven-bit ASN field in each DTB entry. PALcode which supports writes to the architecturally-defined TBIAP register does so by using the hardware-specific HW_MTPR instruction to write to a specific hardware register. This has the effect of invalidating DTB entries which do not have their corresponding ASM bit set.

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

DECchip 21164-AA provides two optional translation extensions referred to as super pages. They are enabled via MCSR<SP<1:0>>. One super page mapping maps virtual address bits <39:13> one-to-one to physical address bits <39:13> when virtual address bits <42:41> = 2. This maps the entire physical address space four times over to the quadrant of the virtual address space with virtual address bits <42:41> = 2. The second super page mapping maps virtual address bits <29:13> one-to-one to physical address bits <29:13> with physical address bits <39:30> set to 0. This mapping occurs for virtual addresses with bits <42:30> = 1FFE(Hex), mapping 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(Hex). Access to either super page mapping is only allowed while executing in kernel mode.

The DTB is filled and maintained by PALcode. Figure 3–6 shows the DTB miss flow. In general, the operating system, via PALcode, must ensure that virtual addresses can only be mapped through a single DTB entry or super page mapping at one time. Multiple simultaneous mapping can cause UNDEFINED results. The only exception to this rule is that one virtual page may be mapped twice with identical data in two different DTB entries. This occurs in operating systems utilizing virtually accessible page tables like those used by VMS. If the level 1 page table is accessed virtually, PALcode ends up loading the translation information twice, once in the double-miss handler, and once again in the primary handler. The PTE mapping the level 1 page table must remain constant during accesses to this page to meet this requirement.

2.5.3 Replay Traps

For implementation reasons, there are no stalls after the instruction issue point in the pipeline. For certain cases, an Mbox instruction can not be executed because of insufficient resources or some other reason. These instructions trap and the Ibox restarts their execution from the beginning of the pipeline. This is called a replay trap. Replay traps occur in the following cases:

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

- Load-after-store trap: If a load is issued in the cycle immediately following a store that hits in the Dcache, and both access the same memory location, a replay trap occurs. The address match is exact with respect to low order bits of the address, but it is TBD whether it ignores address bits <42:13>.
- When a load is followed within one cycle by any instruction which 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 happens because the consumer instruction is issued speculatively while Dcache hit is being evaluated. If the load misses in the Dcache, the speculative issue of the consumer instruction was incorrect. The replay trap brings the consumer instruction to the issue point before or simultaneously with the availability of fill data.

2.5.4 Load Instruction Execution and the Miss Address File

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

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

- Merging only occurs if the new load miss addresses a different INT8 from all loads previously entered or merged to that miss address file entry.
- Merging only occurs if the new load miss is the same access size as the loads previously entered in that miss address file entry. I.e., quadword loads only merge with other quadword loads and longword loads only merge with other longword loads.
- In the case of longword loads, address bit<2> must be the same. I.e., longword loads with even addresses merge only with other even longword loads and longword loads with odd addresses merge only with other odd longword loads.
- The miss address file does not merge floating point and integer load misses in the same entry.
- Merging is prevented for the MAF entry a certain number of cycles after the Scache access corresponding to the MAF entry begins. Merging is prevented for that entry only if the Scache access hits. The minimum number of cycles of merging is three, the cycle in which the first load is issued and the two subsequent cycles. This corresponds to the most optimistic case of a load miss being forwarded to the Scache without delay (accounting for the cycle saved by the bypass which sends new load misses directly to the Scache when there is nothing else pending).

Note that merging is allowed for loads to non-cacheable space (physical address bit <39> = 1). At the pins, these reads will tell the system environment which INT32 is addressed and which INT8s within the INT32 are actually accessed. (Merging stops for a load to non-cacheable space as soon as the Cbox accepts the reference.) This permits the system environment to access only those INT8s actually requested by load instructions. For memory mapped INT4 registers, the system

environment must return the result of reading each register within the INT8 since DECchip 21164-AA only indicates which INT8s are accessed, not the exact length and offset of the access within each INT8. Systems implementing memory mapped registers with side effects from reads should place each such register in a separate INT8 in memory.

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

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

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

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

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

Referring to Figure 2–2, note that an instruction typically writes the register file in stage 6. 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. Again refering to Figure 2–2, note that a load or store 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, the Dcache (write) access for a fill occurs a cycle later than the Dcache access for a load hit. This is because the fill data arrives just in time to be bypassed to the consuming instruction. Since only loads and stores use the Dcache in the pipeline, the second bubble reserved for a fill is a no-Mbox-instruction bubble. See Section 2.9 for more details of pipeline.

The second bubble is a subset of the first bubble. When two fills are in consecutive cycles (as they are for Scache hit) then three total bubbles are allocated, two no-instruction bubbles followed by one no-Mbox-instruction bubble. Note that the bubbles are requested before it is known whether the Scache (and similarly, the Bcache) will hit. In other words, bubble allocation is speculative.

For fills from the Cbox to floating point registers, no cycle is allocated. Loads which conflict in the pipeline with the fill are forced to miss. Stores which conflict in the pipeline force the fill to be aborted in order to keep the Dcache available to the store operation. In all cases, the floating point register(s) are filled as dictated by the associated MAF entry. A single store can block up to four consecutive fills. (Note that the Fbox has separate write ports for fill data as is necessary for this fill scheme.)

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

Loads misses are all checked against the write buffer contents for conflicts between new loads and previously issued stores. See Section 2.5.6 for more detail.

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

2.5.5 Store Execution

Stores execute in the Mbox by reading the Dcache tag store in the pipeline stage in which a load would read the Dcache, checking for a hit in the next stage, and writing the Dcache data store if there is a hit in the second following pipeline stage. See Section 2.9 for pipeline details.

Loads are not allowed to issue in the second cycle after a store (1 bubble cycle). Other instructions can be issued in that cycle. Stores can issue at the rate of one per cycle because stores streaming down the pipeline do not conflict in their use of resources (the Dcache tag store and Dcache data store are the principal resources). However, a load uses the Dcache data store in the same early stage that it uses the Dcache tag store. Therefore a load would conflict with a store if it were issued in the second cycle after any store. Section 2.9 gives details on store execution in the pipeline.

A load which is issued one cycle after a store in the pipeline creates a conflict if both access exactly[1] the same memory location; the store hasn't updated the location when the load reads it. This conflict is handled by forcing the load to trap (a replay trap). The Ibox flushes the pipeline and restarts execution from the load instruction. By the time the load arrives at the Dcache the second time, the conflicting store has written the Dcache and the load is executed normally.

It is recommended that software not load data immediately after storing it. The replay trap that is incurred is fairly expensive. The best solution is to schedule the load to issue three cycles after the store. No issue stalls or replay traps will occur in that case. If the load is scheduled to issue two cycles after the store, it will be issue-stalled for one cycle for the reasons given above. This is not optimal but is much better than incurring a replay trap on the load.

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

[1] It is TBD if this address check will include the most significant bits of the address. It will be precise over bits <12:0>.

If a store is stalled at the issue point for any reason, it interferes with fills just as if it had been issued. Again, this applies only to fills of floating point data. If, when a store issues, subsequent instructions at the issue point do not issue, then a "shadow" of the store remains in the pipeline latches at the issue point. The Mbox has special logic which detects that the stalled "shadow" of the store is not a new store and will never issue, so the store "shadow" is prevented from interfering with concurrent fills.

For each store, a search of the MAF is done to detect load-before-store hazards. If a store 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 loads from merging with that MAF entry.
2. Conflict bits are set with the store in the write buffer to prevent the store from being issued until all conflicting loads have been issued to the Cbox.

This ensures proper results from the loads and prevents incorrect data from being cached in the Dcache.

A check is done for each new store against stores in the write buffer that have already been sent to the Cbox but have not been completed. This is described in the next section.

2.5.6 Write Buffer and the WMB Instruction

The write buffer address file is contained in the Mbox. The write buffer data store is contained in the Cbox. It contains six fully associative 32-byte entries. The purpose of the write buffer is to minimize the number of CPU stall cycles by providing a high bandwidth (but finite) resource for receiving store data. This is required since DECchip 21164-AA can generate store data at the peak rate of one INT8 every CPU cycle which is greater than the average rate at which the Scache can accept the data if Scache misses occur.

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

A write buffer entry is invalid if it does not contain one of the commands listed above.

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 stores from merging with any of the entries. Also, the next entry to be allocated is marked with a WMB flag. (Note that the entry marked with the WMB flag does not yet have any valid data in it). When an entry marked with a WMB flag is ready to issue to the Cbox, it is not issued until every previously issued write is completely finished. This ensures correct ordering between stores issued before the WMB instruction and stores issued after it.

Each write buffer entry contains a CAM for holding physical address bits <39:5>, 32 bytes of data, eight INT4 mask bits which indicate which of the eight INT4s in the entry contain valid data, and miscellaneous control bits. Among the control bits are a WMB flag, already described, and a no-merge bit which indicates the entry is closed to further merging.

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

Each time the write buffer is presented with a store instruction the physical address generated by the instruction is compared to the address in each valid write buffer entry that is open for merging. If the address is in the same INT32 as an address in a valid write buffer entry which also contains a store and the entry is open for merging, then the new store data is merged into that entry and the entry's INT4 mask bits are updated. If no matching address is found or all entries are closed to merging, then the store data is written into the entry at the top of the free entry queue, that entry is validated, and pointer to the entry is moved from the free entry queue to the pending request queue. Note this scheme does not maintain write ordering.

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

The Mbox also requests a write buffer entry be processed every 64 cycles if there is even one valid entry. This ensures writes do not wait forever to be written to memory. Note that the timer which spurs this is free running.

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

The Mbox continues to request that write buffer entries be processed as long as one contains a STQ_C, STL_C, FETCH, FETCH_M instruction or as long as one is marked by a WMB flag or there is an MB being executed by the Mbox. This insures that these instructions are finished as quickly as possible.

Every store that does not merge in the write buffer is checked against every valid entry. If any is an address match, then the WMB flag is set on the newly allocated write buffer entry. This prevents the Mbox from sending two writes to exactly the same block to the Cbox. The Cbox does not necessarily complete writes in the order in which they were issued, and reordering two writes to the same block can lead to an incorrect final result.

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

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

2.5.7 MB Instruction

The Mbox processes the MB instruction by first completing all outstanding loads and flushing the write buffer. It delays issuing the MB until all loads in the MAF and all writes in the write buffer have completed. The Mbox then issues the MB to the Cbox and waits until the Cbox signals that the MB has been processed before signaling the Ibox that the MB is complete. BC_CTL<EI_OPT_CMD> determines whether the Cbox processes the MB by issuing it on the pins and waiting for acknowledgement. If BC_CTL<EI_OPT_CMD> is not set, the Cbox retires the MB and immediately signals the Mbox that it has been processed. The Ibox stops issuing Mbox instructions after issuing the MB until the signal telling it to start again.

2.5.8 Ibox Read Requests

The Mbox has a four entry file of Ibox read requests. There is a strict one-for-one mapping between these request file entries and the four entries in the refill buffer in the Ibox. Allocation of these entries is controlled by the Ibox. The Ibox never reuses an entry until the previous read has completed. For Istream reads in non-cacheable space, the Mbox marks all INT8s as accessed in the request to the Cbox.

2.5.9 Mbox Arbitration

The Mbox arbitrates among the pending Ibox requests, load misses, and write buffer requests to decide which is the next request to be sent to the Scache and Cbox. The Cbox overrides Mbox arbitration to handle fills and system bus requests (invalidates and probes) and to force a write buffer request to reissue when required by shared block write processing in the Cbox. Normally, load misses are the highest priority Mbox request, followed by Ibox requests and write buffer requests. Write buffer requests become higher priority than reads when a write buffer flush condition exists.

In some cases a request is refused by the Cbox due to lack of resources or a conflict. The Mbox places these refused requests in a replay queue. When arbitrating for an entry in the replay queue, the Mbox uses a priority higher than any other Mbox source. However, when only one replay queue entry is allocated, the Mbox delays arbitrating for the replay queue entry such that other Mbox requests can slip in between replays of refused commands. Sometimes the Cbox will be able to process the other request despite the conflict associated with the replayed request. Once the Mbox has two or more commands in the replay queue, it stops sending new references (because those too might be refused).

2.6 The Cbox

The Cbox controls the Scache and the interface to the DECchip 21164-AA pin bus. It responds to all Mbox generated requests: load misses, instruction fetches and prefetches, and write buffer requests. It also implements a generic writeback cache protocol for the Scache and Bcache (external cache). Chapter 4 describes the DECchip 21164-AA pin bus and coherence protocol.

Internal data transfers between the Mbox (and Ibox) and the Cbox are made via 16-byte buses. Since the internal cache fill block size is 32 bytes, cache fill operations result in two data transfers from the Cbox to the appropriate cache. Since each write buffer entry is 32 bytes in size, write

transactions may result in two data transfers from the write buffer to the Scache and/or the external caches.

The Scache is fully pipelined and is able to provide fill data at a sustained rate of two INT8s per CPU cycle indefinitely. It is writeback and write allocate. Writes which hit in a private-dirty block are processed in a pipelined fashion at a rate of 1 INT16 per CPU cycle. Thus, extremely high data bandwidths are supported by the Scache.

The Scache and Bcache block sizes are selected to be 32 or 64 bytes by SC_CTL<SC_BLK_SIZE>. The Scache and Bcache block sizes are always the same.

The optional Bcache supports high data bandwidth as well. It can provide fill data at a rate as high as one INT16 every 4 CPU cycles if pipelined, though the Bcache in many systems operates at a significantly slower rate. Bandwidth of Scache writebacks into the Bcache is the same, one INT16 per 4 CPU cycles. Writeback bandwidth into the Bcache is optimized by maintaining a modified bit for each INT16 in each Scache block. Only those INT16s that have actually been modified since the block was allocated in the Scache are written back to the Bcache. Scache victim writebacks can therefore take one to four Bcache cycles or not occur at all, depending on the state of the modified bits.

Programs which organize (block) their data such that it fits in the Scache for phases of execution will benefit most significantly from the high data bandwidths available from the DECchip 21164-AA Cbox. Data blocked to fit in the Bcache will benefit from the high Bcache bandwidth supported, but only to the degree that the particular system's Bcache has high bandwidth and never as much as for data blocked to fit in the Scache.

The Scache is set associative but is kept a subset of the larger externally implemented Bcache which is always direct mapped. Logic associated with the Scache tag comparators detects the case in which an Scache miss will cause a block in the Scache to be evicted from both the Bcache and Scache. If the Scache victim is dirty, it is copied from the Scache to the Bcache before the new read is allowed to access the Bcache and cause the Bcache block to be copied back to main memory. In other cases, Scache victims are buffered and written back to the Bcache after reading the new block from the Bcache.

The Cbox detects Scache references to INT64 blocks that have already missed in the Scache. They effectively stall the Scache until the fill occurs. When they proceed they should Scache hit. A special case occurs when the Scache block size is 64 bytes and the second Scache miss is an access to the other INT32 within an outstanding INT64 reference. Such a miss is merged in the Cbox such that the Scache pipeline does not stall. The INT64 fill will service both of the original INT32 references when it arrives. Only one such merge can occur for a particular Scache miss; once both halves of an INT64 Scache block have been requested, no additional merging is done.

NOTE

> The Cbox never merges two INT32 references in non-cacheable space (physical address bit <39>=1). This is required so that the Cbox can inform the environment precisely which INT8s are accessed for each non-cacheable space read reference.

Up to two Scache misses can be processed by the Cbox. These can be Bcache hits or misses. Once one of any two Scache misses is resolved, a new Scache miss can be accepted. Once two Scache misses are outstanding, the Cbox and Scache stop accepting new transactions until one of the outstanding misses is completed. Merging of the kind described in the previous paragraph affects this by effectively condensing two misses into one. Merging can not occur if two misses

are outstanding already, so with merging of INT32s into INT64s, up to three misses can be outstanding.

The Cbox implements a writeback coherence protocol characterized by write allocate, write invalidate, and snooping for dirty data in all coherent caches in the system for each bus read issued by each processor. The Cbox facilitates this protocol by:

- Interacting with the external bus interface so that it may maintain an accurate Bcache duplicate tag store (or Scache duplicate tag store in the absence of a Bcache). An accurate duplicate tag store always has the correct dirty status for each cache block.
- Maintaining shared and dirty status bits for each Scache block. Writes to private-dirty blocks occur without external activity. Writes to shared blocks are broadcast externally. Writes to blocks not shared and not dirty require interface acknowledgment to transition into the dirty state.
- Fulfilling reads to dirty blocks in the Scache or Bcache by providing the data directly from the appropriate cache. Reads from the system bus are processed at highest priority. If the block is dirty, the data is transmitted (under external control) from the appropriate cache.

Normally, the Mbox's arbiter determines the next request that enters the Scache pipeline. The Cbox causes override of the Mbox arbiter in the following cases:

- Scache fills from the Bcache or system environment.
- Processing of system probes and invalidates.
- Write broadcast data transmission or write to a private block after receiving acknowledgment from the interface.

2.7 Fbox

DECchip 21164-AA has an on-chip pipelined Fbox capable of executing both DEC and IEEE floating point instructions. IEEE floating point datatypes S and T are supported with all rounding modes. DEC floating point datatypes F and G are fully supported. There is limited support for D floating point format. The Fbox contains a 32-entry 64-bit floating point register file and a user accessible control register, FPCR, containing round mode controls and exception flag information. The Fbox contains two execution pipelines, a floating point multiply pipeline and a floating point add pipeline (which executes all Fbox instructions except multiply operations). The floating point divide unit is associated with the floating point add pipeline but is not itself pipelined. The Fbox can accept a multiply instruction and a non-multiply instruction every cycle, with the exception of floating point divide instructions. The latency for all instructions except divide is four cycles. Bypassers are provided to allow issue of instructions which are dependent on prior results while those results are written to the register file. For detailed information on instruction timing, refer to Section 2.10.

The floating point multiply pipeline and floating point add pipeline are both capable of executing the CPYS instruction. This is important for two reasons. It allows floating point NOPs to be executed in either floating point pipe and it allows floating point data to be moved from register to register simultaneously with execution of any floating point operation. (Recall that floating point NOP is CPYS F31,F31,F31.)

The floating point register file 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. The Mbox arbitrates between floating point loads that hit in the Dcache and floating point fills from the Cbox, making certain that only one register need be written per fill port in each cycle. Floating point loads that conflict with Cbox fills for use of these write ports are forced to miss in the Dcache so that the Cbox fill can occur. The purpose of this is to maximize the available bandwidth for floating point loads.

2.8 Cache Organization

DECchip 21164-AA includes three on-chip caches. All memory cells are fully static CMOS 6T structures. Parity protection is implemented in all on-chip caches.

2.8.1 Data Cache

The DECchip 21164-AA data cache, the Dcache, is a dual-ported cache implemented as two 8 Kbyte cache banks. It is a write-through, read-allocate direct mapped physical cache with 32-byte blocks. One bank is associated with each of the two Ebox execution pipelines, E0 and E1. The cache banks contain exactly the same data. The Cbox keeps the Dcache coherent and keeps it a subset of the Scache.

A load that misses in the Dcache will result in a Dcache fill. The two banks are filled at the same time with identical data.

2.8.2 Instruction Cache

The DECchip 21164-AA instruction cache, the Icache, is an 8 Kbyte virtual direct-mapped cache. Icache blocks contain 32-bytes of instruction stream data, associated predecode data, the corresponding tag, a seven-bit ASN field (MAX_ASN=127), a one-bit ASM field and a 1 bit PALcode indication per block. Coherency with memory is not maintained by Ibox hardware. The virtual instruction Icache is kept coherent with memory via the IMB PAL call, as specified in the Alpha SRM.

The DECchip 21164-AA virtual instruction cache is kept coherent with changes to PTEs via the IMB PAL call or by assigning a new ASN to the affected process. The TBIA, TBIAP, and TBIS PAL calls do not affect the contents of the Icache in any way.

2.8.3 Second Level Cache

The DECchip 21164-AA second level cache, Scache, is a 96 Kbyte, 3-way set associative, physical, writeback, write-allocate cache with 32 or 64 byte blocks (configured by SC_CTL<SC_BLK_SIZE>). It is a mixed data and instruction cache. The Scache is fully pipelined; it processes reads and writes at the rate of 1 INT16 per CPU cycle and can alternate between read and write accesses without "bubble" cycles.

If the Scache block size is configured to 32 bytes, the Scache is organized as three sets of 512 blocks where each block consists of two 32-byte subblocks. Otherwise the Scache is three sets of 512 64-byte blocks.

Scache tags contain the following special bits for each 32-byte sub-block: one dirty bit, one shared bit, two INT16 modified bits, and one valid bit. Dirty and shared are the coherence state of the subblock required for the cache coherence protocol. The modified bits are used to prevent unnecessary writebacks from the Scache to the Bcache. The valid bit indicates the subblock is valid. In 64-byte block mode, the block is made up of two 32-byte subblocks and the valid, shared, and dirty bits in one subblock always match the corresponding bit in the other subblock

The Scache tag compare logic contains extra logic to check for blocks in the Scache which map to the same Bcache block as a new reference. This allows the Scache block to be moved to the Bcache (if dirty) before the block is evicted because of the new reference missing in the Bcache.

The Scache supports write broadcast by merging write data with Scache data in preparation for a write broadcast as required by the coherence protocol.

2.8.4 External Cache - Bcache

The Cbox implements control for an optional external, direct mapped, physical, writeback, write allocate cache with 32 or 64 byte blocks. (The block size is configured by SC_CTL<SC_BLK_SIZE>). It is a mixed data and instruction cache. Bcache sizes of 1, 2, 4, 8, 16, 32, and 64 Mbytes are supported. See Chapter 4.

2.9 Pipeline Organization

DECchip 21164-AA has an seven stage pipeline for integer operate and memory reference instructions. Floating point operate instructions progress through a nine stage pipeline. The Ibox maintains state for all pipeline stages to track outstanding register writes. The pipeline diagrams below show the DECchip 21164-AA pipeline for several significant examples. The first four cycles are executed in the Ibox and the later stages are executed in the Ebox, Fbox, Mbox, and Cbox. There are bypass paths that allow the result of one instruction to be used as a source operand of a following instruction before it is written to the register file.

Figure 2–2: Pipeline Examples

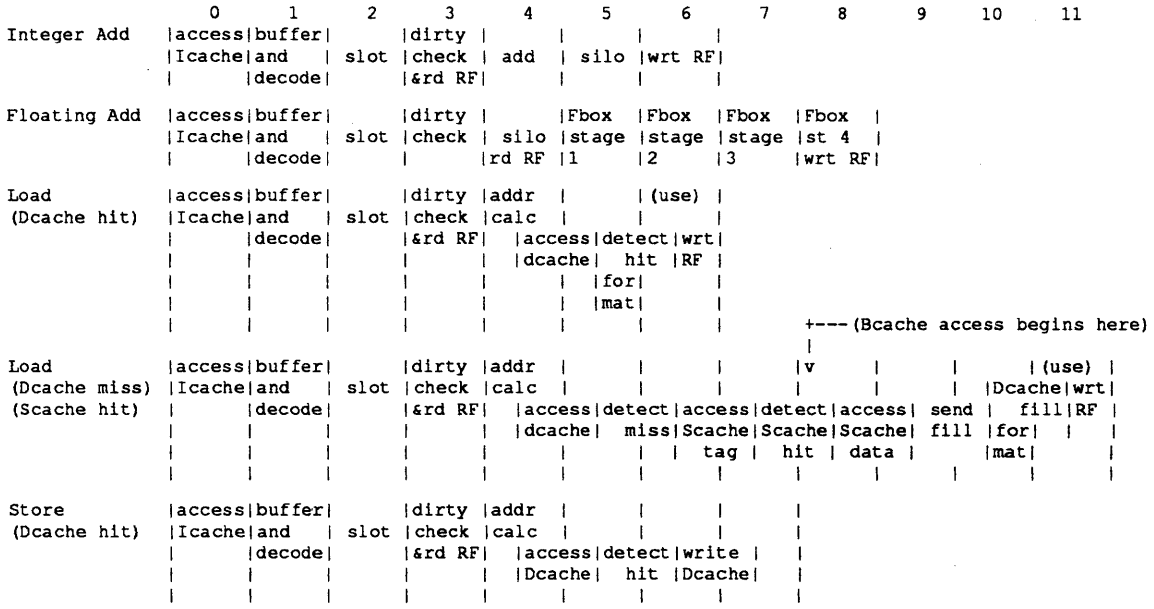


Table 2–2: Pipeline Examples - All Cases

Pipe Stage	Events
0	Access Icache tag and data.
1	Buffer 4 instructions, check for branches, calculate branch displacements, 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 can not 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 integer register file. Stall preceding stages if any instruction can not 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

Pipe Stage	Events
4	Do the add.
5	Result available for use by an operate this cycle.
6	Write the integer register file. Result available for use by an operate this cycle.

Table 2–4: Pipeline Examples - Floating Add

Pipe Stage	Events
4	Read the floating register file.
5	First cycle of Fbox add pipeline.
6	Second cycle of Fbox add pipeline.
7	Third stage of Fbox add pipeline.
8	Fourth stage of Fbox add pipeline. Write the floating point register file.
9	Result available for use by an operate this cycle.

Table 2–5: Pipeline Examples - Load (Dcache hit)

Pipe Stage	Events
4	Calculate the effective address. Begin the Dcache data and tag store access.
5	Finish the Dcache data and tag store access. Detect Dcache hit. Format the data as required. Scache arbitration defaults to E0 in anticipation of a possible miss.
6	Write the integer or floating register file - data available for use by an operate this cycle.

Table 2–6: Pipeline Examples - Load (Dcache miss)

Pipe Stage	Events
4	Calculate the effective address. Begin the Dcache data and tag store access.
5	Finish the Dcache data and tag store access. Detect Dcache miss. Scache arbitration defaults to E0 in anticipation of a possible miss. A load in E1 would be delayed at least one more cycle since default arbitration speculatively selects E0.
6	Begin Scache tag read.
7	Finish Scache tag read. Begin detecting Scache hit.
8	Finish detecting Scache hit. Begin accessing the correct Scache data bank. (Bcache index at pins; Bcache access begins)
9	Finish Scache data bank access. Begin sending fill data from Scache.
10	Finish sending fill data from Scache. Begin Dcache fill. Format the data as required.
11	Finish Dcache fill. Write the integer or floating register file - data available for use by an operate this cycle.

Table 2–7: Pipeline Examples - Store (Dcache hit)

Pipe 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).

The DECchip 21164-AA 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 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 or instructions is (are) presented to be issued but can not proceed.

Upon satisfying all issue requirements, instructions are issued into their slotted pipeline. After issuing, instructions cannot stall in a subsequent pipe stage. It is up to the issue stage to ensure that all resource conflicts are resolved before an instruction is allowed to continue. The only means of stopping instructions after the issue stage is an abort condition. Note that the term abort as used here is different from its use in the Alpha SRM.

Aborts may result from a number of causes. In general, they may be grouped into two classes, namely exceptions (including interrupts) and non exceptions. The basic 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 which were fetched subsequent to the instruction which caused the abort condition. This includes aborting some instructions of a multiply-issued set in the case of an abort condition on the one instruction in the set. The non-exception case, however, does not need to drain the pipeline of all outstanding instructions ahead of the aborting instruction. The pipeline can be immediately restarted at a redirected address. Examples of non exception abort conditions are branch mispredictions, subroutine call/return mispredictions, and 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 above. Due to the nature of some exception conditions, this may occur as late as the integer register file write cycle. (In the case of an arithmetic exception, the processor may execute instructions issued after the exceptional instruction.) Next, the address of the exceptional instruction is latched in the EXC_ADDR IPR. (In the case of an arithmetic exception, the address latched in the EXC_ADDR IPR is that of the lats instruction executed which may be a later instruction than the exceptional instruction.) When the pipeline is fully drained, the processor begins instruction execution at the address given by the PALcode dispatch. The pipeline is drained when all outstanding writes to both the integer and floating point register file have completed and all outstanding instructions have passed the point in the pipeline such that all instructions 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. Generally these are Mbox resources whose availability could not be anticipated accurately at issue time. If the necessary resource is not available when the instruction requires it, the instruction is aborted and the Ibox begins fetching at exactly that instruction, thereby replaying the instruction in the pipeline. A slight variation on this is the load-miss-and-use replay trap in which an operate is issued just as Dcache hit is being evaluated to determine if one of the instructions operands is valid. If it turns out that there is a Dcache miss, then the operate is aborted and replayed.

It should be noted that there are two basic reasons for non-issue 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 type of non-issue cycles can be minimized through code scheduling. The second type of non-issue conditions consist of pipeline bubbles where there is no valid instruction in the pipeline to issue. Pipeline bubbles result from the abort conditions described above. 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.

2.10 Scheduling and Issuing Rules

2.10.1 Instruction Class Definition and Instruction Slotting

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

Table 2–8: Instruction Classes and Slotting

Class Name	Pipeline	Instruction List
LD	E0[1] or E1[2]	all loads except LDx_L
ST	E0	all stores except STx_C.
MBX	E0	LDx_L, MB, WMB, STx_C, HW_LD-lock, HW_ST-cond, FETCH
RX	E0	RS, RC
MXPR	E0 or E1 depending on the IPR	HW_MFPR, HW_MTPR
IBR	E1	integer conditional branches
FBR	FA[3]	floating point conditional branches
JSR	E1	jump to subroutine instructions JMP, JSR, RET, or JSR_COROUTINE, BSR, BR, HW_REI, CALLPAL
IADD	E0 or E1	ADDL ADDL/V ADDQ ADDQ/V SUBL SUBL/V SUBQ SUBQ/V S4ADDL S4ADDQ S8ADDL S8ADDQ S4SUBL S4SUBQ S8SUBL S8SUBQ LDA LDAH
ILOG	E0 or E1	AND BIS XOR BIC ORNOT EQV
SHIFT	E0	SLL SRL SRA EXTQL EXTLL EXTWL EXTBL EXTQH EXTLH EXTWH MSKQL MSKLL MSKWL MSKBL MSKQH MSKLH MSKWH INSQL INSLL INSWL INSBL INSQH INSLH INSWH ZAP ZAPNOT
CMOV	E0 or E1	CMOVEQ CMOVNE CMOVLT CMOVLE CMOVGT CMOVGE CMOVLBS CMOVLBC
ICMP	E0 or E1	CMPEQ CMPLT CMPLE CMPULT CMPULE CMPBGE
IMULL	E0	MULL MULL/V
IMULQ	E0	MULQ MULQ/V
IMULH	E0	UMULH
FADD	FA	floating point operates except multiply and CPYS (but including CPYSN and CPYSE).

[1]Ebox pipeline 0.

[2]Ebox pipeline 1.

[3]Fbox "add" pipeline.

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

Class Name	Pipeline	Instruction List
FDIV	FA	floating point divide.
FMUL	FM[4]	floating point multiply
FCPYS	FM or FA	CPYS (but not CPYSN or CPYSE)
MISC	E0	RPCC, TRAPB
UNOP	none	UNOP

[4]Fbox multiply pipeline.

2.10.1.1 Slotting

The slotting function in Ibox determines which instructions will be sent forward to attempt to issue. The slotting function detects and removes all static functional resource conflicts. The set of instructions output by the slotting function will issue if no register or other dynamic resource conflict is detected in stage 3 of the DECchip 21164-AA pipeline.

The basic slotting algorithm is simple. Starting from the first (lowest addressed) valid instruction in the INT16 in stage 2 of the DECchip 21164-AA Ibox pipeline, attempt to assign that instruction to one of the four pipelines (E0, E1, FA, FM). If it is an instruction which can issue in either of E0 or E1, put it in E0 except that if one the following is true, put it in E1.

- E0 isn't free and E1 is free.
- The next integer instruction‡ in this INT16 can only issue in E0.

If the current instruction is one which can issue in either FA or FM, put it in FA unless FA isn't free. Mark the pipeline selected by this process as taken and begin again with the next sequential instruction. Stop when an instruction can not be allocated an execution pipeline because any pipeline it can use is already taken. The slotting logic also enforces the special rules listed below, stopping the slotting process when a rule would be violated by allocating the next instruction an execution pipeline. Note that the slotting logic doesn't send instructions forward out of logical instruction order because DECchip 21164-AA always issues instructions in order.

1. An instruction of class LD can not be simultaneously issued with an instruction of class ST.
2. All instructions are discarded at the slotting stage after a predicted-taken IBR or FBR class instruction, or a JSR class instruction.
3. After a predicted not-taken IBR or FBR, no other IBR, FBR, or JSR class can be slotted together.
4. The following cases are detected by the slotting logic:
   - from lowest address to highest within an INT16, the arrangement
     I-instruction, F-instruction, I-instruction, I-instruction,
     where I-instruction is any instruction that can issue in one or both of E0 or E1 and F-instruction is any instruction that can issue in one or both of FA or FM.
   - from lowest address to highest within an INT16, the arrangement
     F-instruction, I-instruction, I-instruction, I-instruction.

‡ In this context, an integer instruction is one which can issue in one or both of E0 or E1, not FA or FM.

When this type of case is detected, the first two instructions are forwarded to the issue point in one cycle, and 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. This makes a code sequence that was optimally scheduled for EV4 perform at least as well on DECchip 21164-AA.

2.10.2 Instruction Latencies

After slotting, instruction issue is governed by the availability of registers for read or write 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). Ibox logic in stage 3 of the DECchip 21164-AA pipeline detects all these conflicts.

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

Table 2–9: Instruction Latencies

Class	Latency	Additional time before result available to integer multiply unit†
LD	Dcache hits, latency=2; Dcache miss/Scache hit, latency=7 or longer§	1 cycle
ST	Stores produce no result	-
MBX	LDx_L always Dcache misses, latency depends on memory subsystem state; STx_C, latency depends on memory subsystem state; MB, WMB, and FETCH produce no result	-
RX	RS, RC, latency=1	2 cycles
MXPR	HW_MFPR, latency=1, 2 or longer depending on the IPR; HW_MTPR, produces no result	1 or 2 cycles
IBR	produce no result	-

§When idle, Scache arbitration predicts a load miss in E0. If a load actually does miss in E0, it is sent to the Scache right away. If this hits and no other event in the Cbox affects the operation, the requested data is available for bypass in 7 cycles. Otherwise, the request takes longer, possibly much longer depending on the state of the Scache and Cbox. It should be possible to schedule some unrolled code loops for Scache using a data access pattern that takes advantage of the Mbox load merging function, achieving high throughput with large data sets.

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

Table 2–9 (Cont.): Instruction Latencies

Class	Latency	Additional time before result available to integer multiply unit†
FBR	produce no result	-
JSR	all but HW_REI, latency=1; HW_REI produces no result	2 cycles
IADD	latency=1‡	2 cycles
ILOG	latency=1‡	2 cycles
SHIFT	latency=1	2 cycles
CMOV	latency=2	1 cycle
ICMP	latency=1	2 cycles
IMULL	latency=8 plus up to 2 cycles of added latency depending on the source of the data†; latency until next IMULL, IMULQ, or IMULH can issue if there are no data dependencies is 4 cycles plus the number of cycles added to the latency.	1 cycle
IMULQ	latency=12 plus up to 2 cycles of added latency depending on the source of the data†; latency until next IMULL, IMULQ, or IMULH can issue if there are no data dependencies is 8 cycles plus the number of cycles added to the latency.	1 cycle
IMULH	latency=14 plus up to 2 cycles of added latency depending on the source of the data†; latency until next IMULL, IMULQ, or IMULH can issue if there are no data dependencies is 8 cycles plus the number of cycles added to the latency.	1 cycle
FADD	latency=4	-
FDIV	data dependent latency is preliminary, 2.4 bits per cycle average rate; next floating divide can be issued in the same cycle the result of the previous divide's result is available, regardless of data dependencies.	-
FMUL	latency=4	-
FCPYS	latency=4	-
MISC	RPCC, latency=2; TRAPB produces no result	1 cycle
UNOP	UNOP produces no result	-

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

‡A special bypass provides an effective latency of 0 (zero) cycles for an ICMP or ILOG producing the test operand of an IBR or CMOV. This is only true when the IBR or CMOV issues in the same cycle as the ICMP or ILOG which produces the test operand of the IBR or CMOV. In all other cases the effective latency of ICMP and ILOG is 1 cycle

ISSUE

The actual issue times of floating divides after floating divides is still open. The above statement is approximately correct.

2.10.3 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 by the Ibox forced to do so in different cycles. This is necessary to ensure that the correct result is left in the register file after both instructions have executed. For most instructions, the order in which they write the register file is dictated by issue order, however IMUL, FDIV and LD instructions may require more time than other instructions to complete. Subsequent instructions that write the same destination register are issue-stalled to preserve write ordering at the register file.

Cases involving an intervening producer-consumer conflict are of interest. They can occur commonly in a multiple-issue situation when a register is re-used. In these cases, producer-consumer latencies are equal to or greater than the required producer-producer latency as determined by write ordering and therefore dictate the overall latency.

An example of this case is shown in the code:

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

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

If a load writes a register and within two cycles a subsequent instructions 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, causing the second instruction to be replayed by the Ibox. When the second instruction again reaches the issue point, it is issue-stalled until the load fill occurs.

2.10.4 DECchip 21164-AA Issue Rules

The following is a list of conditions that prevent DECchip 21164-AA from issuing an instruction.

1. No instruction can be issued until all of it's source and destination registers are clean, i.e. all outstanding writes to the destination register are guaranteed to complete in issue order and there are no outstanding writes to the source registers or those writes can be bypassed.

   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 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.
2. An instruction of class LD can not be issued in the second cycle after an instruction of class ST is issued.
3. No LD, ST, LDX_L, MXPR (to an Mbox register), or MBX class instruction after an MB instruction has been issued until until the MB has been acknowledged on the external pin bus.

4. No LD, ST, LDX_L, MXPR (to an Mbox register), or MBX class instruction after a STx_C (or HW_ST-cond) instruction has been issued until the Mbox writes the success/failure result of the STx_C (HW_ST-cond) in its destination register.
5. No IMUL instructions can be issued if the integer multiplier is busy.
6. No floating point divide instructions can be issued if the floating point divider is busy.
7. No instruction can be issued to pipe E0 exactly two cycles before an integer multiplication completes.
8. No instruction can be issued to pipe FA exactly TBD cycles before an floating point divide completes.
9. No instruction can be issued to pipe E0 or E1 exactly two cycles before a integer register fill is requested (speculatively) by the Cbox, except IMULL, IMULQ, IMULH instructions and instructions which do not produce a result at all.
10. No LD, ST, LDX_L, or MBX class instruction can be issued to pipe E0 or E1 exactly one cycle before a integer register fill is requested (speculatively) by the Cbox.
11. 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.

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

2.11 Revision History

Table 2–10: Revision History

Who	When	Description of change
John Edmondson	9-Feb-1992	Initial release.
John Edmondson	1-May-1992	Update to version 1.5.
John Edmondson	29-November-1992	Update to version 1.8.

Chapter 3

PALcode and IPRs

3.1 Overview

PALcode is macrocode that runs with privileges enabled, instruction stream mapping disabled, and interrupts disabled. PALcode has privilege to use five "special" opcodes which allow functions such as physical data stream references and Internal Processor Register (IPR) manipulation. In DECchip 21164-AA, these opcodes are: HW_LD, HW_ST, HW_MFPR, HW_MTPR, and HW_REI. PALmode is the CPU state that distinguishes between native macrocode and PALcode.

Hardware calculates PALcode entry points as offsets to the PAL_BASE IPR. Hardware loads the EXC_ADDR IPR with a return PC when a PALcode flow is begun. EXC_ADDR can also be directly read and written using the HW_MFPR and HW_MTPR instructions. The HW_REI instruction returns instruction flow to the PC stored in EXC_ADDR. The Return Prediction Stack is used to speed execution by predicting the PC to be executed after HW_REI.

PC<0> is used as the PALmode flag both to the hardware and to PALcode itself. When the CPU enters a PAL flow, the Ibox sets PC<0>, and this bit remains set as we move through the PAL Istream. The Ibox hardware ignores this and behaves as if the PC were still longword aligned for the purposes of Istream fetch and execute. On HW_REI, the new state of PALmode is copied from EXC_ADDR<0>.

The DECchip 21164-AA Ebox register file has eight extra registers that are called the PALshadow registers. The PALshadows overlay R8, R9, R10, R11, R12, R13, R14 and R25 when the CPU is in PALmode and ICSR<SDE> is asserted. For additional PAL scratch, the Ibox has a register bank of 24 PALtemps, which are accessable via HW_MTPR and HW_MFPR.

The DECchip 21164-AA architecture group will provide PALcode to support both the OpenVMS and OSF operating systems. We will also provide a DECchip 21164-AA PALcode violation checker (PVC).

3.2 PALcode Entry Points

There are two different types of PALcode entry points: CALL_PAL and traps.

3.2.1 CALL_PAL

The CALL_PAL entry points are used whenever the Ibox encounters a CALL_PAL instruction in the Istream. The CALL_PAL itself is issued into pipe E1 and the Ibox stalls for the minimum number of cycles necessary to perform an implicit TRAPB. The PC of the instruction immediately following the CALL_PAL is loaded into EXC_ADDR and is pushed onto the Return Prediction Stack.

The Ibox contains special hardware to minimize the number of cycles in the TRAPB at the start of a CALL_PAL. Software can benefit from this by scheduling CALL_PALs such that they do not fall in the shadow of:

- IMUL
- Any Floating Point operate, especially FDIV

The Microarchitecture chapter describes the latency of these instructions.

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(hex) to 7F(hex) inclusive.
- is greater than BF(hex).
- between 00 and 3F(hex) inclusive, AND PS<CUR_MOD> 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<11:6> = CALL_PAL function field<5:0>
- PC<5:1> = 0
- PC<0> = 1 - PALmode

The minimum number of cycles for a CALL_PAL execution is 5:

- 1 - issue the CALL_PAL instruction.
- 1 - minimum TRAPB for empty pipe. More typically this will be 4 cycles.
- 2 - The minimum length of a PAL flow. More typically, of course, there will be more than 2 cycles of work for the CALL_PAL.
- 1 - return bubble to do Icache fetch.

3.2.2 Traps

PALcode is started up on a subset of the DECchip 21164-AA traps. (No PALcode assist is required for replay and mispredict type traps). EXC_ADDR is loaded with the return PC and the Ibox performs a TRAPB in the shadow of the trap. The Return Prediction Stack is pushed with the PC of the trapping instruction for precise traps, and with some later PC for imprecise traps.

Table 3–1 shows the PALcode trap entry points, and their offset from the PAL_BASE IPR. The table lists the entry points from highest to lowest priority. (Prioritization among the Dstream traps works because DTB miss is not asserted when there is a sign check error. The priority of ITBmiss and Interrupt is reversed if the is an Icache miss.)

Table 3–1: PALcode Trap Entry Points

Entry Name	Offset(hex)	Description
RESET	0000	Reset
MCHK	0400	Uncorrected hardware error
ARITH	0500	Arithmetic exception
INTERRUPT	0100	Interrupt: hardware, software, and AST
ITBMISS	0180	Istream TBmiss
IACCVIO	0080	Istream access violation or sign check error on PC
FEN	0580	Floating Point Operation attempted with: - FP Instructions(LD, ST and Operates) disabled through FPE bit in ICSR - FP IEEE operation with datatype other than S, T or Q
OPCDEC	0480	Illegal Opcode
DTBMISS_SINGLE	0200	Dstream TBmiss
DTBMISS_DOUBLE	0280	Dstream TBmiss during Virtual PTE fetch
UNALIGN	0300	Dstream unaligned reference
DFAULT	0380	Dstream fault or sign check error on VA

3.3 PAL Opcodes

This section describes the DECchip 21164-AA mapping of the 5 PALRES opcodes. In normal mode, the execution of a PALRES opcode causes an OPCDEC exception if PALmode is not asserted. In addition, ICSR<HWE> is provided as a hook to allow the execution of the PAL opcodes by kernel mode software. Any software executing with ICSR<HWE> set must use extreme care to obey all restrictions listed in this chapter.

3.3.1 HW_LD

The HW_LD instruction is used by PALcode to do special forms of Dstream loads. Figure 3–1 and Table 3–2 describe the format and fields of the HW_LD instruction. Data alignment traps are inhibited for HW_LD instructions.

Figure 3–1: HW_LD instruction

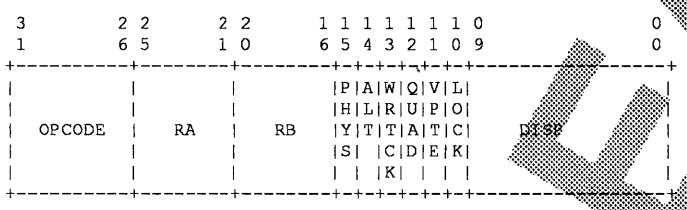


Table 3–2: HW_LD Format description

Field	Description
OPCODE	The OPCODE field contains 1B (hex).
RA	Destination register number.
RB	Base register for memory address.
PHYS	0 - The effective address for the HW_LD is virtual.
	1 - The effective address for the HW_LD is physical. Translation and memory management access checks are inhibited.
ALT	0 - Memory management checks use Mbox IPR DTB_CM for access checks.
	1 - Memory management checks use Mbox IPR ALT_MODE for access checks.
WRTCK	0 - Memory management checks FOR and read access violations.
	1 - Memory management checks FOR, 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.

3.3.2 HW_ST

The HW_ST instruction is used by PALcode to do special forms of Dstream stores. Figure 3–2 and Table 3–3 describe the format and fields of the HW_ST instruction. Data alignment traps are inhibited for HW_ST instructions.

The Ibox logic will always slot HW_ST to pipe E0.

Figure 3–2: HW_ST instruction

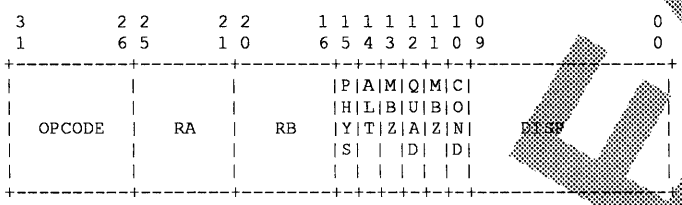


Table 3–3: HW_ST Format description

Field	Description
OPCODE	The OPCODE field contains 1F (hex).
RA	Write data register number.
RB	Base register for memory address.
PHYS	0 - The effective address for the HW_ST is virtual.
	1 - The effective address for the HW_ST is physical. Translation and memory management access checks are inhibited.
ALT	0 - Memory management checks use Mbox IPR DTB_CM for access checks.
	1 - Memory management checks use Mbox IPR ALT_MODE for access checks.
QUAD	0 - Length is longword.
	1 - Length is quadword.
COND	1 - Store_conditional version of HW_ST. In this case, RA will be written with the value of LOCK_FLAG.
DISP	Holds a 10-bit signed byte displacement.
MBZ	Bits 13 and 11 must be zero.

3.3.3 HW_REI

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.

The Ibox uses the Return Prediction Stack to speed the execution of HW_REI. We have two different types of HW_REI:

- Prefetch: In this case, the Ibox will begin fetching the new Istream as soon as possible. This is the version of HW_REI that is normally used.
- Stall Prefetch: This encoding of HW_REI inhibits Istream fetch until the HW_REI itself is issued. Thus, this is the method used to synchronize Ibox changes (such as ITB writes) with the HW_REI. There is a rule that PALcode can only have one such HW_REI in an aligned block of four instructions.

Figure 3–3 and Table 3–4 describe the format and fields of the HW_REI instruction.

The Ibox logic will slot HW_REI to pipe E1.

Figure 3–3: HW_REI instruction

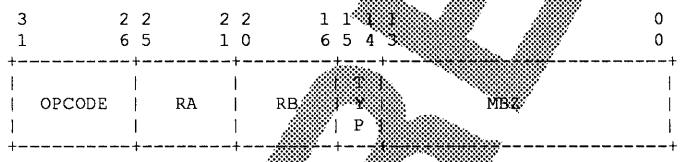


Table 3–4: HW_REI Format description

Field	Description
OPCODE	The OPCODE field contains 1E (hex).
RA/RB	Register numbers, should be R31 to avoid unnecessary stalls.
TYP	10 - normal version
	11 - stall version
MBZ	Bits 13 - 0 Must Be Zero

3.3.4 HW_MFPR and HW_MTPR

The HW_MFPR and HW_MTPR instructions are used to access internal state from the Ibox, Mbox, and Dcache. The data for Ibox IPRs and PALtemps will be moved to and from the Ebox via the PC buses. These HW_MFPRs have a latency of one cycle (HW_MFPR in cycle x results in data available in the Ebox in cycle x+1). For Mbox and Dcache IPRs, the data will be moved to and from the Ebox over the normal load and store datapaths. HW_MFPR from Mbox and Dcache IPRs have a latency of 2 cycles. Ibox hardware slots each type of MXPR to the correct Ebox pipe, see Table 3–6.

Figure 3–4 and Table 3–5 describe the format and fields of the HW_MFPR and HW_MTPR instruction.

Figure 3–4: HW_MFPR, HW_MTPR instruction

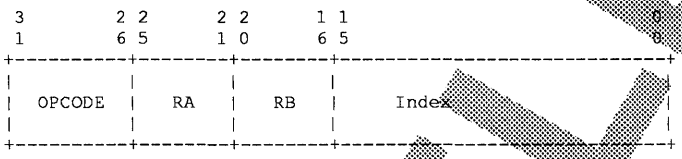


Table 3–5: HW_MTPR and HW_MFPR Format description

Field	Description
OPCODE	The OPCODE field contains 19 (hex) for HW_MFPR, 1D (hex) for HW_MTPR.
RA/RB	Must be the same. Source register for HW_MTPR. Destination register for HW_MFPR.
Index	Specifies the IPR. See Table 3–6 for encodings. See Section 3.9 for more details about a specific IPR.

Table 3–6: IPR Encodings

IPR	Access	Index(hex)	Ibox slots to Pipe
ISR	R	100	E1
ITB_TAG	W	101	E1
ITB_PTE	R/W	102	E1
ITB_ASN	R/W	103	E1
ITB_PTE_TEMP	R	104	E1
ITB_IA	W	105	E1

Table 3–6 (Cont.): IPR Encodings

IPR	Access	Index(hex)	Ibox slots to Pipe
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/WC	10C	E1
EXC_MASK	R	10D	E1
PAL_BASE	R/W	10E	E1
PS	R/W	10F	E1
IPL	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	W	119	E1
IC_PERR_STAT	R/WC	11A	E1
PMCTR	R/W	11C	E1
PALtemp[0:23]	R/W	140-157	E1
DTB_ASN	W	200	E0
DTB_CM	W	201	E0
DTB_TAG	W	202	E0
DTB_PTE	R/W	203	E0
DTB_PTE_TEMP	R	204	E0
MM_STAT	R	205	E0
VA	R	206	E0
VA_FORM	R	207	E0
MVPTBR	W	208	E0
DTBIAP	W	209	E0
DTBIA	W	20A	E0
DTBIS	W	20B	E0
ALT_MODE	W	20C	E0
CC	W	20D	E0

Table 3–6 (Cont.): IPR Encodings

IPR	Access	Index(hex)	Ibox slots to Pipe
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

3.4 PAL storage registers

The DECchip 21164-AA Ebox register file has eight extra registers that are called the PALshadow registers. The PALshadows overlay R8 - R14 and R25 when the CPU is in PALmode and ICSR<SDE> is set. Thus, PALcode can consider R8 - R14 and R25 as local scratch. PALshadow registers cannot be written in the last 2 cycles of a PALcode flow, as the Ibox does not implement complete dirty logic on these registers. The normal state of the CPU is ICSR<SDE> = ON. PALcode disables SDE for the unaligned trap and for error flows.

The Ibox holds a bank of 24 PALtemp registers. The PALtemps are accessed with the HW_MTPR and HW_MFPR instructions. The latency from a PALtemp read to availability is one cycle.

3.5 SRM defined State - OpenVMS

This table is an accounting of the DECchip 21164-AA storage used to implement the SRM defined state for OpenVMS.

Table 3–7: OpenVMS SRM defined State

Register Name	Mnemonic	Access	Internal Storage
Processor Status	PS	R/W	PALtemp /Ibox–PS /Mbox–DTB_CM / Interrupt logic–IPL/ PALshadow
Program Counter	PC	–	Ibox
AST Enable	ASTEN	R/W	Interrupt logic–ASTER
AST Summary	ASTSR	R/W	Interrupt logic–ASTRR
Interproc. Interrupt	IPIR	W	—

Table 3–7 (Cont.): OpenVMS SRM defined State

Register Name	Mnemonic	Access	Internal Storage
Interrupt Priority Level	IPL	R/W	Interrupt Logic–IPL
Machine Check Error Summary	MCES	R/W	PALtemp
Privileged Context Block Base	PCBB	R	PALtemp
Processor Base Register	PRBR	R/W	PALtemp
Page Table Base Register	PTBR	R	PALtemp
System Control Block Base	SCBB	R/W	PALtemp
SW Interrupt Request Register	SIRR	W	—
SW Interrupt Summary Register	SISR	R	Interrupt logic–ISR
TB Check	TBCHK	R	Not implemented
TB Invalidate All	TBIA	W	—
TB Invalidate All Process	TBIAP	W	—
TB Invalidate All Dstream	TBIAD	W	—
TB Invalidate All Istream	TBIAI	W	—
TB Invalidate Single	TBIS	W	—
Kernel Stack Pointer	KSP	None	PCB
Executive Stack Pointer	ESP	R/W	PCB
Supervisor Stack Pointer	SSP	R/W	PCB
User Stack Pointer	USP	R/W	PALtemp
Virtual Page Table Base	VPTB	R/W	PALtemp / Ibox–IVPTBR/ Mbox–MVPTBR
Who Am I	WHAMI	R	PALtemp
Floating Point Enable	FEN	W	Ibox–ICSR
Address Space Number	ASN	W	Ibox–ITB_ASN/ Mbox–DTB_ASN
Cycle Counter	CC	R/W	Mbox–CC,CC_CTL. Read with RPCC
Unique	UNQ	R/W	PCB
lock_flag	—	R/W	Cbox/System. Access with LDx_L and STx_C, and HW_LD and HW_ST variants.

3.6 SRM defined State - OSF

This table is an accounting of the DECchip 21164-AA storage used to implement the SRM defined state for OSF.

Table 3–8: OSF SRM defined State

IPR Name	Mnemonic	Access	Internal Storage
Processor Status	PS	R/W	PALtemp /Ibox–PS /Mbox–DTB_CM / Interrupt logic–IPL/ PALshadow
Program Counter	PC	–	Ibox
Interrupt Entry Address	entINT	W	PALtemp
Arith Trap Entry Address	entARITH	W	PALtemp
MM Fault Entry Address	entMM	W	PALtemp
Unaligned Access Entry Address	entUNA	W	PALtemp
Instruction Fault Address	entIF	W	PALtemp
Call System Entry Address	entSys	W	PALtemp
User Stack Pointer	USP	R/W	PALtemp
Kernel Stack Pointer	KSP	W	PALtemp
Kernel Global Pointer	Kgp	W	PALtemp
System Value	sysval	R/W	PALtemp
Page Table Base Register	ptptr	R/W	PALtemp
Virtual Page Table Base	vptbr	W	Ibox–IVPTBR/ Mbox–MVPTBR
Process Control Block Base	PCBB	R/W	PALtemp
Address Space Number	ASN	W	Ibox–ITB_ASN/ Mbox–DTB_ASN
Cycle Counter	CC	R/W	Mbox–CC,CC_CTL. Read with RPCC.
Floating Point Enable	FEN	W	Ibox–ICSR
lock_flag	–	R/W	Cbox/System. Access with LDx_L and STx_C, and HW_LD and HW_ ST variants.
Unique	UNQ	R/W	PCB
Who Am I	WHAMI	R	PALtemp

3.7 Performance

This list is a summary of DECchip 21164-AA features that improve PAL performance:

- PALshadows save cycles that would have been spent stashing and restoring GPRs.
- Ibox performs minimum TRAPB on CALL_PAL entry.
- Return Prediction Stack is used to speed HW_REI.
- Ibox and Mbox hardware calculate the Virtual Address of the PTE entry needed on a TBmiss.
- Ibox and Mbox support distinct trap entry points for single and double TBmiss.
- The design of the interrupt hardware is specifically tailored to speed up OpenVMS CALL_ PALs like MTPR_IPL.

- The PALtemps have a 1 cycle latency.
- The more frequent PAL trap entry points are grouped together to improve Icache hit on traps.

3.8 TBmiss flows

Figure 3–5: Istream TBmiss flow

```
Assumptions, info, etc.
   This is the entry for Istream TBmiss.  A virtual fetch of the PTE will
       be attempted.  If the virtual PTE fetch TBmisses, a trap will
       be taken to the double_miss routine, which will fill the TB for
       the PTE fetch and HW_REI back to this routine.
   Instruction pairs show E0/E1.
   Best case timing: 16 cycles (6 in, 8 execute, 2 out)

ITBMISS:
   nop
   mfpr    r8, ev5$_ifault_va_form ; Get virtual address of PTE.

   nop
   mfpr    r10, exc_addr           ; Get PC of faulting instruction.

   ld_vpte r8, 0(r8)               ; Get PTE, traps to DTBMISS_DOUBLE in case of TBmiss
   mtpr    r10, exc_addr           ; Restore exc_address if there was a trap.

   mfpr    r31, ev5$_va            ; Unlock VA in case there was a double miss
   nop

   and     r8, #pte$m_foe, r25     ; Look for FOE set.
   blbc    r8, INVALID_OR_FOE_ITBE_HANDLER ; PTE not valid.

   nop
   bne     r25, INVALID_OR_FOE_ITBE_HANDLER

   nop
   mtpr    r8, ev5$_itb_pte        ; Ibox remembers the VA, load the PTE into the ITB.

   hw_rei_stall                    ; Done, synch and return.
```

Figure 3–6: Dstream TBmiss flow

```
Assumptions, info, etc.
   This is the entry for Dstream TBmiss (from native or PALmode).
       A virtual fetch of the PTE will be attempted.  If the virtual
       PTE fetch TBmisses,  a trap will be taken to the double_miss routine,
       which will fill the TB for the PTE fetch and HW_REI back to this routine.
   Instruction pairs show E0/E1.
   Best case timing: 18 cycles (8 trap shadow, 9 execute, 1 out)

DTBMISS_SINGLE:
   mfpr    r8, ev5$_va_form        ; Get virtual address of PTE.
   mfpr    r10, exc_addr           ; Get PC of faulting instruction.

   mfpr    r9, ev5$_mm_stat        ; Get read/write bit.
   mtpr    r10, pt6                ; Stash exc_addr away.

   ld_vpte r8, 0(r8)               ; Get PTE, traps to DTBMISS_DOUBLE in case of TBmiss
   nop                             ; Pad MFPR VA

   mfpr    r10, ev5$_va            ; Get original faulting VA for TB load.
   nop

   mtpr    r8, ev5$_dtb_pte        ; Write DTB PTE part.
   blbc    r8, INVALID_DPTE_HANDLER    ; Handle invalid PTE

   mtpr    r10, ev5$_dtb_tag       ; Write DTB TAG part, completes DTB load.  No virt ref for 3 cycles.
   mfpr    r10, pt6

                                   ; Following 2 instructions take 2 cycles
   mtpr    r10, exc_addr           ; Return linkage in case we trapped.
   mfpr    r31,  pt0               ; Pad the write to dtb_tag,

   hw_rei                          ; Done, return
```

3.9 IPRs

This section describes, on a box by box basis, all the DECchip 21164-AA Internal Processor Registers. Ibox, Mbox, and Dcache IPRs are accessable to PALcode via the HW_MTPR and HW_MFPR instructions. Table 3–6 lists the IPR numbers. Cbox, Scache, and Bcache IPRs are accessable in the physical address region FFFFF00000 to FFFFFFFFFF. Table 3–30 summarizes the Cbox, Scache, and Bcache IPRs. Table 3–43 lists restrictions on the IPRs.

3.9.1 Ibox IPRs

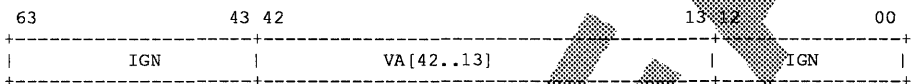
NOTE

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

3.9.1.1 ITB_TAG

The ITB_TAG register is a write only register. This register is written by hardware on an ITBMISS/IACCVIO, with the tag field of the faulting VA. To ensure the integrity of the ITB, the TAG and PTE fields of an ITB entry are updated simultaneously by a write to the ITB_PTE register. This write causes the contents of the ITB_TAG register to be written into the tag field of the ITB location, which is determined by a NLU algorithm. The PTE field is obtained from the MTPR ITB_PTE instruction.

Figure 3–7: Istream TB Tag, ITB_TAG

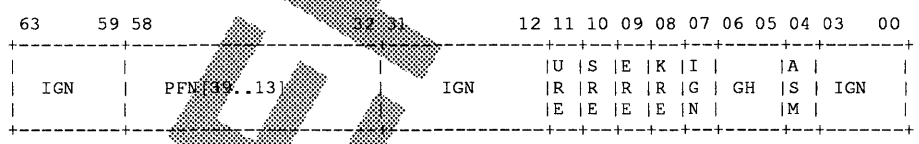


3.9.1.2 ITB_PTE

The ITB_PTE register is a read/write register. A write to this register, writes both the PTE and TAG fields of an ITB location determined by a not-last-used algorithm. The TAG and PTE fields are updated simultaneously to insure the integrity of the ITB. A write to the ITB_PTE register increments the NLU pointer, which allows for writing the entire set of ITB PTE and TAG entries. The TAG field of the ITB location is determined by the contents of the ITB_TAG register. The PTE field is available in the MTPR ITB_PTE instruction. Writes to this register use the memory format bits as described in the Open VMS memory management chapter of the Alpha SRM.

Note: The NLU pointer is bumped in trap shadows.

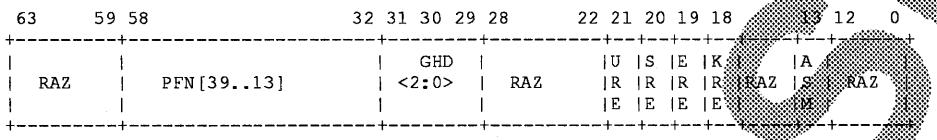
Figure 3–8: Istream TB PTE Write Format, ITB_PTE



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 updates the NLU pointer according to the not-last-used algorithm. A zero value is returned to the integer register file. A second read of the ITB_PTE_TEMP register returns the PTE the the general purpose integer register file.

Note: The NLU pointer is bumped in trap shadows.

Figure 3–9: Istream TB PTE Read Format, ITB_PTE



3.9.1.3 Address Space Number, ITB_ASN

The ITB_ASN register is a read/write register which contains the address space number (ASN) of the current process.

Figure 3–10: Address Space Number Read/Write Format, ITB_ASN

3.9.1.4 ITB_PTE_TEMP

The ITB_PTE_TEMP register 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 integer general purpose register file.

Figure 3–11: Istream TB PTE Temp Read Format, ITB_PTE_TEMP

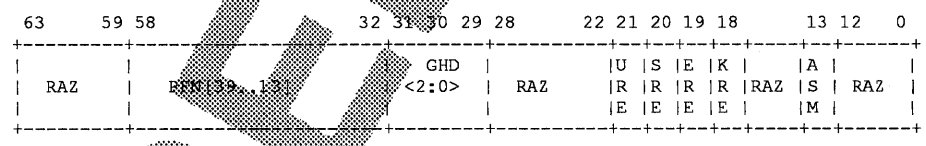


Table 3–9: Description of GHD bits in ITB_PTE_TEMP read format

Name	Extent	Type	Description
GHD	31	RO	Is set if GH(granularity hint) equals 11.
GHD	30	RO	Is set if GH(granularity hint) equals 10 or 11.
GHD	29	RO	Is set if GH(granularity hint) equals 01, 10 or 11.

3.9.1.5 Istream TB Invalidate All Process, ITB_IAP

This is a write-only register. Any write to this register invalidates all ITB entries, whose ASM bit equals zero.

3.9.1.6 IStream TB Invalidate All, ITB_IA

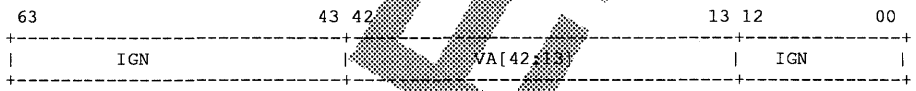
This is a write-only register. Any write to this register invalidates all ITB entries, and resets the ITB NLU pointer to its initial state. RESET Palcode must execute an MTPR ITB_IA instruction in order to initialize the NLU pointer.

3.9.1.7 ITB_IS

This is a write-only register. Writing a virtual address to this IPR invalidates the ITB entry that meets any one of the following criteria:

- An ITB entry whose VA field matches ITB_IS<42:13> and whose ASN field matches ITB_ASN<10:4>.
- An ITB entry whose VA field matches ITB_IS<42:13> and whose ASM bit is set.

Figure 3–12: ITB_IS



3.9.1.8 Formatted Faulting VA register, IFAULT_VA_FORM

This is a read-only register which contains the formatted faulting virtual address on an ITBMiss/IAC IACCVIO's generated by sign-check errors). The formatted faulting address generated depends on whether NT super page mapping is enabled through the SPE <0> bit of the ICSR.

Figure 3–13: IFAULT_VA_FORM in non NT mode

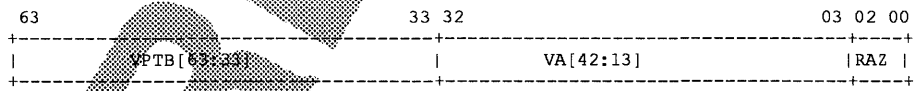
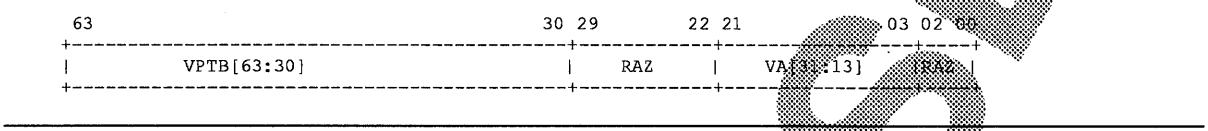


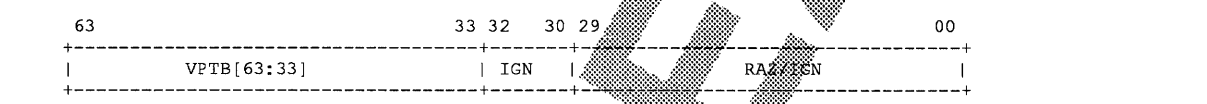
Figure 3–14: IFAULT_VA_FORM in NT mode



3.9.1.9 Virtual Page Table Base register, IVPTBR

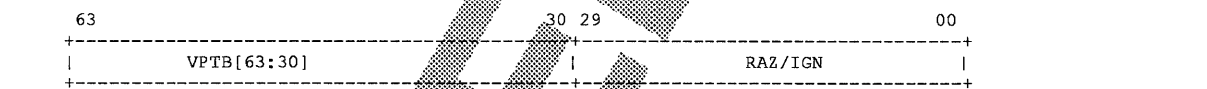
This is a read-write register.

Figure 3–15: IVPTBR in non NT mode



Bits <32:30> are undefined on a read of this register in non NT mode.

Figure 3–16: IVPTBR in NT mode



3.9.1.10 Icache Parity Error Status register, ICPERR_STAT

This is read/write register that contains information about an Icache Parity error. The error status bits may be cleared by writing a 1 to the appropriate bits.

Figure 3–17: ICPERR_STAT Read format

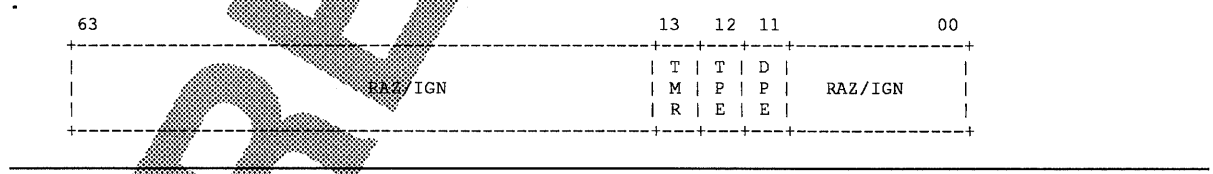


Table 3–10: ICPERR_STAT Field Descriptions

Name	Extent	Type	Description
DPE	11	W1C	Data parity error occurred.
TPE	12	W1C	Tag parity error occurred.
TMR	13	W1C	Timeout reset error or CFAIL_H/no CACK_H error occurred.

3.9.1.11 ICache Flush Control register, IC_FLUSH_CTL

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

3.9.1.12 Exception Address register, EXC_ADDR

The EXC_ADDR register is a read-write register used to restart the machine 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 writes happen as a result of exceptions/interrupts and CALLPAL instructions. Hardware writes which occur as a result of exceptions/interrupts take precedence over all other writes.

In case of an exception/interrupt, hardware writes a PC to this register in S6 of the execution pipeline. In case of precise exceptions, this is the PC of the instruction that caused the exception. In case of imprecise exceptions/interrupts, this is the PC of the next instruction that would have issued if the exception/interrupt was not reported.

In case of a CALLPAL instruction, the PC of the instruction after the CALLPAL is written to EXC_ADDR in S5. Software writes of the register through the HW_MTPR instruction also take place in S5. At a given time only a CALLPAL or HW_MTPR instruction will attempt to write EXC_ADDR as both these instructions are slotted to the E1 pipe.

BIT <0> of this register is used to indicate PAL mode. On a HW_REI the mode of the machine is determined by BIT <0> of the EXC_ADDR register.

Figure 3–18: EXC_ADDR Read/Write format

3.9.1.13 Exception Summary register, EXC_SUM

The exception summary register records the different arithmetic traps that have occurred since the last time EXC_SUM was written. Any write to this register clears bits <16:10>.

Figure 3–19: Exception Summary register Read Format, EXC_SUM

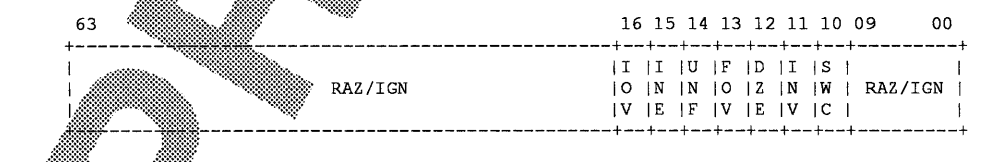


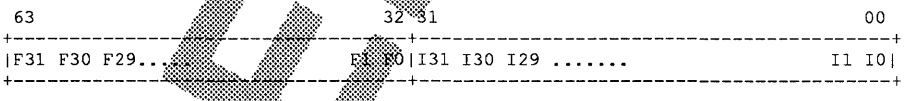
Table 3–11: EXC_SUM Field Descriptions

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 all previous floating point instructions that trapped since the last MTPR EXC_SUM also contained the /S modifier. The SWC bit is cleared whenever a floating point instruction without the /S modifier completed with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps until the register is written via an MTPR instruction. The bit is always cleared upon any MTPR write 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 Fbox convert to integer overflow or Integer Arithmetic Overflow.

3.9.1.14 Exception Mask Register, EXC_MASK

The exception mask register records the destinations of instructions that have caused an arithmetic trap, since the last time EXC_MASK was cleared. The destination is recorded as a single bit mask in the 64 bit IPR representing F0-F31 and I0-I31. A write to EXC_SUM clears the EXC_MASK register.

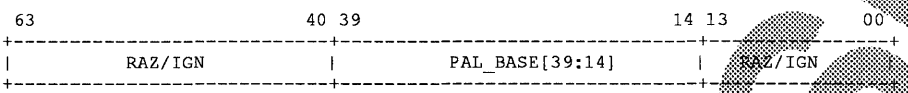
Figure 3–20: Exception Mask register Read Format, EXC_MASK



3.9.1.15 PAL Base Register, PAL_BASE

The PAL_BASE register is a read/write register which contains the base address for PALcode. The register is cleared by hardware on reset.

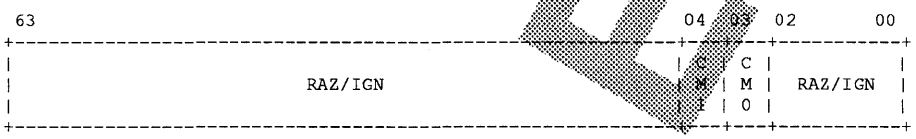
Figure 3–21: PAL_BASE



3.9.1.16 Processor Status, PS

The processor_status register is a read/write register containing the current mode bits of the architecturally defined PS.

Figure 3–22: Processor Status, PS



3.9.1.17 Ibox Control/Status Register, ICSR

This is a read-write register which contains Ibox related control and status information.

Figure 3–23: Ibox Control/Status Register ICSR

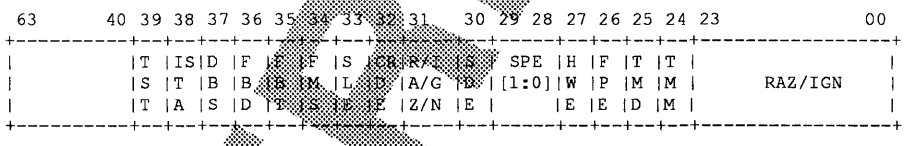


Table 3–12: ICSR Field Descriptions

Name	Extent	Type	Description
TMM	24	RW,0	If set, the timeout counter counts 5K cycles before asserting timeout reset. If clear, the timeout counter counts 1 billion cycles before asserting timeout reset.
TMD	25	RW,0	If set, disables the ibox timeout counter. Does not affect CFAIL/no CACK error.
FPE	26	RW,0	If set floating point instructions may be issued. When clear floating point instructions cause FEN exceptions.
HWE	27	RW,0	If set, allows PALRES instructions to be issued in kernel mode.

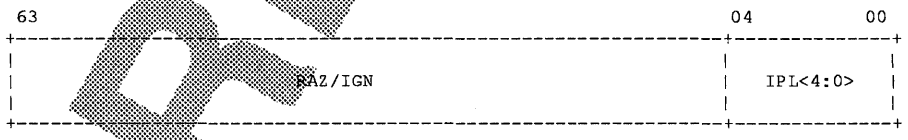
Table 3–12 (Cont.): ICSR Field Descriptions

Name	Extent	Type	Description
SPE	29:28	RW,0	If SPE<1> is set, it enables super page mapping of istream virtual addresses VA<39:13> directly to physical address PA<39:13>, if VA<42:41> = 10. Virtual address bit VA<40> is ignored in this translation. Access is allowed only in kernel mode. SPE<0> when set, enables super page mapping of istream virtual addresses VA<42:30>=1FFE (Hex) directly to physical address PA<39:30>= 0(Hex). VA<30:13> is mapped directly to PA<30:13>. Access is allowed only in kernel mode.
SDE	30	RW,0	If set, enables PAL shadow registers.
CRDE	32	RW,0	If set, enables correctable error interrupts.
SLE	33	RW,0	If set, enables serial line interrupts.
FMS	34	RW,0	If set, forces miss on Icache references.
FBT	35	RW,0	If set, forces bad Icache tag parity.
FBD	36	RW,0	If set, forces bad Icache data parity.
DBS	37	RW,1	This bit controls the selection of the multiplexer for the debug port. If set the debug port sees bits <11:4> of the siloed PC. If cleared, the packet from the MBOX is selected.
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 causes the TEST_STATUS_H pin of the chip to be asserted.

3.9.1.18 Interrupt Priority Level Register, IPL

This is a read/write register containing the value of the architecturally specified IPL register. Whenever hardware detects an interrupt whose target IPL level is greater than the value in IPL<4:0>, an interrupt is taken.

Figure 3–24: Interrupt Priority Level Register, IPL



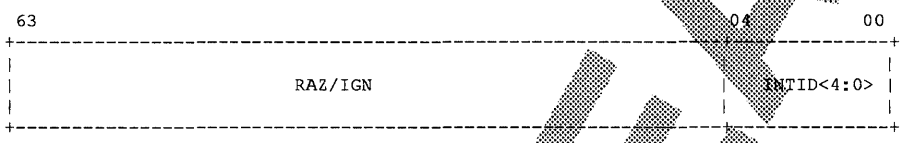
3.9.1.19 Interrupt Id Register, INTID

This is a read only register. It is written by hardware with the target IPL of the highest priority pending interrupt. The hardware recognizes an interrupt if this IPL is greater than the IPL given by IPL<4:0>. Interrupt service routines may use the value of this register to determine the cause of the interrupt. PAL code, for the interrupt service, must ensure that the IPL level in INTID is greater than the IPL level specified by the IPL register. 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, as 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. PAL code for interrupt service must check the ISR to determine if a HALT interrupt has occured.

Figure 3–25: Interrupt Id Register, INTID



3.9.1.20 Aynchronous System Trap Request Register, ASTRR

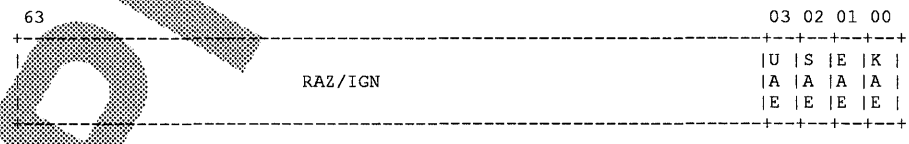
The Asynchronous System Trap Request Register is a read/write register which contains bits to request AST interrupts in each of the four processor modes(USEK). In order to generate an AST interrupt, the corresponding enable bit in the ASTER must be set and the current processor mode given in PS<4:3> should be equal or higher than the mode associated with the AST request.

Figure 3–26: Asynchronous System Trap Request Register, ASTRR

3.9.1.21 Aynchronous System Trap Enable Register, ASTER

The Asynchronous System Trap Enable Register is a read/write register which contains bits to enable corresponding AST interrupt requests.

Figure 3–27: Asynchronous System Trap Enable Register, ASTER format



3.9.1.22 Software Interrupt Request Register. SIRR

The Software Interrupt Request Register 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<15:1>.

Figure 3–28: Software Interrupt Request Register, SIRR write format

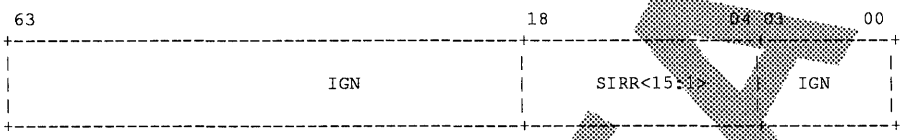


Table 3–13: SIRR Field Descriptions

Name	Extent	Type	Description
SIRR	18:4	RW	Request software interrupts.

3.9.1.23 HW Interrupt Clear register, HWINT_CLR

This is a write-only register, used to clear edge-sensitive hardware interrupt requests.

Figure 3–29: Hardware Interrupt Clear Register, HWINT_CLR

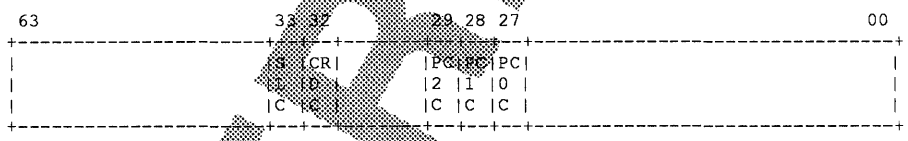


Table 3–14: HWINT_CLR Field Descriptions

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

3.9.1.24 Interrupt Summary register, ISR

The Interrupt Summary register is a read only register which contains information about all pending hardware/software/AST interrupt requests.

Figure 3–30: Interrupt Summary Register, ISR read format

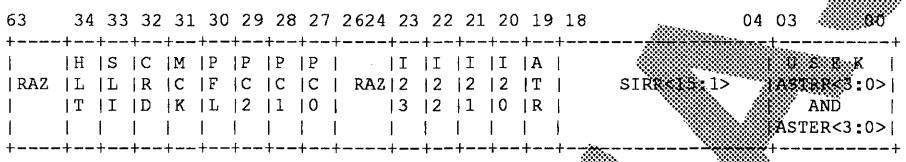


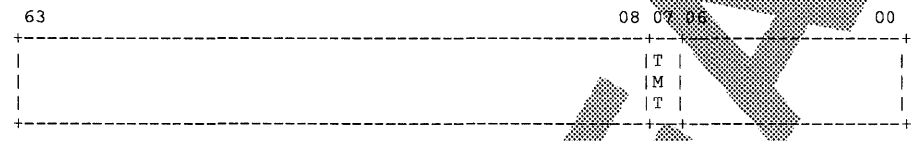
Table 3–15: ISR read format Field Descriptions

Name	Extent	Type	Description
ASTRR[3:0]	3:0	RO	AST requests 3 through 0 (USEK) at IPL 2.
SIRR[15:1]	18:4	RO,0	Software interrupt requests 15 through 1 corresponding to IPL 15 through 1.
ATR	19	RO	Is 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 at IPL 20.
I21	21	RO	External hardware interrupt at IPL 21.
I22	22	RO	External hardware interrupt at IPL 22.
I23	23	RO	External hardware interrupt at IPL 23.
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 - Powerfail (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 .

3.9.1.25 Serial line transmit, SL_XMIT

The serial line transmit register is a write-only register used to transmit bit-serial data off chip under the control of a software timing loop. The value of the TMT bit is transmitted off chip on the SROM_CLK_H pin. In normal operation mode (not in debug mode), the SROM_CLK_H pin is overloaded and serves both the serial line transmission and the Icache serial ROM interface.

Figure 3–31: Serial line transmit Register, SL_XMIT



3.9.1.26 Serial line receive, SL_RCV

The serial line receive register 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_DAT_H pin. A serial line interrupt is requested whenever a transition is detected on the SROM_DAT_H pin and the SLE bit in the ICSR is set. During normal operations (not in test-mode), the SROM_DAT_H pin is overloaded and serves both the serial line reception and the ICache serial ROM interface.

Figure 3–32: Serial line receive Register, SL_RCV

3.9.1.27 Performance Counter register, PMCTR

The Performance Counter register is a read/write register which controls the three on-chip performance counters. Performance counter interrupt requests are summarized in ISR (Interrupt Summary Register).

Figure 3–33: Performance Counter Register, PMCTR

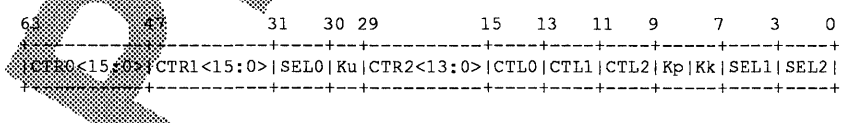


Table 3–16: PMCTR Field Descriptions

Name	Extent	Type	Description
CTR0[15:0]	63:48	RW	16 bit counter
CTR1[15:0]	47:32	RW	16 bit counter
CTR2[13:0]	29:16	RW	14 bit counter
CTL0[1:0]	15:14	RW,0	CTR0 counter control: 00 counter disable,interrupt disable 01 counter enable, interrupt disable 10 counter enable, interrupt at freq 65536 (16384) 11 counter enable, interrupt at freq 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 freq 65536 (16384) 11 counter enable, interrupt at freq 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 freq 65536 (16384) 11 counter enable, interrupt at freq 256
SEL0	31	RW	Counter0 Select - see Table 3–17
SEL1[3:0]	7:4	RW	Counter1 Select - see Table 3–17
SEL2[3:0]	3:0	RW	Counter2 Select - see Table 3–17
Ku	30	RW	Kill User mode, Disables measurements in user mode
Kp	9	RW	Kill PAL mode, Disables measurements in PAL mode
Kk	8	RW	Kill Kernel,Executive, Supervisor mode. Disables measurements in Kernel, Executive, and Supervisor modes. Ku=1,Kp=1,Kk=1 allows measurements in Exec/Supervisor modes only

Table 3–17: PMCTR Counter Select Options

Counter0 Select0[0]	Counter1 Select1[3:0]	Counter2 Select2[3:0]
0:Cycles	0x0:Non-Issue cycles	0x0: Long(>15 cycle) Stalls
	0x1:Split-Issue cycles	0x1: reserved
	0x2:Pipe-Dry cycles	
	0x3:Replay Trap	
	0x4:Single-Issue cycles	
	0x5:Dual-Issue cycles	
	0x6:Triple-Issue cycles	
	0x7:Quad-Issue cycles	

Table 3–17 (Cont.): PMCTR Counter Select Options

Counter0 Select0[0]	Counter1 Select1[3:0]	Counter2 Select2[3:0]
1:Instructions	0x8:jsr-ret	0x2:PC-Mispredicts
	0x8:Cond-Branch	0x3:BR-Mispredicts
	0x8:All Flow-change instructions if sel2=!(PC-M or BR-M)	
	0x9:IntOps	0x4:Icache/RFB Misses
	0xA:FPOps	0x5:ITB Miss Services
	0xB:Loads	0x6:Dcache LD Misses
	0xC:Stores	0x7:DTB Misses
	0xD:Icache	0x8:LDs merged in MAF
	0xE:Dcache Accesses	0x9:LDU Replays
		0xA:WB/MAF full Replays
		0xB:External Perf_Mon input
		0xC:Cycles
		0xD:MB Instructions
		0xE:LDxL Instructions
	0xF:Pick CBOX input 1	0xF:Pick CBOX input 2

3.9.2 Mbox and Dcache IPRs

NOTE

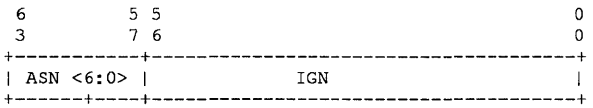
Traps are factored into MBOX IPR write operations unless noted otherwise.

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

3.9.2.1 DTB_ASN, Dstream TB Address Space Number

The DTB_ASN register is a write-only register which, when not in PALmode, must be written with an exact duplicate of the ITB_ASN register's ASN field.

Figure 3–34: DTB_ASN



3.9.2.2 DTB_CM, Dstream TB Current Mode

The DTB_CM register is a write-only register which, when not in PALmode, must be written with an exact duplicate of the Ibox Processor Status (IPS) register's CM field. These bits indicate the Current Mode of the machine.

Figure 3–35: DTB_CM

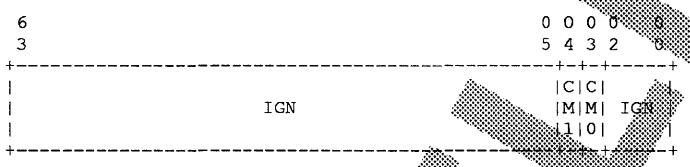


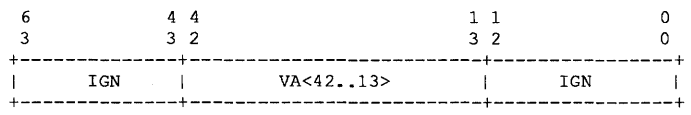
Table 3–18: DTB_CM Mode Bits

CM<1>	CM<0>	Current Mode
0	0	Kernel Mode
0	1	Executive Mode
1	0	Supervisor Mode
1	1	User Mode

3.9.2.3 DTB_TAG, Dstream TB TAG

The DTB_TAG register is a write-only register which writes the DTB tag and the contents of the DTB_PTE register to the DTB. To insure 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 algorithm implemented in hardware. A write to the DTB_TAG register increments the 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 3–36: DTB_TAG, Dstream TB Tag



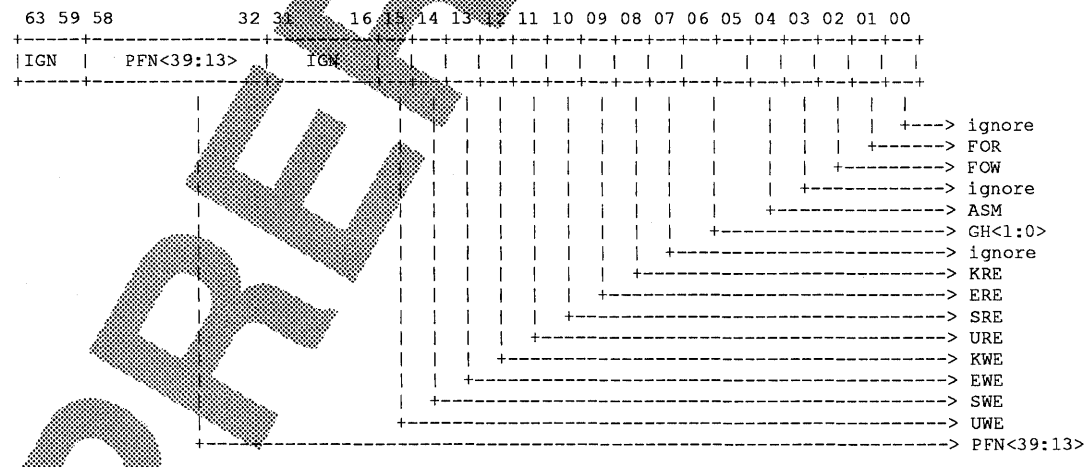
3.9.2.4 Dstream TB PTE, DTB_PTE

The DTB_PTE register is a read/write register representing the 64-entry DTB page table entries. The entry to be written is chosen by a not-last-used algorithm implemented in hardware. Writes to the DTB_PTE use the memory format bit positions as described in the Alpha SRM with the exception that some fields are ignored. In particular the 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 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 will not return the data previously written to the DTB_PTE register.

Reads 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 on a DTB_PTE read. 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 3–37: DTB_PTE, Dstream TB PTE

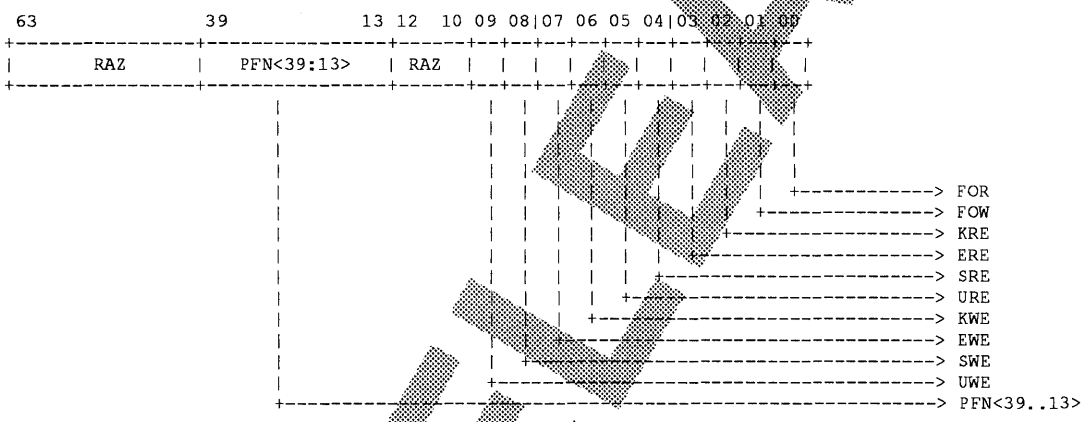


Note: The fields of the Page Table Entry are described in the ALPHA SRM.

3.9.2.5 DTB_PTE_TEMP

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

Figure 3–38: DTB_PTE_TEMP



3.9.2.6 MM_STAT, Dstream MM Fault Status Register

When D-stream faults or Dcache parity errors occur the information about the fault is latched and saved in the MM_STAT register. The VA, VA_FORM and MM_STAT registers are locked against further updates until software reads the VA register. 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 is not unlocked or cleared on reset.

Figure 3–39: MM_STAT, Dstream MM Fault Register

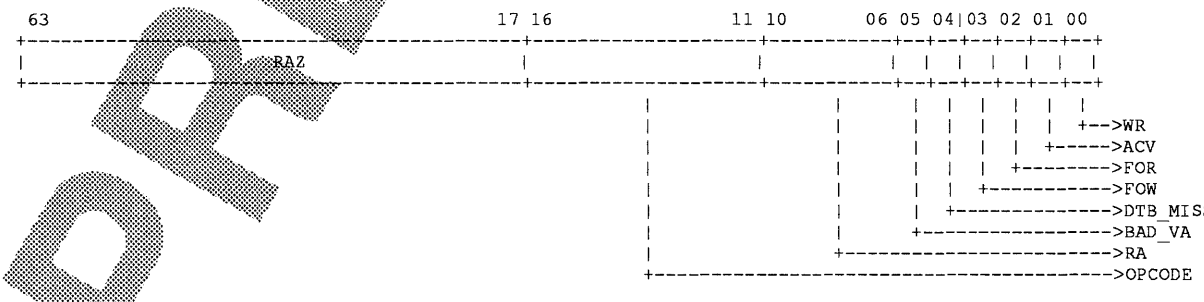


Table 3–19: MM_STAT Field Descriptions

Name	Extent	Type	Description
WR	0	RO	Set if reference which caused error was a write.
ACV	1	RO	Set if reference caused an access violation. Includes bad VA.
FOR	2	RO	Set if reference was a read and the PTE's FOR bit was set.
FOW	3	RO	Set if reference was a write and the PTE's FOW bit was set.
DTB_MISS	4	RO	Set if reference resulted in a DTB miss.
BAD_VA	5	RO	Set if reference had a bad virtual address.
RA	10:6	RO	Ra field of the faulting instruction.
OPCODE	16:11	RO	Opcode field of the faulting instruction.

3.9.2.7 VA, Faulting Virtual Address

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

Figure 3–40: VA, Faulting VA Register

3.9.2.8 VA_FORM, Formatted Virtual Address

VA_FORM contains the virtual page table entry address calculated as a function of the faulting VA and the Virtual Page Table Base (VA and MVPTBR registers). This is done as a performance enhancement to the Dstream TBmiss PALflow. The VA is formatted as a 32-bit PTE when the NT_Mode bit, MCSR<SP0>, is set. VA_FORM is a read-only IPR, and is locked on any D-stream 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 IPR is not unlocked on reset. Figure 3–41 describes VA_FORM when MCSR<SP0> is clear. Figure 3–42 describes VA_FORM when MCSR<SP0> is set.

Figure 3–41: VA_FORM, Formatted VA Register for NT_Mode=0

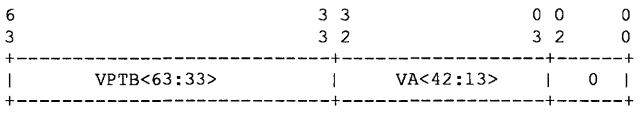


Figure 3–42: VA_FORM, Formatted VA Register, NT_Mode=1

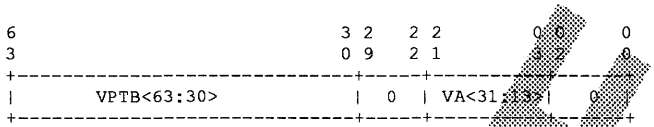


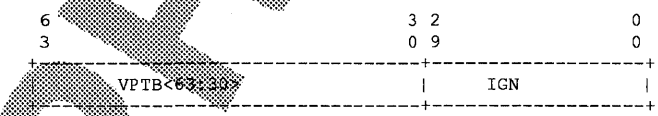
Table 3–20: VA_FORM Field Descriptions

Name	Extent	Type	Description
VA<42:13>	32:03	RO	Subset of the original faulting Virtual Address,NT_Mode=0.
VPTB	63:33	RO	Virtual Page Table Base address as stored in MVPTBR,NT_Mode=0.
VA<31:13>	21:03	RO	Subset of the original faulting Virtual Address,NT_Mode=1.
VPTB	63:30	RO	Virtual Page Table Base address as stored in MVPTBR,NT_Mode=1.

3.9.2.9 MVPTBR, Mbox Virtual Page Table Base Register

MVPTBR contains the virtual address of the base of the page table structure. It is stored in the Mbox to be used in calculating the VA_FORM IPR 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. The MVPTBR is a write-only IPR that looks like this:

Figure 3–43: MVPTBR



3.9.2.10 DC_PERR_STAT, Dcache Parity Error Status

When a Dcache parity error occurs, the error status is latched and saved in the DC_PERR_STAT register. 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<5:2>. The LOCK bit is set and the register is locked against further updates (except for the SEO bit) until software writes a "one" 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 "one" to DC_PERR_STAT<0> to unlock and clear the bit. Note the SEO bit does not get set when Dcache parity errors are detected on both pipes within the same cycle. For this particular situation, the pipe0/pipe1 Dcache parity error status bits will indicate the existence of a second parity error. The DC_PERR_STAT is not unlocked or cleared on reset.

Figure 3–44: DC_PERR_STAT, Dcache Parity Error Status

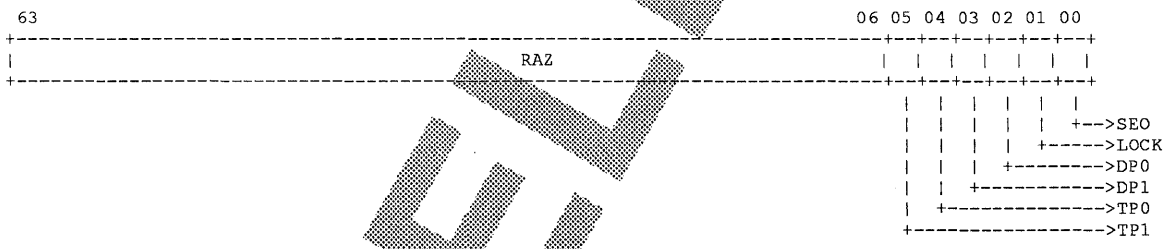


Table 3–21: DC_PERR_STAT Field Descriptions

Name	Extent	Type	Description
SEO	0	W1C	Set if second Dcache parity error occurred in a cycle after the register was locked. The SEO bit will not be set as a result of a second parity error that occurs within the same cycle as the first.
LOCK	1	W1C	Set if parity error detected in Dcache. Bits <5:2> are locked against further updates when this bit is set. Bits <5:2> are cleared when the LOCK bit is cleared.
DP0	2	RO	Set on data parity error in Dcache bank 0.
DP1	3	RO	Set on data parity error in Dcache bank1.
TP0	4	RO	Set on tag parity error in Dcache bank 0.
TP1	5	RO	Set on tag parity error in Dcache bank 1.

3.9.2.11 Dstream TB Invalidate All Process, DTBIAP

This is a write-only register. Any write to this register invalidates all DTB entries in which the ASM bit is equal to zero.

3.9.2.12 Dstream TB Invalidate All, DTBIA

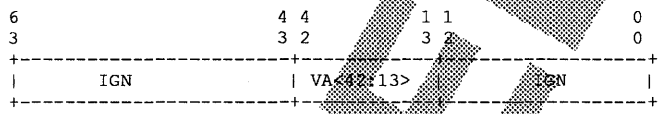
This is a write-only register. Any write to this register invalidates all 64 DTB entries, and resets the DTB NLU pointer to its initial state.

3.9.2.13 DTBIS, Dstream TB Invalidate Single

This is a write-only register. Writing a virtual address to this IPR invalidates the DTB entry that meets any one of the following criteria:

- A DTB entry whose VA field matches DTBIS<42:13> and whose ASN field matches DTB_ASN<63:57>.
- A DTB entry whose VA field matches DTBIS<42:13> and whose ASM bit is set.

Figure 3–45: DTBIS



NOTE

The DTBIS is written before the normal IBOX trap point. The DTB invalidate single operation will be aborted by the IBOX only for the following trap conditions: ITB miss, PC mispredict, or when the HW_MTPR DTBIS is executed in user mode.

3.9.2.14 MCSR, Mbox Control Register

The MCSR register is a read/write register that controls features and records status in the Mbox. This register is cleared on chip reset but not on timeout reset.

Figure 3–46: MCSR, Mbox Control Register

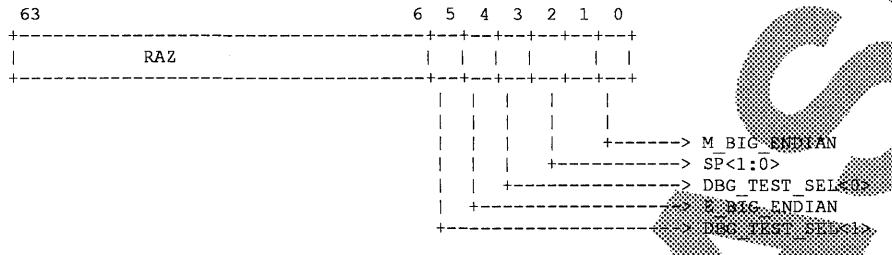


Table 3–22: MCSR Field Descriptions

Name	Extent	Type	Description
M_BIG_ENDIAN	0	RW,0	Mbox Big Endian mode enable. When set, bit 2 of the physical address is inverted for all longword Dstream references.
SP<1:0>	2:1	RW,0	Super page mode enables. 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 super page mapping of D-stream virtual addresses with VA<42:30> = 1FFE(Hex). 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 VA formatting on a read from the VA_FORM IPR. Superpage access is only allowed in kernel mode.
DBG_TEST_SEL<0>	3	RW,0	Debug Test Select. The DBG_TEST_SEL<1:0> bits are used to control the Mbox/Cbox DECchip 21164-AA parallel test port mux selection. When DBG_TEST_SEL<1:0> = (00), the Cbox DBG_DATA<7:0> is selected. When DBG_TEST_SEL<1:0> = (01), the Mbox DCI debug packet is selected. When DBG_TEST_SEL<1:0> = (10), the Mbox MAF_OUT debug packet is selected. When DBG_TEST_SEL<1:0> = (11), the debug packet selection is dynamically controlled by the state of the RFB_DATA_VALID signal from the Cbox. (**Need a reference to the Mbox test packet signal description.**) These bits are used for diagnostic and test purposes only.
E_BIG_ENDIAN	4	RW,0	Ebox Big Endian mode enable. This bit is sent to the Ebox to enable Big Endian support for the EXTxx, MSKxx and INSxx byte instructions. This bit causes the shift amount to be inverted (ones-complemented) prior to the shifter operation.
DBG_TEST_SEL<1>	5	RW,0	Mbox debug packet select. See DBG_TEST_SEL<0>.

3.9.2.15 DC_MODE, Dcache Mode Register

The DC_MODE register 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 3–47: DC_MODE, Dcache Mode Register

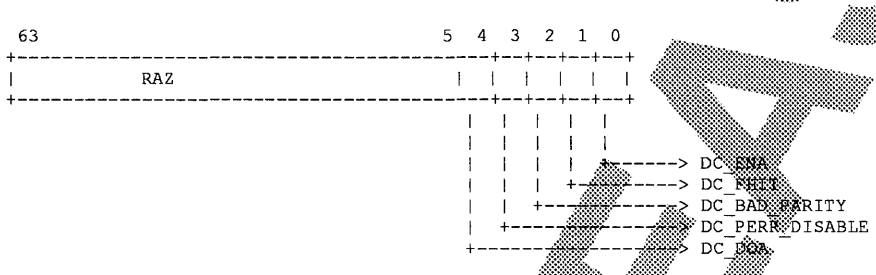


Table 3–23: DC_MODE Field Descriptions

Name	Extent	Type	Description
DC_ENA	0	RW,0	Software Dcache enable. Unless the Dcache has been disabled in hardware (DC_DOA is set), the DC_ENA bit enables the Dcache. (The Dcache is enabled if DC_ENA=1 AND DC_DOA=0). When clear, the Dcache command will be set to NOP, all D-stream references will be forced to miss in the Dcache, and outstanding fills will be blocked from filling the Dcache.
DC_FHIT	1	RW,0	Dcache force hit. When set, this bit forces all D-stream references to hit in the Dcache.
DC_BAD_PARITY	2	RW,0	When set, this bit inverts the data parity inputs to the Dcache on integer stores. This will have the effect of putting bad data parity into the Dcache on integer stores that hit in the Dcache. This bit will have no effect on the tag parity written to the Dcache or the data parity written to the CBOX Write Data Buffer on integer stores. Note: Floating point stores should NOT be issued when this bit is set because it may result in bad parity being written to the CBOX Write Data Buffer.
DC_PERR_DISABLE	3	RW,0	When set, this 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.

Table 3–23 (Cont.): DC_MODE Field Descriptions

Name	Extent	Type	Description
DC_DOA	4	RO	Hardware Dcache Disable. When set, the Dcache is faulty and has been disabled under hardware control (a programmable/readable fuse resides in the MBOX). The Dcache command will be set to NOP, all D-stream references will be forced to miss in the Dcache, and outstanding fills will be blocked from filling the Dcache. When DC_DOA is clear, the Dcache can be enabled under software control (DC_ENA=1). Note the DC_MODE register must be written under software control at least once before the state of the DC_DOA fuse is readable.

NOTE

The DC_MODE bits are only used for diagnostics and test. For normal operation, they will only be supported in the following configuration:

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

3.9.2.16 MAF_MODE, MAF Mode Register

The MAF_MODE register is a read/write register that controls diagnostic and test modes in the Mbox Miss Address File. This register is cleared on chip reset. Bit<5> is also cleared on timeout reset.

Figure 3–48: MAF_MODE, MAF Mode Register

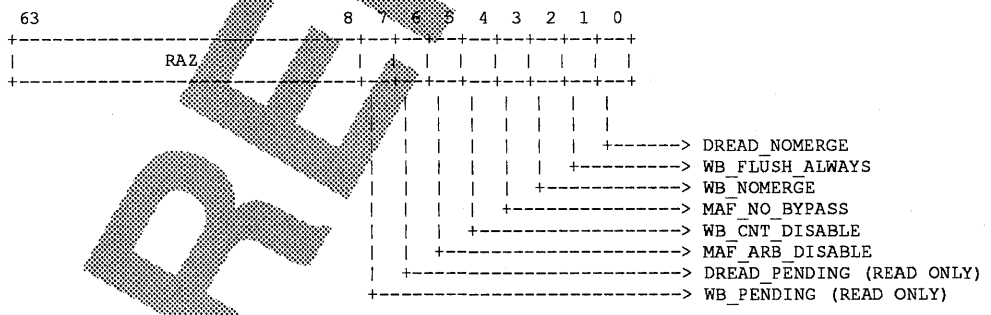


Table 3–24: MAF_MODE Field Descriptions

Name	Extent	Type	Description
DREAD_NOMERGE	0	RW,0	Miss Address File DREAD Merge Disable. When set, this bit disables all merging in the DREAD portion of the miss address file. Any load that is issued when DREAD_NOMERGE is set will be forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if DREAD_NOMERGE is cleared).
WB_FLUSH_ALWAYS	1	RW,0	When set, this bit forces the write buffer to flush whenever there is a valid WB entry.
WB_NOMERGE	2	RW,0	When set, this bit disables all merging in the write buffer. Any store that is issued when WB_NOMERGE is set will be forced to allocate a new entry. Subsequent merging to that entry is not allowed (even if WB_NOMERGE is cleared.)
MAF_NO_BYPASS	3	RW,0	When set, this bit disables Dread bypass requests in the MAF arbiter. All Dread requests will be loaded into the MAF pending queue before arbitration takes place.
WB_CNT_DISABLE	4	RW,0	When set, this bit disables the 64-cycle WB counter in the MAF arbiter. The top entry of the WB will arb at low priority only when a LDx_L is issued or a second WB entry is made.
MAF_ARB_DISABLE	5	RW,0	When set, this bit disables all Dread and WB requests in the MAF arbiter. WB_Reissue, Replay, Iref and MB requests are not blocked from arbitrating for the Scache. This bit is cleared on both timeout and chip reset.
DREAD_PENDING	6	R,0	This bit indicates the status of the MAF Dread file. When set, there are one or more outstanding Dread requests in the MAF file. When clear, there are no outstanding Dread requests.
WB_PENDING	7	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.

NOTE

Bits <5:0> of the MAF_MODE register are only used for diagnostics and test. For normal operation, they are supported in the following configuration:

DREAD_NOMERGE = 0
WB_FLUSH_ALWAYS = 0
WB_NOMERGE = 0
MAF_NO_BYPASS = 0
DREAD_WB_ARB_DISABLE=0
WB_CNT_DISABLE=0

3.9.2.17 DC_FLUSH, Dcache Flush Register

A write to this register clears all the valid bits in both banks of the Dcache.

3.9.2.18 ALT_MODE, Alternate mode

ALT_MODE is a write-only IPR. The AM field specifies the alternate processor mode used by HW_LD and HW_ST instructions.

Figure 3–49: ALT_MODE

Table 3–25: ALT Mode

ALT_MODE<4:3>	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

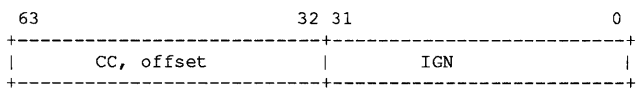
3.9.2.19 CC, Cycle Counter

DECchip 21164-AA supports a cycle counter as described in the Alpha SRM. The low half of the counter, when enabled, increments once each CPU cycle. The upper half of the CC register is the counter offset. CC<63:32> is written on a HW_MTPR to the CC IPR; bits <31:0> are unchanged. CC_CTL<32> is used to enable or disable the cycle counter. The lower half of the cycle counter is written on a HW_MTPR to the CC_CTL IPR.

The CC register is read by the RPCC instruction as defined in the Alpha SRM (The RPCC instruction returns a 64-bit value). The cycle counter is enabled to increment only 3 cycles after the MTPR CC_CTL (with CC_CTL<32> set) is issued. This means that an RPCC instruction issued 4 cycles after an MTPR CC_CTL that enables the counter will read a value that is 1 greater than the initial count. The cycle counter is disabled on chip reset.

The write-only CC Register looks like this:

Figure 3–50: CC, Cycle Counter Register



3.9.2.20 CC_CTL, Cycle Counter Control

The CC_CTL register is a write-only register that is used to write the low 32 bits of the cycle counter and to enable or disable the counter. Bits CC<31:4> are written with the value CC_CTL<31:4> on a HW_MTPR to the CC_CTL register. Bits CC<3:0> 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 3–51: CC_CTL, Cycle Counter Control Register

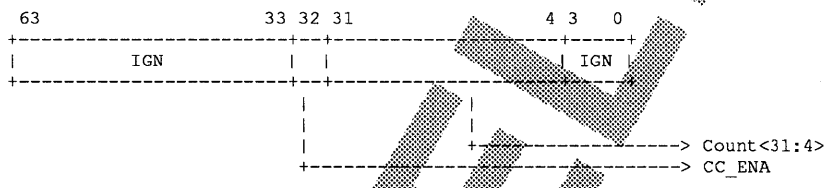


Table 3–26: CC_CTL Field Descriptions

Name	Extent	Type	Description
Count<31:4>	31:4	WO	Cycle count. This value is loaded into bits <31:4> of the CC register.
CC_ENA	32	WO	Cycle Counter enable. When set, this bit enables the CC register to begin incrementing 3 cycles later. An RPCC issued 4 cycles after CC_CTL<32> is written will see the initial count incremented by 1.

3.9.2.21 DC_TEST_CTL, Dcache Test TAG Control Register

The DC_TEST_CTL register is a read/write IPR used exclusively for test and diagnostics.

An address written to this register will be used to index into the Dcache array when reading or writing the DC_TEST_TAG register. See Section 3.9.2.22 for a description of how this register is used.

Figure 3–52: DC_TEST_CTL, Dcache Test TAG Control Register

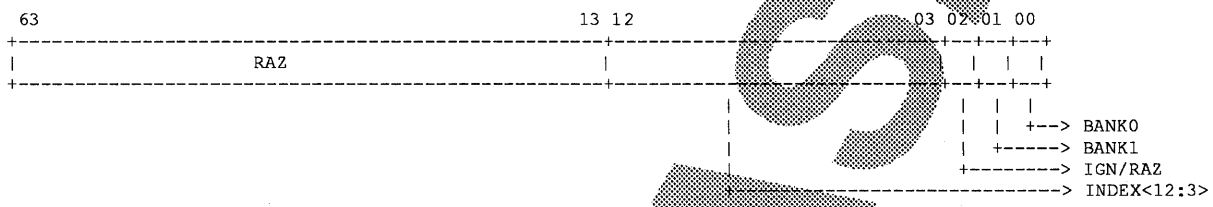


Table 3–27: DC_TEST_CTL Field Descriptions

Name	Extent	Type	Description
BANK0	0	RW	Dcache Bank0 enable. When set, reads from DC_TEST_TAG will return the tag from Dcache bank0 and writes to DC_TEST_TAG will write to Dcache bank0. When clear, reads from DC_TEST_TAG will return the tag from Dcache bank1.
BANK1	1	RW	Dcache Bank1 enable. When set, writes to DC_TEST_TAG will write to Dcache bank1. This bit has no effect on reads.
INDEX	12:3	RW	Dcache tag index. This field is used on reads/writes from/to the DC_TEST_TAG register to index into the Dcache tag array.

3.9.2.22 DC_TEST_TAG, Dcache Test TAG Register

The DC_TEST_TAG register is a read/write IPR used exclusively for test and diagnostics.

When DC_TEST_TAG is read, the value in the DC_TEST_CTL register is used to index into the Dcache and 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 IPR register. A zero value is returned to the integer register file. If BANK0 is set, the read is from Dcache bank0. Otherwise it 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. Data parity bits are not affected by this write (use DC_MODE<DC_BAD_PARITY> and force bit modes). If BANK0 is set, the write is to Dcache bank0. If BANK1 is set, the write is to Dcache bank1. If both are set, the write will occur to both banks.

Figure 3–53: DC_TEST_TAG, Dcache Test TAG Register

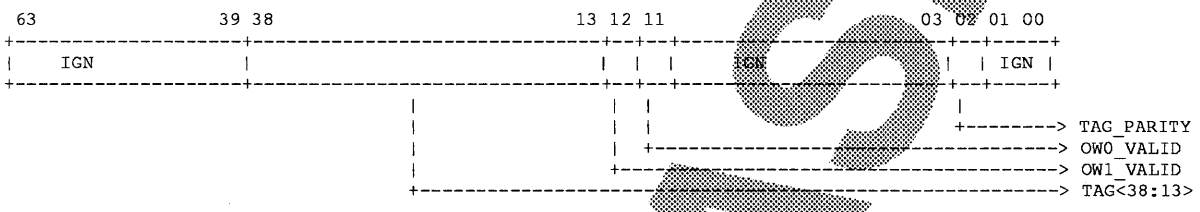


Table 3–28: DC_TEST_TAG Field Descriptions

Name	Extent	Type	Description
TAG_PARITY	2	WO	Tag Parity. This bit refers to the Dcache tag parity bit which covers tag bits 38 through 13 (valid bits not covered).
OW0_VALID	11	WO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low order octaword within a Dcache 32B 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 32B block.
TAG	38:13	WO	Tag<38:13>. This refers to the tag field in the Dcache array. (Note: Bit 39 is not stored in the array)

3.9.2.23 DC_TEST_TAG_TEMP, Dcache Test TAG Temp Register

The DC_TEST_TAG_TEMP register is a read-only IPR used exclusively for test and diagnostics.

Reading the Dcache tag array requires a 2 step process. First, a read 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. A second read of the DC_TEST_TAG_TEMP register will return the Dcache test data to the register file.

Figure 3–54: DC_TEST_TAG_TEMP, Dcache Test TAG Temp Register

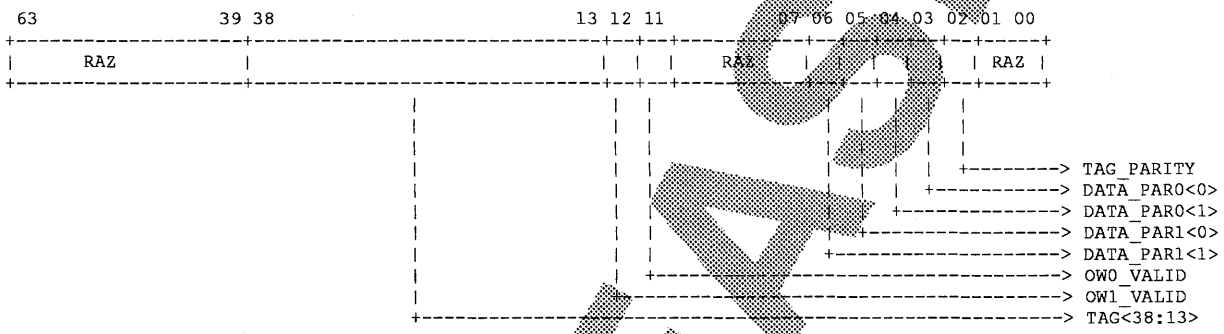


Table 3–29: DC_TEST_TAG_TEMP Field Descriptions

Name	Extent	Type	Description
TAG_PARITY	2	RO	Tag Parity. This bit refers to the Dcache tag parity bit which covers tag bits 38 through 13 (valid bits not covered).
DATA_PAR0<0>	3	RO	Data Parity. This bit refers to the Bank0 Dcache data parity bit which covers the lower longword of data indexed by dc_test_ctl<INDEX>.
DATA_PAR0<1>	4	RO	Data Parity. This bit refers to the Bank0 Dcache data parity bit which covers the upper longword of data indexed by DC_TEST_CTL<INDEX>.
DATA_PAR1<0>	5	RO	Data Parity. This bit refers to the Bank1 Dcache data parity bit which covers the lower longword of data indexed by DC_TEST_CTL<INDEX>.
DATA_PAR1<1>	6	RO	Data Parity. This bit refers to the Bank1 Dcache data parity bit which covers the upper longword of data indexed by DC_TEST_CTL<INDEX>.
OW0_VALID	11	RO	Octaword valid bit 0. This bit refers to the Dcache valid bit for the low order octaword within a Dcache 32B 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 32B block.
TAG	38:13	RO	Tag<38:13>. This refers to the tag field in the Dcache array. (Note: Bit 39 is not stored in the array)

3.9.3 Cbox IPRs

DECchip 21164-AA specific IPRs for controlling Scache, Bcache, System Configuration, and logging error information are listed below. These IPR's cannot be read or written from the system. These IPRs have been placed in the 1MB region of DECchip 21164-AA specific I/O address space ranging from FFFFF00000 to FFFFFFFFFF. Any read or write to undefined IPR in this address space will produce **UNDEFINED** behavior. The operating system should not map any address in this region as writeable in any mode.

Table 3–30: CBOX_IPRS Descriptions

Register	Address	Type[1]	Description
SC_CTL	FF FFF0 00A8	(RW)	Controls Scache behavior.
SC_STAT	FF FFF0 00E8	(R)	Logs Scache related errors.
SC_ADDR	FF FFF0 0188	(R)	Contains the address for Scache related errors.
BC_CONTROL	FF FFF0 0128	(W)	Controls Bcache/System Interface and Bcache testing.
BC_CONFIG	FF FFF0 01C8	(W)	Contains Bcache configuration parameters.
BC_TAG_ADDR	FF FFF0 0108	(R)	Contains tag and control bits for fills from Bcache.
EI_STAT	FF FFF0 0168	(R)	Logs Bcache/system related errors.
EI_ADDR	FF FFF0 0148	(R)	Contains the address for Bcache/system related errors.
FILL_SYN	FF FFF0 0068	(R)	Contains fill syndrome or parity bits for fills from Bcache/memory.
LD_LOCK	FF FFF0 01E8	(R)	Contains the address for LDx_L commands.

3.9.3.1 Scache Control Register, SC_CTL

SC_CTL is a read/write register which controls the behavior of the Scache.

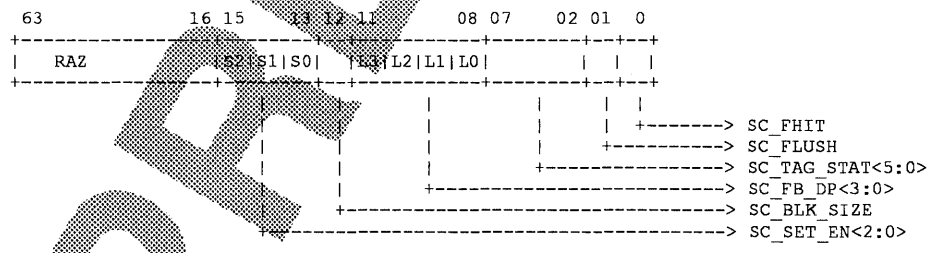


Table 3–31: SC_CTL Field Descriptions

Field	Extent	Type	Description
SC_FHIT	0	(RW,0)	When set, this bit can be used to force cacheable Ld's and ST's to hit in the Scache, irrespective of the tag status bits. Non-cacheable references will not be forced to hit in the Scache and will be driven off-chip. In this mode, only one Scache set may be enabled. The Scache tag and data parity checking will be disabled. For STx, value of status, parity and tag bits specified by SC_TAG_STAT field will be used to write the Scache tag. Scache tag will be the STx address received by the Cbox. Scache tags will be written with the STx address. SC_FHIT bit will be cleared on reset.
SC_FLUSH	1	(RW,0)	When set, all the tag valid bits in the Scache will be cleared everytime SC_CTL ipr is written. SC_FLUSH bit will be cleared on reset.
SC_TAG_STAT	7:2	(RW)	This bit field can be only used in the SC_FHIT mode to write any combination of tag status and parity bits in the Scache. The parity bit can be used to write bad tag parity. The correct value of tag parity is even. These bits will be cleared on reset. See Table 3–32 for the encodings.
SC_FB_DP	11:08	(RW,0)	This field can be used to write bad data parity for the selected LW's within the OW when writing the Scache. If any one of these bits is set to one, then the corresponding LW's computed parity value will be inverted when writing the Scache. For Scache writes, the Cbox allocates two consecutive cycles to write up to two OW's based on the LW valid bits received from the Mbox. Therefore, the same LW parity control bits will be used for writing both OWs. For example, Bit 8 corresponds to LW0 and LW4. This bit field will be cleared on reset.
SC_BLK_SIZE	12	(RW,1)	This bit can be used to select the Scache and Bcache block size to be either 64 byte or 32 byte. The Scache and Bcache will always have identical block size. All the Bcache and memory fills or write transactions will be of the selected block size. At the power up time this bit will be set and the default block size will be 64 byte. When clear, the block size will be 32 byte. This bit must be set to the desired value to reflect the correct Scache/Bcache block size before DECchip 21164-AA does the first cacheable read or write from Bcache or system.

Table 3–31 (Cont.): SC_CTL Field Descriptions

Field	Extent	Type	Description
SC_SET_EN	15:13	(RW,1)	This field will be used to enable the Scache sets. Only *one* or *all* three sets may be enabled at a time. Enabling any combination of *two* sets at a time results in unpredictable behavior. During chip test, bad sets in the Scache will be permanently disabled by blowing away their fuses. "Fuse Blower" mechanism will enable either all sets or one set. Any write to enable permanently disabled set will have no effect. Power-up code must first read this ipr to find out the good sets before enabling them.

Table 3–32: SC_TAG_STAT Field Description

Scache Tag Status<7:2>	Description
SC_TAG_STAT<7:4>	Tag Parity, Valid, Shared, Dirty; bits 7, 6, 5, 4 respectively
SC_TAG_STAT<3:2>	OW Modified bits

3.9.3.2 Scache Status Register, SC_STAT

The Scache status register is read only. It is not cleared or unlocked by reset. Any PAL code read of this register unlocks SC_ADDR and SC_STAT and clears SC_STAT.

If an Scache tag or data parity error is detected during an Scache lookup, the SC_STAT register is locked against further updates from subsequent transactions.

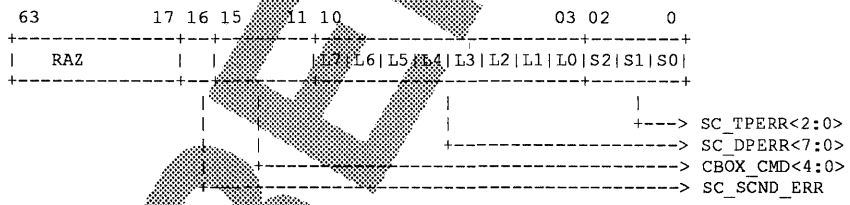


Table 3–33: SC_STAT Field Descriptions

Field	Extent	Type	Description
SC_TPERR	2:0	(RO)	These bits, when set indicate that Scache tag lookup resulted in a tag parity error and identify the set that had the tag parity error.
SC_DPERR	10:3	(RO)	These bits, when set indicate that Scache read resulted in a data parity error and indicate which LW within the two OWs had the data parity error. These bits are loaded if any LW within two OW's read from the Scache during lookup had a data parity error. Bit 3 corresponds to LW0 as shown in the diagram above. If SC_CTL<SC_FHIT> is set, this field will be used for loading the LW parity bits read out from the Scache.
CBOX_CMD	15:11	(RO)	This field indicates the Scache transaction that resulted in a Scache tag or data parity error. This field will be written at the time the actual Scache error bit is written. The Scache transaction may be D-read, I-read, or Write command from the Mbox, Scache victim command, or the system command being serviced. See Table 3–34 for the encodings.
SC_SCND_ERR	16	(RO)	This bit, when set indicates that an Scache transaction resulted in a parity error while the SC_TPERR or SC_DPERR bit was already set from the earlier transaction. This bit is not set for two errors in different octawords of the same transaction.

Table 3–34: SC_CMD Field Descriptions

SC CMD Source<15:14>	SC CMD Encoding<13:11>	Description
1x	110	Set Shared from System
	101	Read Dirty from System
	100	Invalidate from System
	001	Scache Victim
00	001	Scache I-read
01	001	Scache D-read
	011	Scache D-write

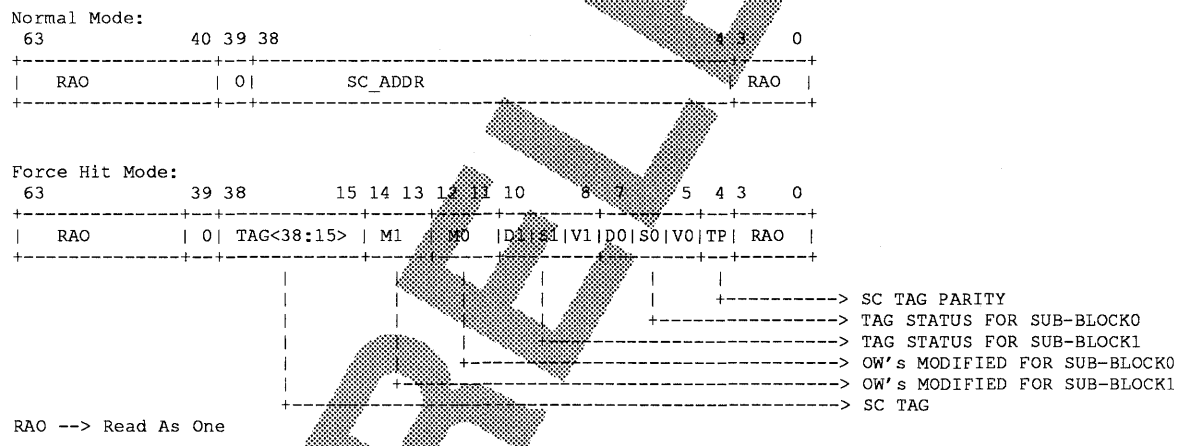
3.9.3.3 Scache Address Register, SC_ADDR

The Scache Address register is read only. It is not cleared or unlocked by reset. The address gets loaded in this register every time the Scache is accessed if one of the error bits in the SC_STAT register is not set. If an Scache tag or data parity error is detected, then this register gets locked preventing further updates. This register is unlocked whenever SC_STAT is read.

For Scache Reads the address bits <39:4> are valid to identify the address being driven to the Scache. Address bit <4> identifies which OW was accessed first. For each Scache lookup, there is one tag access and two data access cycles. If there is a hit, two OWs are read out in consecutive CPU cycles. Tag parity error is detected only while reading the first OW. However, data parity error can be detected on either of the two OWs.

If SC_CTL<SC_FHIT> is set, SC_ADDR is used for storing the tag and status bits. For each tag in the Scache, there are unique valid, shared, dirty and modify bits on a sub-block (32B) basis. Tag and parity bits are common for both sub-blocks. In Force Hit mode, ONLY reads and probes will load the SC_ADDR register. The Scache will drive the tag and status from the set which is enabled. In this mode, tag and data parity checking are disabled and the SC_ADDR and SC_STAT iprs are not locked on a error.

In SC_FHIT mode, to write the Scache and read back the same block and corresponding tag status bits, a minimum of 5 cycle spacing is required between the Scache write and read of the SC_ADDR or SC_STAT.



3.9.3.4 Bcache Control Register, BC_CONTROL

The Bcache control register is write only.

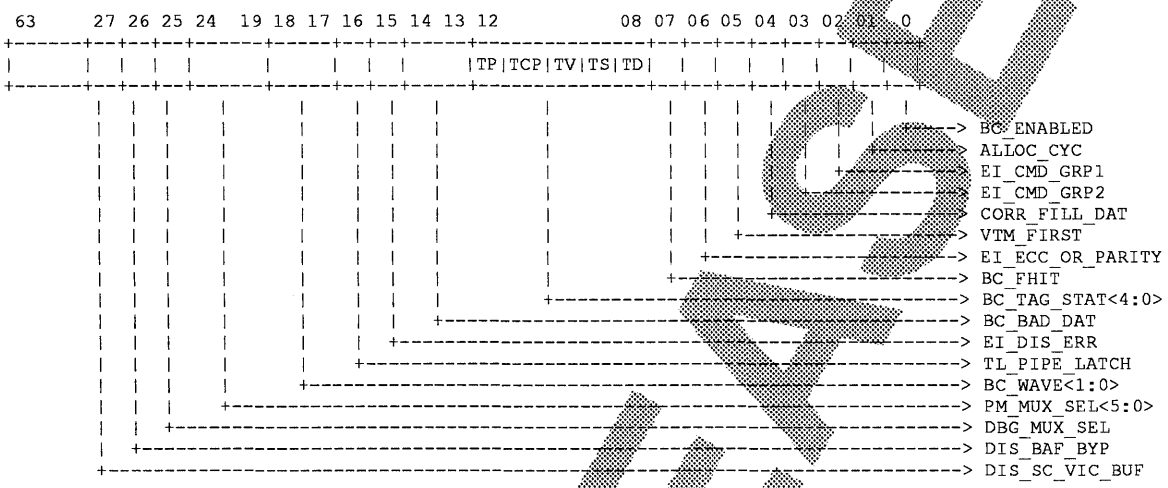


Table 3–35: BC_CONTROL Field Descriptions

Field	Extent	Type	Description
BC_ENABLED	0	(WO,0)	When set, the external Bcache is enabled. When clear, the Bcache is disabled. When the Bcache is disabled, the BIU will neither do external cache reads nor writes. This bit will be cleared on reset.
ALLOC_CYC	1	(WO,0)	When clear, the instruction issue unit will be asked to allocate cycle for non-cacheable LDs. When set, the issue unit will not allocate cycle for non-cacheable LDs. This bit must be clear before reading any Cbox IPR. It can be clear when reading all other IPR's and non-cacheable LDs. This bit will be clear on reset.
EI_CMD_GRP2	2	(WO,0)	When set, the optional commands, LOCK and SET DIRTY will be driven to the DECchip 21164-AA external interface command pins to be acknowledged by the system interface. When clear, it is unpredictable if these commands will be driven to the command pins. However, system should never CACK these commands if this bit is clear.
EI_CMD_GRP3	3	(WO,0)	When set, the MB command will be driven to the DECchip 21164-AA external interface command pins to be acknowledged by the system interface. When clear, it is unpredictable if MB command will be driven to the command pins. However, system should never CACK the command if this bit is clear.

Table 3–35 (Cont.): BC_CONTROL Field Descriptions

Field	Extent	Type	Description
CORR_FILL_DAT	4	(WO,1)	Correct fill data from Bcache or memory, in ECC mode. When this bit is set, fill data from Bcache or memory will first go through error correction logic before being driven to the Scache or Dcache. If the error is correctable, it will be transparent to the machine. When this bit is clear, fill data from Bcache or memory will be directly driven to the Dcache before ECC error is detected. If the error is correctable, corrected data will be returned again, Dcache will be invalidated, and error trap will be taken. This bit will be set on reset.
VTM_FIRST	5	(WO,1)	Set for systems without a victim buffer. On a Bcache miss, DECchip 21164-AA will **first** drive out the victimized block's address on the system address bus, followed by the read miss address and command. Cleared for systems with a victim buffer. If clear, on a Bcache miss with victim, DECchip 21164-AA will first drive out the read miss followed by the victim address and command. This bit will be set on reset.
EI_ECC_OR_PARITY	6	(WO,1)	This bit determines whether to operate the external interface in QW ECC or Byte parity mode. When set, DECchip 21164-AA generates/expects QW ECC on the data check pins. When clear, DECchip 21164-AA generates/expects even byte parity on the data check pins. This bit will be set on reset.
BC_FHIT	7	(WO,0)	Bcache force hit. When this bit is set and the Bcache is enabled, all references in cached space are forced to hit in the Bcache. Fill to the Scache will be forced to be private. Software should turn off BC_CONTROL<2> to allow clean to private transitions without going to the System. For STx, value of status, parity and tag bits specified by BC_TAG_STAT field will be used to write the Bcache tags. Bcache tag and index will be the STx address received by the BIU. It will write the Bcache tag RAM's with the STx address minus the Bcache index. BC_FHIT bit will be cleared on reset.
BC_TAG_STAT	12:8	(WO)	This bit field can be only used in BC_FHIT mode to write any combination of tag status and parity bits in the Bcache. The parity bit can be used to write bad tag parity. These bits will be undefined on reset. See Table 3–36 for the encodings.

Table 3–35 (Cont.): BC_CONTROL Field Descriptions

Field	Extent	Type	Description
BC_BAD_DAT	14:13	(WO,0)	When set, this field can be used to write bad data with correctable or uncorrectable error in ECC mode. When bit <13> is set, data bit <0> and <64> are inverted. When bit <14> is set, data bit <1> and <65> are inverted. When the same OW is read from the Bcache, DECchip 21164-AA will detect correctable/uncorrectable ECC error on both the QWs based on the value of bits <14:13> used when writing. This field will be cleared on reset.
EI_DIS_ERR	15	(WO,1)	When set, this bit causes the DECchip 21164-AA to ignore any ECC (parity) error on a fill data received from the Bcache or memory, or Bcache tag or control parity error. It also ignores system address/command parity error. No machine check is taken when this bit is set. This bit will be set on power-up.
TL_PIPE_LATCH	16	(WO,0)	When set, this bit causes DECchip 21164-AA to pipe the system control pins (ADDR_BUS_REQ_H, CACK_H, and DACK_H) for one sys clock. This bit will be cleared on reset.
BC_WAVE	18:17	(WO,0)	This bit field will determine the number of cycles of wave pipelining that should be used during private reads of the Bcache. Wave pipelining can not be used in 32 byte block systems. To enable wave pipelining, the BC_RD_SPD should be set to the latency of the Bcache read. BC_CONTROL<18:17> should be set to the number of cycles to subtract from BC_RD_SPD to get the Bcache repetition rate. For example, if BC_CONFIG<BC_RD_SPD> is set to 7 and BC_CONTROL<18:17> is set to 2, it will take 7 cycles for valid data to arrive at the pins, but a new read will start every 5 cycles. The read repetition rate must be greater than 3. For example it is not permitted to set BC_CONFIG<BC_RD_SPD> to 5 and BC_CONTROL<18:17> to 2.
PM_MUX_SEL	24:19	(WO,0)	This bit field is used for selecting the BIU parameters to be driven to the two performance monitoring counters in the Ibox. See Table 3–37 for the encodings. See the Performance Monitoring chapter for the detailed functionality. On power-up, this field will be initialized to a value of 0.
DBG_MUX_SEL	25	(WO,0)	This bit field is used for selecting the first group of 8 Cbox signals driven to the Mbox for debug purpose. See XX chapter for the details of these signals. On power-up, this field will be initialized to a value of 0.

Table 3–35 (Cont.): BC_CONTROL Field Descriptions

Field	Extent	Type	Description
DIS_BAF_BYP	26	(WO,0)	When set, speculative Bcache READs are disabled while Scache READs are in progress and there is no pending READ-MISS or WRITE in the BIU. Thus, the Bcache Index pins will change at the rate of Bcache READ speed. When clear, if reads are hitting in the Scache and and there is no pending READ-MISS or WRITE in the BIU, the Bcache Index pins will change every other cpu cycle. On power-up, this field will be initialized to a value of 0.
DIS_SC_VIC_BUF	27	(WO,0)	When set, the on-chip Scache victim buffer will disabled. Any Scache miss, de-allocating a dirty block will write back the dirty block first and then process the miss. This mode is for de-bugging purpose only. On power-up, this field will be initialized to a value of 0.

Table 3–36: BC_TAG_STAT Field Description

Bcache Tag Status<12:8>	Description
BC_TAG_STAT<12>	Parity for Bcache tag
BC_TAG_STAT<11>	Parity for Bcache tag status bits
BC_TAG_STAT<10>	Bcache tag valid bit
BC_TAG_STAT<9>	Bcache tag shared bit
BC_TAG_STAT<8>	Bcache tag dirty bit

Table 3–37: PM_MUX_SEL Field Description

PM_MUX_SEL<21:19>	Counter 1	PM_MUX_SEL<24:22>	Counter 2
0x0	Scache Accesses	0x0	Scache Misses
0x1	Scache Reads	0x1	Scache Read Misses
0x2	Scache Writes	0x2	Scache Write Misses
		0x3	Scache Shared Writ
0x3	Scache Victims		
0x4	Don't Care	0x4	Scache Writes
0x5	Bcache References	0x5	Bcache Misses
0x6	Bcache Victims		
0x7	System Requests	0x6	System Invalidates
		0x7	System Read Reque

3.9.3.5 Bcache Configuration Register, BC_CONFIG

The Bcache configuration register is write only.

Table 3–38: BC_CONFIG Field Descriptions

Field	Extent	Type	Description
BC_SIZE	2:0	(WO,1)	This field is used to indicate the size of the Bcache. On power-up, this field will be initialized to a value of 1MB Bcache. See Table 3–39 for the encodings.
RESERVED	3	(WO,0)	Must Be Zero.
BC_RD_SPD	7:4	(WO,4)	This field is used to indicate to the BIU the read access time of the Bcache, measured in CPU cycles, from the start of a read until data is valid at the input pins. The Bcache read speed must be with in four to ten CPU cycles. On power-up, this field will be initialized to a value of four CPU cycles. For systems without a Bcache, the read speed must be equal to SYS clock to CPU clock ratio. The Bcache read and write speeds must be within three cycles of each other, i.e., Absolute_value(bc_rd_spd - bc_wrt_spd) < 4
BC_WR_SPD	11:8	(WO,4)	This field is used to indicate to the BIU the write time of the Bcache, measured in CPU cycles. The Bcache write speed must be with in four to ten CPU cycles. On power-up, this field will be initialized to a value of four CPU cycles. For systems without a Bcache, the write speed must be equal to SYS clock to CPU clock ratio.
BC_RD_WR_SPC	14:12	(WO,7)	This field is used to indicate to the BIU the number of CPU cycles to wait when switching from a private read to a private write Bcache transaction. For other data movement commands, such as Read Dirty or Fill from memory, it is up to the system to direct system wide data movement in a way that is safe. ONE must be the minimum value for this field. BIU will always insert 2 CPU cycles between private Bcache reads and private Bcache writes in addition to the number of CPU cycles specified by this field. The maximum value should not be greater than the Bcache READ speed when Bcache is enabled. On power-up, this field will be initialized to a read/write spacing of seven CPU cycles.
RESERVED	15	(WO,0)	Must Be Zero.

Table 3–38 (Cont.): BC_CONFIG Field Descriptions

Field	Extent	Type	Description
FILL_WE_OFFSET	18:16	(WO,1)	Bcache write enable pulse offset, from the Sysclock edge, for fills from the system. This field does not affect private writes to Bcache. It is used during fills from the system, when writing the Bcache to determine the number of CPU cycles to wait before driving out the write pulse value as programmed in the BC_WE_CTL field. This field is programmed with a value in the range of one to seven CPU cycles. It must never exceed the sysclock ratio. (E.g., if the sysclock ratio is 3, this field must not be larger than 3.) On power-up, this field is initialized to a write offset value of one CPU cycle.
RESERVED	19	(WO,0)	Must Be Zero.
BC_WE_CTL	28:20	(WO,0)	Bcache write enable control. This field is used to control the timing of the write enable during write or fill. If the bit is set the write pulse is asserted. If the bit is clear the write pulse is not asserted. Each bit corresponds to a CPU cycle. At the start of a Bcache write cycle, write pulse will always be de-asserted for one CPU cycle. After the first cycle, bit <20> of the register is used to assert the write pulse. Each cycle, the next bit will be used to assert the write pulse. On power-up, all bits in this field will be cleared.
RESERVED	35:29	(WO)	Must Be Zero.

Table 3–39: BC_SIZE Field Descriptions

Bcache Size<2:0>	Size
000	Invalid Bcache size
001	1M
010	2M
011	4M [1]
100	8M [1]
101	16M
110	32M
111	64M

[1] Preferred Bcache size for DECchip 21164-AA verification

3.9.3.6 External Interface Status Register, EI_STAT

The External Interface (EI) Status Register is read only. Any PAL code read of this register unlocks and clears it. Read of EI_STAT will also unlock EI_ADDR, BC_TAG, and FILL_SYN registers subject to some restrictions listed below. This register is not unlocked or cleared by reset.

Fill data from Bcache or memory could have correctable (c) or uncorrectable (u) errors in ECC mode. In parity mode, fill data parity errors are treated as uncorrectable hard errors. System address/cmd parity errors are always treated as uncorrectable hard errors irrespective of the mode. The sequence for reading, unlocking, and clearing EI_ADDR, BC_TAG, FILL_SYN, and EI_STAT are as follows:

1. Read EI_ADDR, BC_TAG, FILL_SYN: Can be read in any order. Doesn't unlock or clear any register.
2. Read EI_STAT register: Reading of this register will unlock EI_ADDR, BC_TAG, FILL_SYN registers as described below. EI_STAT will also be unlocked and cleared on read subject to conditions listed below.

Table 3–40: Loading/Locking Rules for External Interface Registers

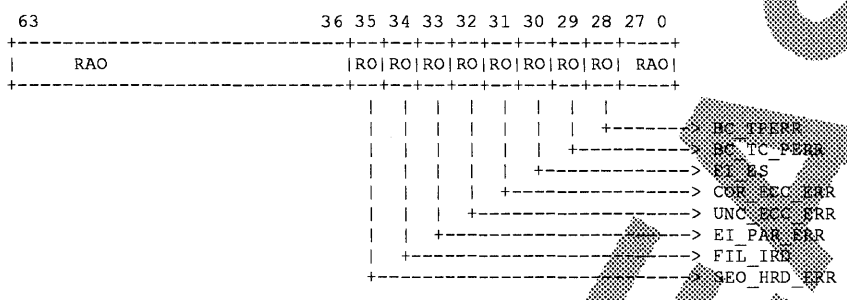
Corr. Error	Uncorr. Error	2nd Hard Error	Load Reg	Lock Reg	Action when EI_STAT is read
0	0	not possible	no	no	clear and unlock everything
1	0	not possible	yes	no	clear and unlock everything
0	1	0	yes	yes	clear and unlock everything
1[1]	1	0	yes	yes	clear (c) bit don't unlock. Transition to (0,1,0) state.
0	1	1	no	already locked	clear and unlock everything
1[1]	1	1	no	already locked	clear (c) bit don't unlock. Transition to (0,1,1) state.

[1] These are special cases. It is possible that when EI_ADDR was read, only correctable error bit is set and registers are not locked. By the time EI_STAT is read, uncorrectable error is detected and the registers get loaded again and locked. The value of EI_ADDR read earlier is no longer valid. Therefore, for the (1, 1, x) case, when EI_STAT is read correctable error bit is cleared and registers are not unlocked or cleared. Software must re-do the ipr read sequence. On the second read, error bits are in (0,1,x) state, all the related iprs are unlocked, and EI_STAT is cleared.

- If the first error is correctable, the registers are loaded but NOT locked. On second correctable error, registers are neither loaded nor locked.
- Registers are locked on first uncorrectable error except the second hard error bit.

- The second hard error bit is set ONLY for an uncorrectable error followed by uncorrectable error. If correctable error follows an uncorrectable, it will not be logged as a second error. Note that Bcache tag parity errors are uncorrectable in this context.

EI_STAT Register



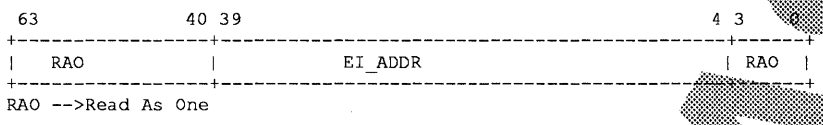
RAO --> Read As One

Table 3–41: EI_STAT Field Descriptions

Field	Extent	Type	Description
BC_TPERR	28	RO	This bit, when set, indicates that a Bcache read encountered bad parity in the tag address RAM.
BC_TC_PERR	29	RO	This bit, when set, indicates that a Bcache read encountered bad parity in the tag control RAM.
EI_ES	30	RO	External interface error source. This field indicates if the error source for fill data is Bcache or memory, or system for address/command parity error. When set, it indicates that the error source is memory or system. If not set, it is Bcache.
COR_ECC_ERR	31	RO	Correctable ECC error. This bit, when set, indicates that a fill data received from outside the CPU contained a correctable ECC error.
UNC_ECC_ERR	32	RO	Uncorrectable ECC error. This bit, when set, indicates that a fill data received from outside the CPU contained an uncorrectable ECC error. In the parity mode it indicates data parity error.
EI_PAR_ERR	33	RO	External Interface address/command parity error. This bit, when set, indicates that an address and command received by the CPU has a parity error.
FIL_IRD	34	RO	This bit is only meaningful when one of the ECC or parity error bits is set. FIL_IRD is set to indicate that the error which caused one of the error bits to get set occurred during an I-ref fill and clear to indicate that the error occurred during a D-ref fill.
SEO_HRD_ERR	35	RO	Second external interface hard error. This field indicates that the fill from Bcache or memory or the system address/command received by the CPU has a hard error while one of the hard error bit in the EI_STAT register is already set.

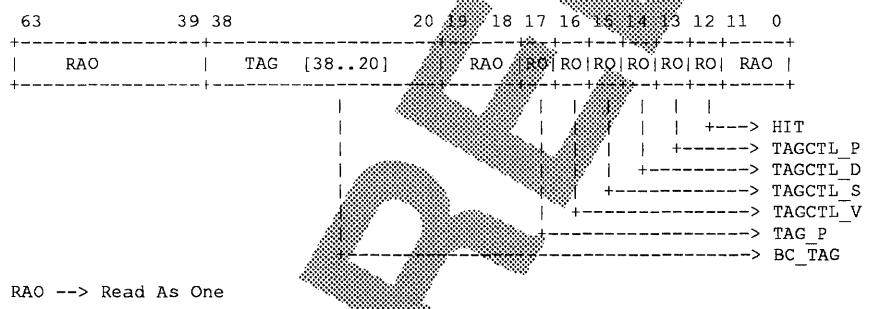
3.9.3.7 External Interface Address Register, EI_ADDR

External Interface Address register. This read-only register contains the physical address associated with errors reported by EI_STAT register. Its content is meaningful only when one of the error bits is set. Read of EI_STAT unlocks EI_ADDR register.



3.9.3.8 Bcache Tag Address Register, BC_TAG_ADDR

BC_TAG_ADDR is a read-only IPR. Unless locked, the BC_TAG_ADDR register is loaded with the results of every Bcache tag read. When a tag or tag control parity error occurs this register is locked against further updates. Software may read this register by using the DECchip 21164-AA specific I/O space address instruction. This register is unlocked whenever EI_STAT register is read. This register is not unlocked by reset.



Unused tag bits in the TAG field of this register are always zero, based on the size of the Bcache as determined by the BC_SIZE field of the BC_CTL register.

3.9.3.9 Fill_Syndrome Register, FILL_SYN

The FILL_SYN register is a 16-bit read-only register. It is loaded but not locked on a correctable ECC error, except that another correctable error does not re-load it. It gets loaded and locked if an uncorrectable ECC error or parity error is recognized during a fill from Bcache or memory as shown in Table 3–40. The FILL_SYN register is unlocked when the EI_STAT register is read. This register is not unlocked by reset.

If the chip is in ECC mode and an ECC error is recognized during a cache fill operation, the syndrome bits associated with the bad quadword are loaded in the FILL_SYN register. FILL_SYN[7..0] contains the syndrome associated with the lower quadword of the octaword, and FILL_SYN[15..8] contain the syndrome associated with the higher quadword of the octaword. A syndrome value of zero means that no errors where found in the associated quadword. See Table 3–42 for a list of syndromes associated with correctable single-bit errors.

If the chip is in parity mode and a parity error is recognized during a cache fill operation, the FILL_SYN register indicates which of the bytes in the octaword got bad parity. FILL_SYNDROME[7..0] is set appropriately to indicate the bytes within the lower quadword that were corrupted and FILL_SYN[15..8] is set to indicate the corrupted bytes within the upper quadword.

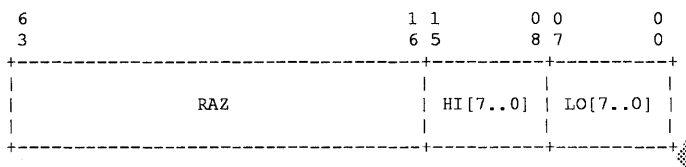


Table 3–42: Syndromes For Single-Bit Errors

Data Bit	Syndrome(Hex)	Check Bit	Syndrome(Hex)
00	CE	00	01
01	CB	01	02
02	D3	02	04
03	D5	03	08
04	D6	04	10
05	D9	05	20
06	DA	06	40
07	DC	07	80
08	23		
09	25		
10	26		
11	29		
12	2A		
13	2C		
14	31		
15	34		
16	0E		
17	0B		
18	13		
19	15		
20	16		
21	19		
22	1A		
23	1C		
24	E3		
25	E5		

Table 3–42 (Cont.): Syndromes For Single-Bit Errors

Data Bit	Syndrome(Hex)	Check Bit	Syndrome(Hex)
26	E6		
27	E9		
28	EA		
29	EC		
30	F1		
31	F4		
32	4F		
33	4A		
34	52		
35	54		
36	57		
37	58		
38	5B		
39	5D		
40	A2		
41	A4		
42	A7		
43	A8		
44	AB		
45	AD		
46	B0		
47	B5		
48	8F		
49	8A		
50	92		
51	94		
52	97		
53	98		
54	9B		
55	9D		
56	62		
57	64		
58	67		
59	68		
60	6B		
61	6D		

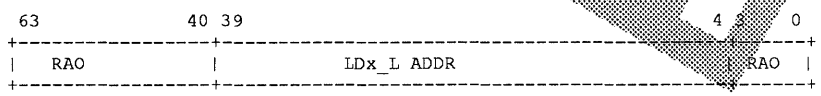
Table 3–42 (Cont.): Syndromes For Single-Bit Errors

Data Bit	Syndrome(Hex)	Check Bit	Syndrome(Hex)
62	70		
63	75		

3.9.3.10 Load Lock Register, LD_LOCK

The Load Lock register is read only. It can be read by PAL code for diagnostic purpose. It is not cleared by reset.

The address gets loaded in this register whenever the Cbox receives LDx_L command and the Scache is accessed. This register is always loaded irrespective of hit/miss or any parity error in the Scache.



RAO -->Read As One

3.9.4 PAL Restrictions

3.9.4.1 Definitions

The list of Mbox instructions is: LDx, LDQ_U, LDx_L, HW_LD, STx, STQ_U, STx_C, HW_ST, and FETCHx. The list of virtual Mbox instructions is: LDx, LDQ_U, LDx_L, HW_LD (virtual), STx, STQ_U, STx_C, HW_ST (virtual), and FETCHx. Load instructions are LDx, LDQ_U, LDx_L, and HW_LD. Store instructions are STx, STQ_U, STx_C, HW_ST.

Table 3–43: PAL Restrictions Table

The following in cycle 0:	Restrictions(note:numbers refer to cycle number):	Y if checked by PVC:
CALL_PAL entry	No HW_REI or HW_REI_STALL in cycle 0	
	No MFPR EXC_ADDR in cycle 0,1	
PALshadow write	No HW_REI or HW_REI_STALL in 0, 1	
HW_LD, lock bit set	PAL must slot to E0	
	No other Mbox instruction in 0	
HW_LD, VPTE bit set	No other virtual reference in 0	
Any Load instruction	No Mbox MTPR or MFPR in 0	Y
	No MFPR MAF_MODE in 1,2	Y
	No MFPR DC_PERR_STAT in 1,2	Y

Table 3–43 (Cont.): PAL Restrictions Table

The following in cycle 0:	Restrictions(note:numbers refer to cycle number):	Y if checked by PVC:
	No MFPR DC_TEST_TAG slotted in 0	
Any Store instruction	No MFPR DC_PERR_STAT in 1,2	Y
Any Virtual Mbox instruction	No MTPR DTBIS in 1	Y
Any Mbox instruction or WMB, if it traps	MTPR any Ibox IPR not aborted in 0,1 (except that EXC_ADDR is updated with correct faulting PC)	
	MTPR DTBIS not aborted in 0,1	Y
Any Ibox trap except pc mispred, itbmiss, or OPCDEC due to user mode	MTPR DTBIS not aborted in 0,1	
HW_REI_STALL	Only 1 HW_REI_STALL in an aligned block of 4 instructions	
MTPR any undefined IPR number	Illegal in any cycle	
ARITH trap entry	No MFPR EXC_SUM or EXC_MASK in cycle 0,1	
Machine_check trap entry	No register file read or write in 0,1,2,3,4,5,6,7	
	No MFPR EXC_SUM or EXC_MASK in cycle 0,1	
MTPR Any Ibox IPR (including PALtemps)	No MFPR same IPR in cycle 1,2	Y
	No Floating Point conditional branch in 0	
	No FEN or OPCDEC instruction in 0	
MTPR ASTRR, ASTER, SIRR, SICR	No MFPR INTID in 0,1,2,3,4	Y
MTPR EXC_ADDR	No HW_REI in cycle 0,1	Y
MTPR IC_FLUSH_CTL	Must be followed by 33 NOPs	
MTPR ICSR: HWE, FPE	No HW_REI in 0,1,2	Y
MTPR ICSR: SPE, FMS	If HW_REI_STALL, then no HW_REI_STALL in 0,1	Y
	If HW_REI, then no HW_REI in 0,1,2,3,4	Y
MTPR ICSR: SPE	Must flush Icache	
MTPR ICSR: SDE	No PALshadow read/write in 0,1,2,3	
	No HW_REI in 0,1,2	Y
MTPR ITB_ASN	Must be followed by HW_REI_STALL	
	No HW_REI_STALL in cycle 0,1,2,3,4	Y
MTPR ITB_PTE	Must be followed by HW_REI_STALL	
MTPR ITB_IAP, ITB_IS, ITB_IA	Must be followed by HW_REI_STALL	
MTPR ITB_IS	HW_REI_STALL must be in the same Istream octaword	
MTPR IVPTBR	No MFPR IFAULT_VA_FORM in 0,1,2	Y
MTPR PAL_BASE	No CALL_PAL in 0,1,2,3,4,5,6,7	Y
	No HW_REI in 0,1,2,3,4,5,6	Y
MTPR PS	No HW_REI in 0,1,2	Y

Table 3–43 (Cont.): PAL Restrictions Table

The following in cycle 0:	Restrictions(note:numbers refer to cycle number):	Y if checked by PVC:
	No priv. CALL_PAL in 0,1,2,3	
MTPR CC, CC_CTL	No RPCC in 0,1,2	Y
MTPR DC_FLUSH	No Mbox instructions in 1,2	Y
	No outstanding fills in 0	
MTPR DC_MODE	No Mbox instructions in 1,2,3,4	Y
	No MFPR DC_MODE in 1,2	Y
	No outstanding fills in 0	
MTPR DC_PERR_STAT	No load or store instructions in 1	Y
	No MFPR DC_PERR_STAT in 1,2	Y
MTPR DC_TEST_CTL	No MFPR DC_TEST_TAG in 1,2,3	Y
	No MFPR DC_TEST_CTL issued or slotted in 1,2	
MTPR DC_TEST_TAG	No outstanding DC fills in 0	
	No MFPR DC_TEST_TAG in 1,2,3	Y
MTPR DTB_ASN	No virtual Mbox instructions in 1,2,3	Y
MTPR DTB_CM, ALT_MODE	No virtual Mbox instructions in 1,2	Y
MTPR DTB_PTE	No virtual Mbox instructions in 2	Y
	No MTPR DTB_ASN, DTB_CM, ALT_MODE, MCSR, MAF_MODE, DC_MODE, DC_PERR_STAT, DC_TEST_CTL, DC_TEST_TAG in 2	Y
MTPR DTB_TAG	No virtual Mbox instructions in 1,2,3	Y
	No MTPR DTB_TAG in 1	Y
	No MFPR DTB_PTE in 1,2	Y
	No MTPR DTBIS in 1,2	Y
MTPR DTBIAP, DTBIA	No virtual Mbox instructions in 1,2,3	Y
	MTPR DTBIS in 0,1,2	Y
MTPR DTBIA	No MFPR DTB_PTE in 1	Y
MTPR MAF_MODE	No Mbox instructions in 1,2,3	Y
	No WMB in 1,2,3	Y
	No MFPR MAF_MODE in 1,2	Y
MTPR MCSR	No virtual Mbox instructions in 0,1,2,3,4	Y
	No MFPR MCSR in 1,2	Y
	No MFPR VA_FORM in 1,2,3	Y
MTPR MVPTBR	No MFPR VA_FORM in 1,2	Y
MFPR DC_TEST_TAG	No outstanding DC fills in 0	
	No MFPR DC_TEST_TAG_TEMP issued or slotted in 1	
	No LDx instructions slotted in 0	

Table 3–43 (Cont.): PAL Restrictions Table

The following in cycle 0:	Restrictions(note:numbers refer to cycle number):	Y if checked by PVC:
	No MTPR DC_TEST_CTL between MFPR DC_TEST_TAG and MFPR DC_TEST_TAG_TEMP	
MFPR DTB_PTE	No Mbox instructions in 0,1	Y
	No MTPR DC_TEST_CTL, DC_TEST_TAG in 0,1	Y
	No MFPR DTB_PTE_TEMP issued or slotted in 1,2,3	
	No MFPR DTB_PTE in 1	Y
	No virtual Mbox instructions in 0,1,2	Y
MFPR VA	Must be done in ARITH, MACHINE CHECK, DTBMISS_SINGLE, UNALIGN, DFAULT traps and ITBMISS flow after the VPTE load	

Table 3–44: Cbox IPR Restrictions Table

Store to SC_CTL, BC_CTL, BC_CONFIG except if no bit is changed other than: BC_CTL<ALLOC_CYC>, BC_CTL<PM_MUX_SEL>, or BC_CTL<DBG_MUX_SEL>	Must be preceded by MB Must be followed by MB No concurrent cacheable Istream references No concurrent system commands
Store to BC_CTL that only changes bits: BC_CTL<ALLOC_CYC>, BC_CTL<PM_MUX_SEL>, or BC_CTL<DBG_MUX_SEL>	Must be preceded by MB Must be followed by MB
Load from SC_STAT	Unlocks SC_ADDR and SC_STAT
Load from EI_STAT	Unlocks EI_ADDR, EI_STAT, FILL_SYN, and BC_TAG_ADDR
Any Cbox IPR address	No Ldx_L or Stx_C
Any undefined Cbox IPR address	No stores
Scache or Bcache in force_hit mode	No Stx_C to cacheable space
Clearing of SC_FHIT in SC_CTL	Must be followed by MB, read of SC_STAT, then MB prior to subsequent store
Clearing of BC_FHIT in BC_CTL	Must be followed by MB, read of EI_STAT, then MB prior to subsequent store

3.10 Revision History

Table 3–45: Revision History

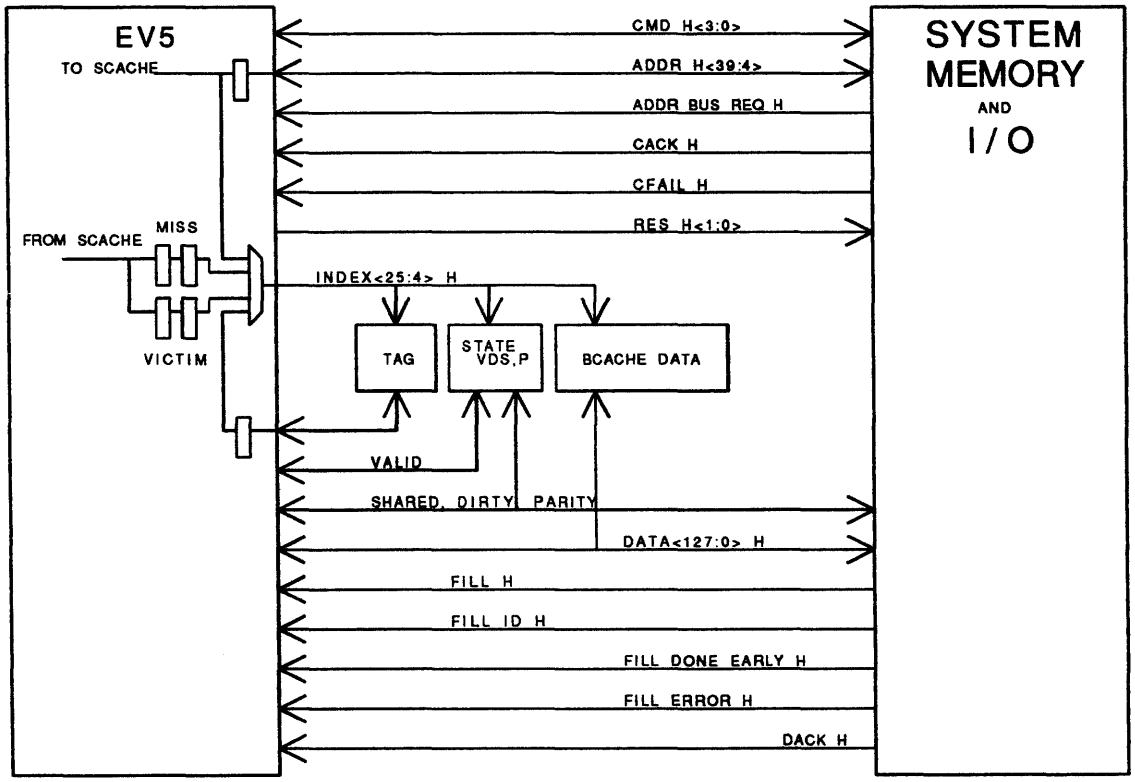
Who	When	Description of change
EV5 team	March 2, 1992	First pass.
EV5 team	November 24, 1992	Update to reflect current design.
EV5 team	September 7,1993	Update to reflect design changes.

Chapter 4

External Interface

4.1 Chip Interface

Figure 4–1: DECchip 21164-AA System Interface



4.1.1 Overview

The DECchip 21164-AA chip is contained in the 503 pin package. All of the extra pins, compared to EV4, are used for power and ground. This means that the system interface will remain a 128 bit bi-directional data bus. The only way to improve the bandwidth of the system interface is to cycle it faster and to use it more often.

The cycle time of the system interface will be some integer multiple of the DECchip 21164-AA cycle time. The minimum multiple is 3x. The maximum multiple is 15x. The tested points between the min. and max. are TBD. The DECchip 21164-AA team will focus on the testing of values that our SYSTEM partners plan to use. Some testing of all possible values will be done.

DECchip 21164-AA can be used to build systems with or without a module level Bcache. The read and write speed of the Bcache can be programmed independantly of the Sysclock ratio and each other. Some care must be taken to make fills and read/read dirty transactions work. The cache system supports a block size of 32 bytes or 64 bytes. The block size is selected by mode bit.

Section 9.1 lists the DECchip 21164-AA signal pins. Figure 4–1 shows a simple picture of the system interface.

Chapter 8 describes the AC requirements for DECchip 21164-AA.

DECchip 21164-AA can take one command/address from the SYSTEM at a time. The Scache and/or Bcache will be probed to determine what must be done with the command. If nothing will be done, the command is ACKed and removed. If a Bcache read, set shared, or invalidate is required it will be done as soon as the Bcache is free. The command will be ACKed at the start of the Bcache transaction.

In general, the DECchip 21164-AA BIU can hold one or two misses and one or two Scache victim address. These four addresses along with the SYSTEM request will ARB for the Bcache. Data movement for the SYSTEM is the highest priority for the Bcache. This includes fill, reading dirty data, invalidates, and set shared. If there are no SYSTEM requests for the Bcache, a DECchip 21164-AA command will be selected.

All transactions between DECchip 21164-AA and the SYSTEM are non-pended, except for fills. DECchip 21164-AA may request up to two fills from memory (if the SYSTEM allows two). Any read or write transaction in the cache must be completed once it is started.

Blocks in the Scache/Bcache that have data movement pending to them will not be read or written by the CPU until the data movement is completed. The SYSTEM will not be prevented from reading or writing blocks in the Scache/Bcache. For example if the CPU has requested a write to a clean block, it will not be allow to access that block until the block until the write completes, but the SYSTEM will always be able to access the block.

The SYSTEM may have one or more Bcache victim buffers. Each time a Bcache victim is produced, DECchip 21164-AA will stop reading the Bcache until the SYSTEM takes the current victim. Bcache operations will then resume.

DECchip 21164-AA requires wrapped reads on INT16 boundaries. The valid wrap orders for 64 byte blocks are selected by bits PA<5:4>, they are:

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

- 3, 2, 1, 0

For 32 byte blocks the valid wrap orders are selected by PA<4>, they are:

- 0, 1
- 1, 0

WRITE BLOCK and WRITE BLOCK LOCK commands from DECchip 21164-AA will not be wrapped. They will always write INT16 zero, one, two, and three. BCACHE VICTIM commands will provide the data with the same wrap order as the read miss that produced them.

4.1.2 Physical Memory Regions

DECchip 21164-AA physical memory is divided into three regions. The first region is the first half of the physical address space. It is treated by DECchip 21164-AA as memory-like. The second region is the second half of the physical address space except for a 1MByte region reserved for Cbox IPRs. It is treated by DECchip 21164-AA as non-cachable. The third region is the 1Mbyte region reserved for Cbox IPRs.

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

DECchip 21164-AA does not cache data accessed in the second and third region of the physical address space. DECchip 21164-AA read accesses in these regions are always 32-byte requests. Load merging is permitted, but the request includes a mask to tell the SYSTEM environment which INT8s are accessed. Write accesses are 32-byte requests, with a mask indicating which INT4s are actually modified. DECchip 21164-AA will never write more than 32-bytes at a time in non-cached space.

DECchip 21164-AA does not emit accesses to the Cbox IPR region if they map to a Cbox IPR. Accesses in this region that are not to a defined Cbox IPR produce UNDEFINED results.

Table 4–1: Physical Memory Regions

Region	Address Range	Description
memory-like	0000000000-7FFFFFFFFF(hex)	Writeback cached, load and store merging allowed
non-cacheable	8000000000-FFFFEFFFFF(hex)	not cached, load merging limited
Cbox IPR region	FFFFF00000-FFFFFFFFFF(hex)	Cbox IPRs, accesses do not appear on the pins unless an undefined location is accessed (which produces UNDEFINED results)

4.1.3 Possible Configurations

The DECchip 21164-AA cache system allows for several system configurations. They can be broken into two classes: those that use the write invalidate cache coherence protocol and those that use the flush based protocol. Table 4–2 shows the components that would make up the system designs that are possible with DECchip 21164-AA.

Table 4–2: System Designs

System Type	Scache	Scache Duplicate Tag	Bcache	Bcache Duplicate Tag	Lock Reg.
Write Invalidate	Yes	No	No	No	No
Write Invalidate	Yes	Yes	No	No	Required
Write Invalidate	Yes	No	Yes	Required	Required
Flush	Yes	No	No	No	No
Flush	Yes	No	Yes	No	No
Flush	Yes	No	Yes	Yes	Required

In a write invalidate based design, DECchip 21164-AA will expect the SYSTEM to use the READ DIRTY, READ DIRTY/INVALIDATE, INVALIDATE, and SET SHARED, commands to keep the state of each block up to date.

In a flush based design, DECchip 21164-AA will expect the READ and FLUSH commands to be used to remove blocks from the cache.

4.1.4 Maintaining Cache Coherence

In a coherent design, DECchip 21164-AA requires the SYSTEM to have some properties to make things work.

DECchip 21164-AA requires the SYSTEM to allow only one change to a block at a time. This means that if DECchip 21164-AA wins the bus to read or write a block, no other node on the bus will be allowed to access that block until the data has been moved.

If DECchip 21164-AA attempts to write a clean/private block of memory, it will send a SET DIRTY command to the SYSTEM. At the same time the SYSTEM might be sending a SET SHARED or INVALIDATE command to DECchip 21164-AA for the same block. The bus is the coherence point in the SYSTEM, so if the bus has already changed the state of the block to shared, setting the dirty bit is the wrong thing to do. DECchip 21164-AA will not resend the SET DIRTY command when the ownership of the ADDRESS/CMD bus is returned. The write will be restarted and use the new tag state to generate a new system request.

It is also possible for the SYSTEM to send an INVALIDATE at the same time DECchip 21164-AA is attempting to do a WRITE BLOCK or WRITE BLOCK LOCK. In this case DECchip 21164-AA will abort the WRITE BLOCK transaction, service the INVALIDATE, and then restart the WRITE BLOCK transaction.

In both of these cases if the SET DIRTY or WRITE BLOCK is started by DECchip 21164-AA, and then interrupted by the SYSTEM, DECchip 21164-AA will resume the same transaction unless the SYSTEM request was to the same block as the request DECchip 21164-AA had started. In this case the DECchip 21164-AA request will be restarted internally by the CPU and it is unpredictible what transaction DECchip 21164-AA will next present to the system.

DECchip 21164-AA will maintain the processors Dcache as a subset of the Scache. If a Bcache is present, the Scache will be maintained as a subset of the Bcache.

The processors Icache is not a subset of any cache and is incoherent with the rest of the cache system.

4.1.5 Cache State

The following tables describe the DECchip 21164-AA multiprocessor cache coherence protocol, a modification of the protocol described in the Laser System Bus Specification Revision 1.2. DECchip 21164-AA will not take an update to a shared block, the block will always be invalidated.

Table 4–3: Cache States

V	S	D	State of cache line assuming tag match
0	X	X	Not valid
1	0	0	Valid for read or write. This cache line contains the only cached copy of the block and the copy in memory is identical to this line.
1	0	1	Valid for read or write. This cache line contains the only cached copy of the block. The contents of the block have been modified more recently than the copy in memory.
1	1	0	Valid for read or write but writes must be broadcast on the bus. This block MAY be in some other CPUs cache.
1	1	1	Valid for read or write but write must be broadcast on the bus. This block MAY be in some other CPUs cache. The contents of the block have been modified more recently than the copy in memory.

Table 4–4: System Actions

System Command	Tag Probe Results	Bus Response	New Cache State	Comments
Read	Miss	~Shared, ~Dirty	No change	
Rd_ex	Miss	~Shared, ~Dirty	No change	
Write	Miss	~Shared, ~Dirty	No change	
Read	Hit, ~Dirty	Shared, ~Dirty	Shared, ~Dirty	
Read	Hit, Dirty	Shared, Dirty	Shared, Dirty	This cache supplies the data
Rd_ex	Hit, ~Dirty	~Shared, ~Dirty	Invalid	
Rd_ex	Hit, Dirty	~Shared, Dirty	Invalid	This cache supplies the data
Write	Hit	~Shared, ~Dirty	Invalid	

Table 4–5: Processor Actions

Processor Command	Cache Probe Results	DECchip 21164-AA ADDR CMD	Bus Response	New Cache State
Read	Invalid	Read Miss	~Shared	~Shared, ~Dirty
Read	Invalid	Read Miss	Shared	Shared, ~Dirty
Write	Invalid	Read Miss Mod	~Shared	~Shared, Dirty
Read	Miss, ~Dirty	Read Miss	~Shared	~Shared, ~Dirty
Read	Miss, ~Dirty	Read Miss	Shared	Shared, ~Dirty
Write	Miss, ~Dirty	Read Miss Mod	~Shared	~Shared, Dirty
Read	Miss, Dirty	Victim, Read Miss	~Shared	~Shared, ~Dirty
Read	Miss, Dirty	Victim, Read Miss	Shared	Shared, ~Dirty
Write	Miss, Dirty	Victim, Read Miss Mod	~Shared	~Shared, Dirty
Read	Hit	NOP	NOP	No change
Write	Hit, Dirty, ~Shared	NOP	NOP	~Shared, Dirty
Write	Hit, Dirty, Shared	Write Block	~Shared	~Shared, ~Dirty
Write	Hit, ~Dirty, ~Shared	Set Dirty	NOP	~Shared, Dirty
Write	Hit, ~Dirty, Shared	Write Block	~Shared	~Shared, ~Dirty

If DECchip 21164-AA requests a READ MISS MOD, DECchip 21164-AA expects the block to be returned ~shared, dirty. However, if the system returns the data shared, ~dirty DECchip 21164-AA will follow with a WRITE BLOCK command. Doing this might expose the system to livelock problems.

4.1.6 DECchip 21164-AA Interface

The interface can be divided into two parts. The SYSTEM interface and the Bcache interface. Both parts share the data bus.

The SYSTEM interface is made up of a bi-directional command and address bus, and several control signals. They are described in Section 4.1.6.1. The Bcache interface signals are described in Section 4.1.6.2.

4.1.6.1 System Interface

These are the signals that make up the SYSTEM interface. All are driven and received by DECchip 21164-AA on the rising edge of Sysclock.

- ADDR_H<39:4>

  Bi-directional

  This is the address of the requested data or operation. If bit 39 is asserted, the reference is to non-cached memory.

- CMD_H<3:0>

  Bi-directional

  Table 4–6 lists the encodings for the commands that DECchip 21164-AA can drive on the CMD bus. Optional commands can be disabled in systems that do not require them. It is unpredictable if DECchip 21164-AA will drive a disabled command to the SYSTEM, however, no CACK should ever be sent for a disabled command.

Table 4–6: DECchip 21164-AA Commands to the System

CMD<3:0>	Command	Optional	Comments
0000	NOP	No	Nothing
0001	LOCK	Yes	New lock register address
0010	FETCH	No	DECchip 21164-AA passing a FETCH to the SYSTEM
0011	FETCH_M	No	DECchip 21164-AA passing a FETCH_M to the SYSTEM
0100	MEMORY BARRIER	Yes	MB instruction
0101	SET DIRTY	Yes	Dirty bit will be set if shared is still clear
0110			spare
0111			spare
1000	READ MISS0	No	Request for data
1001	READ MISS1	No	Request for data
1010	READ MISS MOD0	No	Request for data, modify intent
1011	READ MISS MOD1	No	Request for data, modify intent
1100	BCACHE VICTIM	No	Bcache victim should be removed
1101			spare
1110	WRITE BLOCK	No	Request to write a block
1111	WRITE BLOCK LOCK	No	Request to write a block with lock

Table 4–7 lists the encodings for the commands that the SYSTEM can drive on the CMD bus.

Table 4–7: System Commands to DECchip 21164-AA

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

- ADDR_CMD_PAR_H

  Bi-directional
  This is the odd parity on the current command and address bus. DECchip 21164-AA will take a machine check if a parity error is detected. The SYSTEM should do the same if it detects an error.
- VICTIM_PENDING_H

  Output
  Indicates that the current read miss had generated a victim. Systems may want to hold off requesting the command/address bus until the victim has been removed.
- ADDR_BUS_REQ_H

  Input
  If this signal is asserted before the rising edge of a Sysclock, DECchip 21164-AA will not drive the ADDRESS or CMD busses during the next cycle.
- CACK_H

  Input
  If this signal is asserted before the rising edge of a Sysclock, DECchip 21164-AA will drive the next address and cmd during the next cycle.
- CFAIL_H

  Input
  CFAIL has two uses. It should be used during the CACK cycle of a WRITE_BLOCK_LOCK command to indicate that the write has failed. It can also be used in cycles were CACK is not asserted to force an Ibox timeout event which, in turn, causes a partial reset of DECchip 21164-AA and will trap to the MCHK PAL code entry point.
- RES_H<1:0>

  Output
  Table 4–8 lists the encoding of DECchip 21164-AA responses to SYSTEM requests.

Table 4–8: DECchip 21164-AA Responses to System Commands

RES<1:0>	Command	Comments
00	NOP	Nothing
01	NOACK	Data not found or clean
10	ACK/Scache	Data from Scache
11	ACK/Bcache	Data from Bcache

- INT4_VALID_H<3:0>

  Output

  During writes, these wires are used to indicate which INT4 of data are valid. This is useful for non-cached writes that have been merged in the write buffer. During reads, these wires indicate which INT8 bytes of a 32 byte block need to be read and returned to the processor. This is useful for reads to non-cached memory.

- SCACHE_SET_H<1:0>

  Output

  During a read miss request, these pins will indicate the Scache set number that will be filled when the data is returned. This information can be used by the SYSTEM to maintain a duplicate copy of the Scache tag store.

- FILL_H

  Input

  If this signal is asserted in Sysclock N, DECchip 21164-AA will provide the address indicated by the FILL ID to the Bcache in Sysclock N+2. The Bcache will begin to write in that Sysclock. At the end of the write, DECchip 21164-AA will wait for the next Sysclock and then begin the write again (It may take more than one Sysclock to write the Bcache).

- FILL_ID_H

  Input

  If this signal is asserted in Sysclock N, DECchip 21164-AA will provide the address from miss register 1. If it is deasserted, the address in miss register zero will be used for the fill.

- FILL_ERROR_H

  Input

  If this signal is asserted while a fill is pending from memory, it will indicate to DECchip 21164-AA that system has detected an invalid address or hard error. System will still provide an apparently normal fill sequence with correct ECC/parity though the data is not valid. DECchip 21164-AA will trap to the MCHK PAL code entry point.

- DACK_H

  Input

  For Fills, if this signal is asserted before the Sysclock edge, it will indicate to DECchip 21164-AA that fill data was valid that Sysclock and DECchip 21164-AA should switch to the next address at the Sysclock edge.

  For writes, if this signal is asserted before the Sysclock edge, it indicates that DECchip 21164-AA should provide the next address and data at the Sysclock edge.

- FILL_NOCHECK_H

  Input

  Do not check the parity or ECC for the current data cycle on a fill.

- SYSTEM_LOCK_FLAG_H

  Input

  This wire indicates the state of the system lock flag. During Fills, DECchip 21164-AA will AND the value of the system copy with its own copy to produce the true value of the lock flag.

- IDLE_BC_H

  Input

  When this wire is asserted, DECchip 21164-AA will finish the current Bcache read or write. The CPU will not be allowed to start a new read or write until the wire is deasserted. Systems must assert this wire in time to idle the Bcache before a fill arrives. It can also be used to improve the response time of DECchip 21164-AA to SYSTEM requests.

  The time required to idle the Bcache is a function of the internal design of DECchip 21164-AA, the block size, the read and write speed of the Bcache, the amount of tri-state overlap that must be avoided, and the Sysclock ratio. Take the larger of:

```
   read_idle = 3 + (block_size/16)*BC_RD_SPD + tri-state_ram_turn_off
or
   write_idle = 5 + (block_size/16)*BC_WRT_SPD + tri-state_cpu_turn_off
```

  and round up to the next Sysclock value. This is the number of Sysclocks required between DECchip 21164-AA receiving IDLE_BC until the Bcache will be idle.

  For example if the Sysclock ratio is 6, BC_RD_SPD is 4, BC_WRT_SPD is 5, block size is 32B, and two idle CPU cycle are required to turn off the RAM drivers, for reads, and zero are required to turn off DECchip 21164-AA's write drivers, then it will take max(3+2*4+2, 5+2*5+0)/6 = 3 Sysclocks to idle the cache. If IDLE_BC is asserted in Sysclock N, then the first fill data could be written in Sysclock N+3.

  For FILL requests, IDLE_BC can be de-asserted any time after the fill starts.

- DATA_BUS_REQ_H

  Input

  If this signal is asserted in Sysclock N, DECchip 21164-AA will not drive the data bus in Sysclock N+2. Before asserting this signal the system should assert IDLE_BC for the correct number of cycles. If this signal is deasserted in Sysclock N, DECchip 21164-AA will drive the data bus in Sysclock N+2.

4.1.6.2 Bcache Interface

These signals make up the Bcache interface. Reads and writes of the Bcache that do not involve the SYSTEM will begin on any CPU clock. If the Bcache read or write involves receiving or sending data to the SYSTEM, then the access will begin on a rising Sysclock edge.

- INDEX_H<25:4>

  Output

  These wire are used to index the Bcache.

- DATA_H<127:0>

  Bi-directional

  This bus is used to move data between DECchip 21164-AA, the Bcache, and the SYSTEM.

- DATA_CHECK_H<15:0>

  Bi-directional

  Either even byte parity or INT8 ECC for the current data cycle.

- TAG_DATA_H<38:22>
  Bi-directional
  Bcache tag data bits. This allows for Bcaches in the 4MB to 64MB range.
- TAG_DATA_PAR_H
  Bi-directional
  Odd parity for TAG_DATA_H<38:22>, the SYSTEM should force unused bits to zero.
- TAG_VALID_H
  Bi-directional
  The current tag contains a valid block. DECchip 21164-AA will assert this pin during fills.
- TAG_SHARED_H
  Bi-directional
  The block is in the shared state. During fills the SYSTEM should drive TAG_SHARED_H with the correct value.
- TAG_DIRTY_H
  Bi-directional
  The block is in the dirty state. During fills the SYSTEM should assert this bit if the DECchip 21164-AA request was a READ MISS MOD, and the shared bit is not asserted.
- TAG_CTL_PAR_H
  Bi-directional
  Odd parity for TAG_VALID_H, TAG_SHARED_H, and TAG_DIRTY_H. During fills the system should drive the correct parity based on the state of the V, S and D bits.
- TAG_RAM_OE_H
  Output
  This signal will be asserted by DECchip 21164-AA during any Bcache read.
- TAG_RAM_WE_H
  Output
  This signal will be asserted by DECchip 21164-AA, using the write pulse register contents, during any tag write. During the first CPU cycle of a write, the write pulse will be de-asserted. In the second and following CPU cycles of the write, the write pulse will be asserted if the corresponding bit in the write pulse register is asserted.
- DATA_RAM_OE_H
  Output
  This signal will be asserted by DECchip 21164-AA during any Bcache read.
- DATA_RAM_WE_H
  Output
  This signal will be asserted by DECchip 21164-AA, using the write pulse register contents, during any data write. During the first CPU cycle of a write, the write pulse will be de-asserted. In the second and following CPU cycles of the write, the write pulse will be asserted if the corresponding bit in the write pulse register is asserted.

4.1.7 DECchip 21164-AA Interface Command Descriptions

- FETCH/FETCH_M

  From DECchip 21164-AA
  These commands are issued by DECchip 21164-AA when the FETCH and FETCH_M instructions are executed.

- FLUSH

  From SYSTEM
  The FLUSH command will cause a block to be removed from the DECchip 21164-AA cache system. If the block is not found, DECchip 21164-AA will respond with NOACK. If the block is found and the block is clean, DECchip 21164-AA will respond with NOACK. The block will be invalidated in the Dcache, Scache, and Bcache. If the block is found and dirty, DECchip 21164-AA will respond with ACK/Scache or ACK/Bcache. If the data was found dirty in the Scache it will be driven at the pins in the same Sysclock as the ACK/Scache. If the data is found dirty in the Bcache, the Bcache read will start on the same Sysclock as ACK. The block will be invalidated in the Dcache, Scache, and Bcache.

- LOCK

  From DECchip 21164-AA
  This command is used to load the System lock register. The state of the SYSTEM lock register flag is used on each fill to update the DECchip 21164-AA copy of the lock flag. See Section 4.1.8.12 for the full story.

- MEMORY BARRIER

  From DECchip 21164-AA
  This command is issued by DECchip 21164-AA to synchronize read and write accesses with other processors in the SYSTEM. DECchip 21164-AA issues this command when a MB instruction is executed. DECchip 21164-AA will stop issuing memory reference instructions and wait for the command to be acknowledged before continuing.

- NOP

  From DECchip 21164-AA or SYSTEM
  Nothing. This command should be driven by the owner of the CMD bus if it has nothing to do.

- READ

  From SYSTEM
  The READ command will probe the Scache and Bcache to see if the requested block is present. If the block is present, DECchip 21164-AA will respond with ACK/Scache or ACK/Bcache. If the data is in Scache, the data will be driven on the DATA bus in the same Sysclock as the ACK. If the data is in the Bcache, a Bcache read will begin in the same Sysclock as the ACK. If the block is not present in either cache, DECchip 21164-AA will assert NOACK on the RES wires.

- READ DIRTY

  From SYSTEM
  The READ DIRTY command will probe the Scache to see if the requested block is present and dirty. If the block is not found, or the block is clean, and the SYSTEM does not contain a Bcache, DECchip 21164-AA will respond with a NOACK. If the block is found and dirty in the Scache, DECchip 21164-AA will respond with ACK/Scache and drive the data on the DATA bus. If the block is not found in the Scache, and the SYSTEM contains a Bcache, it is

assumed to be in the Bcache. DECchip 21164-AA will respond with ACK/Bcache, index the Bcache to read the block, and will change the block status to the shared dirty state.

- READ DIRTY INVALIDATE

  From SYSTEM
  This command is identical to the READ DIRTY command except if the block is present it will be invalidated from the caches.

- READ MISSn

  From DECchip 21164-AA
  This command is used to indicate that DECchip 21164-AA has probed its caches and that the addressed block was not present.

- READ MISS MODIFYn

  From DECchip 21164-AA
  This command is used to indicate that DECchip 21164-AA plans to write to the returned cache block. Normally the dirty bit should be set when the tag status is returned to DECchip 21164-AA.

- SET SHARED

  From SYSTEM
  The SET SHARED command is used by the SYSTEM to change the state of a block in the cache system to shared. The shared bit in the Scache will be set if the block is present. The Bcache tag will be written to the shared not dirty state. DECchip 21164-AA assumes that this is ok, because the SYSTEM would have sent a READ DIRTY if the dirty bit were set.

  If the block is found in the Scache, DECchip 21164-AA will respond with ACK/Scache. Otherwise, if the SYSTEM contains a Bcache, the block is assumed to be in the Bcache and DECchip 21164-AA will respond with ACK/Bcache. If the SYSTEM does not contain a Bcache and the block is not found in the Scache, DECchip 21164-AA will respond with a NOACK.

- SET DIRTY

  From DECchip 21164-AA
  DECchip 21164-AA wants to write a clean, private block in its Scache and wants the dirty bit set in the duplicate tag store. The CPU will not proceed with the write until an CACK response is received from the SYSTEM. When the CACK is received, DECchip 21164-AA will attempt to set the dirty bit. If the shared bit is still clear the dirty bit will be set and the write completed. If the shared bit is set the dirty bit will not be set, and DECchip 21164-AA will request a WRITE BLOCK. The copy of the dirty bit in the Bcache will not be updated until the block is removed from the Scache.

- INVALIDATE

  From SYSTEM
  DECchip 21164-AA will probe the Scache and invalidate the block if it is present. If the Bcache is present the block will be changed to the invalid state without probing.

  If the block is found in the Scache, DECchip 21164-AA will respond with ACK/Scache. Otherwise, if the SYSTEM contains a Bcache, the block is assumed to be in the Bcache and DECchip 21164-AA will respond with ACK/Bcache. If the SYSTEM does not contain a Bcache and the block is not found in the Scache, DECchip 21164-AA will respond with a NOACK.

- BCACHE VICTIM

From DECchip 21164-AA
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 that produced the victim will preceed the BCACHE VICTIM command. The VICTIM_PENDING wire will be asserted during the read miss command to indicate that a BCACHE victim command is waiting, and that the Bcache is starting the read of the victim data.

If the SYSTEM does not have a victim buffer the BCACHE VICTIM command will preceed the read miss commands. The BCACHE VICTIM command will be driven, along with the address of the victim. At the same time the Bcache will be read to provide the victim data.

- WRITE BLOCK

  From DECchip 21164-AA
  DECchip 21164-AA wants to write a block of data back to memory. DECchip 21164-AA will drive the command, address, and first INT16 of data on a Sysclock edge. DECchip 21164-AA will output the next INT16 of data when a DACK is received. When the SYSTEM asserts CACK, DECchip 21164-AA will remove the command and address from the pins and begin the write of the Scache. CACK can be asserted before all the data is removed.

- WRITE BLOCK LOCK

  From DECchip 21164-AA
  This command is the same as a WRITE BLOCK except that a CFAIL may be asserted by the SYSTEM to indicate that the data can not be written. this command is only used for STx_C in non-cached space.

4.1.8 Transactions

This section will describe how the commands are used to move data in and out of DECchip 21164-AA and its cache system.

Figure 4–1 shows the resources that can be used by the CPU and SYSTEM. They are listed here.

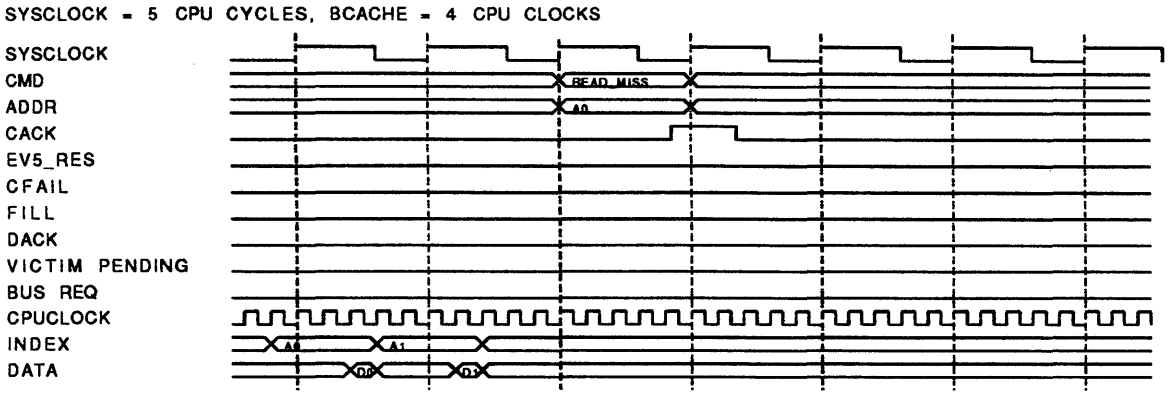
- 2 CPU commands and addresses
- 2 Scache victim address
- 2 System command and address

4.1.8.1 Read Miss

DECchip 21164-AA will start a Bcache read on any CPU clock. The index will be asserted to the RAM for a programmable number of CPU cycles in the range of 4 to 10. The tag will be accessed at the same time. At the end of the first read, DECchip 21164-AA will latch the data and tag information and begin the read of the next 16 bytes of data. The tag will be checked for a hit. If there is a miss, a READ MISS or READ_MISS_MOD command along with the address will be queued to the CMD/ADDRESS bus. It will appear on the pins at the next Sysclock edge. Figure 4–2 shows the timing of a Bcache read and the resulting READ MISS request.

Figure 4–2 shows the READ MISS command being CACKed as soon as it is sent. This will allow DECchip 21164-AA to make additional READ MISS requests. It is also possible for the SYSTEM to defer the CACK until the fill data is returned. This allows the SYSTEM to use CMD<0> for the value of FILL_ID. The CACK should arrive no later than the last fill DACK.

Figure 4–2: Read Miss



4.1.8.2 Read Miss with victim

DECchip 21164-AA supports two models for removing displaced dirty blocks from the Bcache. The first assumes that the SYSTEM does not contain a victim buffer. In this case the victim must be read from the Bcache before the new block can be requested. In the second case, if the SYSTEM does have a victim buffer, DECchip 21164-AA will request the new block from memory while it starts to read the victim from the Bcache. The victim command and address will follow the miss request.

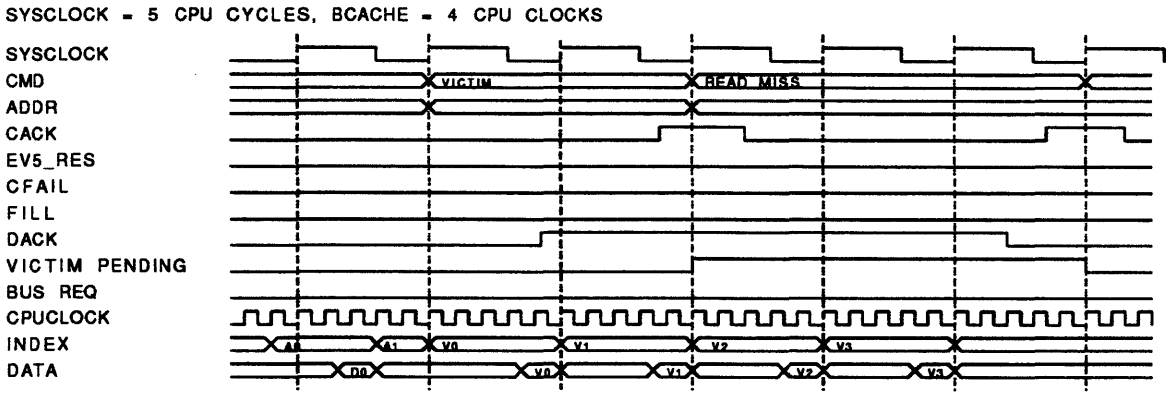
In either case, DECchip 21164-AA treates a miss/victim as single transaction. If the assertion of ADDR_BUS_REQ or IDLE_BC causes the BIU sequencer to reset, both the miss and victim transactions will be restarted from the begining. For example if DECchip 21164-AA is operating in victim first mode and it sends a BCACHE VICTIM command to the SYSTEM and then the system sends an INVALIDATE to DECchip 21164-AA, DECchip 21164-AA will restart the Bcache read and resend the BCACHE VICTIM command and data and then the READ_MISS.

The next two sections describe each of these methods of victim processing.

4.1.8.2.1 Without a Victim Buffer

If the SYSTEM does not contain a victim buffer, DECchip 21164-AA will stop reading the Bcache as soon as the miss is detected. This will be sometime during the second read. A BCACHE VICTIM command will be asserted at the next Sysclock along with the victim address. A Bcache read of the victim will also be started at the Sysclock edge. When the DACK is received for the first part of the victim, DECchip 21164-AA will begin reading the next part of the victim. CACK can be sent anytime during the processing of the victim. DECchip 21164-AA will send out the READ MISS command in the Sysclock after the CACK is received. Figure 4–3 shows the timing of a victim being removed.

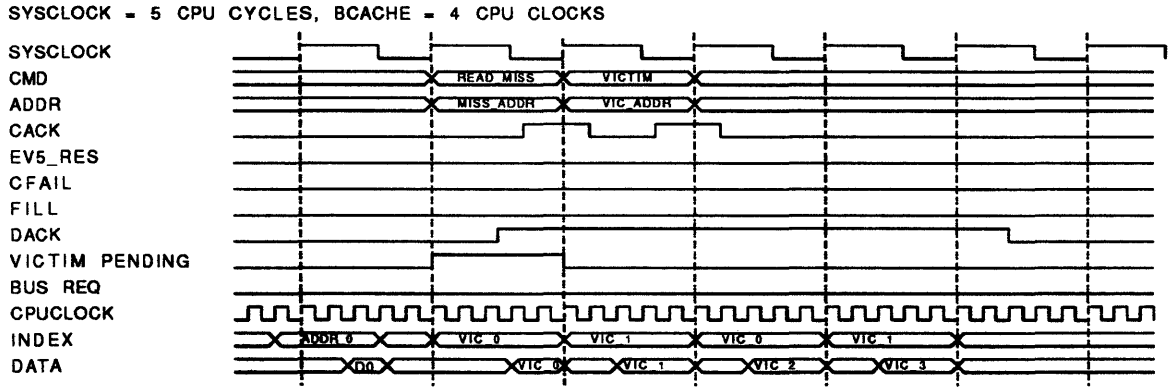
Figure 4–3: Read Miss with Victim



4.1.8.2.2 With a Victim Buffer

When the miss is detected, if the SYSTEM has a victim buffer, DECchip 21164-AA will wait for the next Sysclock edge and then assert a READ MISS command, the read miss address, the VICTIM_PENDING wire, and index the Bcache to begin the read of the victim. When the SYSTEM asserts CACK, DECchip 21164-AA will send out the BCACHE VICTIM command along with the victim address. Each assertion of DACK will cause the Bcache index to advance to the next part of the block. Figure 4–4 shows the timing of a read miss with a victim.

Figure 4–4: Read Miss with Victim Buffer



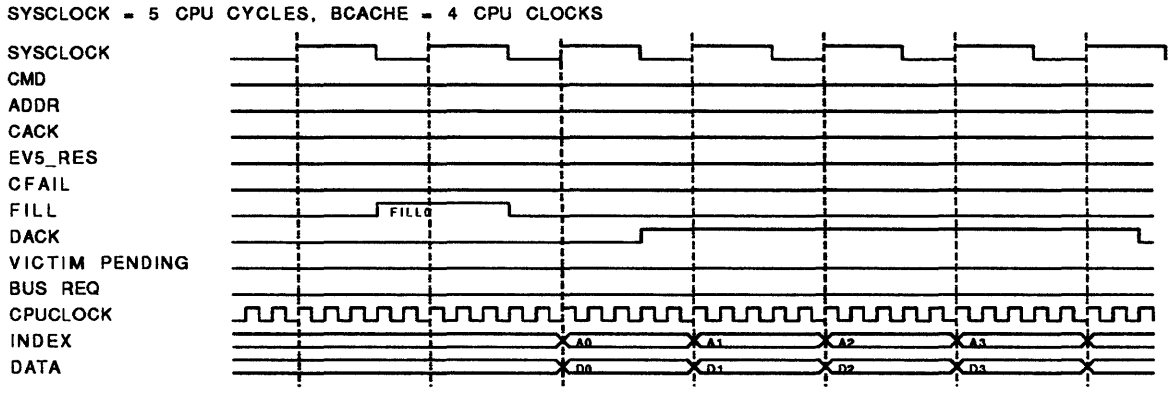
4.1.8.3 Fill

The fill wires are used to control the return of fill data to DECchip 21164-AA and the Bcache if it is present. The IDLE_BC_H wire 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). FILL_H should be asserted at least two Sysclocks before the fill data arrives. The FILL_ID_H wire should

be asserted at the same time to indicate if the fill will be for a READ MISS0 or READ MISS1. DECchip 21164-AA will use this information to select the correct fill address. If FILL and FILL_ID are asserted at the end of Sysclock N, then DECchip 21164-AA will assert the Bcache index and begin a Bcache write during Sysclock N+2. The SYSTEM should drive the data onto the DATA bus and assert DACK before the end of the Sysclock cycle. This will cause DECchip 21164-AA to move on to the next fill address and begin another write of the Bcache. The SYSTEM must allocate the right number of Sysclock cycles to allow the writing of the Bcache if it is present. For example if the Bcache requires 17ns to write and the Sysclock is 12ns, two Sysclock cycles will be required for each write.

During the first fill of a block, the SYSTEM should also drive the correct values on the TAG_SHARED, TAG_DIRTY, and, TAG_PARITY wires. DECchip 21164-AA will assert TAG_VALID and write the Bcache tag store during the first fill.

Figure 4–5: Fill



4.1.8.4 Write Block

The WRITE BLOCK command will be used to complete writes to shared data, to remove Scache victims in Bcache-less systems, and to complete writes to non-cached memory.

The WRITE BLOCK LOCK command follows the same protocol. The LOCK qualifier might allow the SYSTEM to be more aggressive on non-interlocked writes.

DECchip 21164-AA will assert the WRITE BLOCK command along with the address and the first 16 bytes of data at the start of a Sysclock. If the SYSTEM takes away the ownership of the CMD and ADDRESS bus, DECchip 21164-AA will hold on to the write and wait for the ownership of the bus to be returned. If the block in question is invalidated, the write will be restarted by the CPU and will result in the READ MISS MOD request instead.

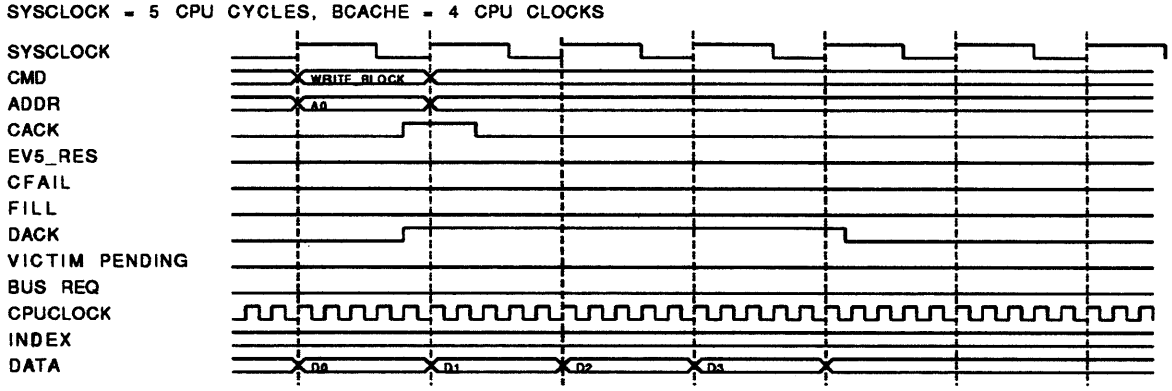
When the SYSTEM has taken the first part of the data it should assert DACK. This will cause DECchip 21164-AA to drive the next 16 bytes of data at the next Sysclock edge.

If the SYSTEM asserts CACK, DECchip 21164-AA will output the next command in the next Sysclock. Receiving the CACK indicates to DECchip 21164-AA that the write will be taken and that it is safe to update the Scache with write data.

During each cycle the INT4_VALID_H<3:0> wires will indicate which INT4 parts of the write are really being written by the processor. For writes to cached memory, all of the data will be valid. For writes to non-cached memory, only those INT4 with the INT4_VALID_H<n> signal asserted are valid.

Figure 4–6 shows the timing of a write block command.

Figure 4–6: Write Block



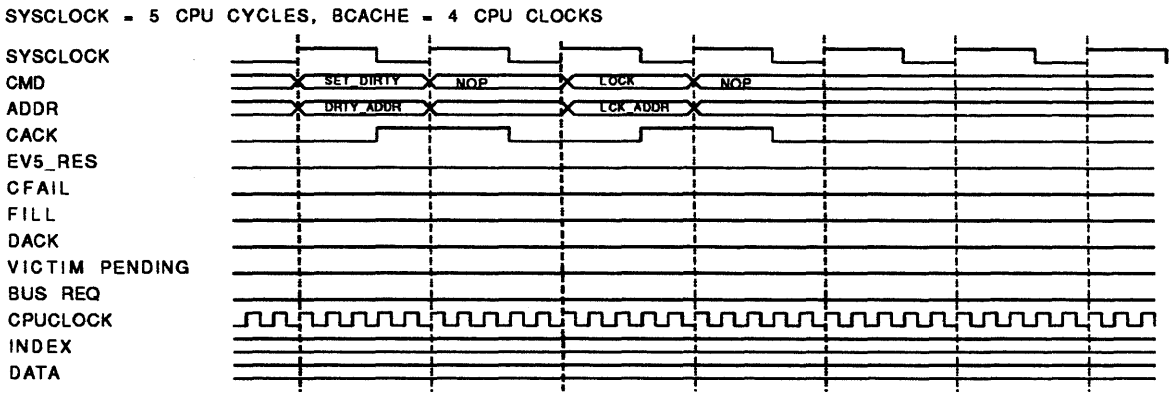
4.1.8.5 Set Dirty, Lock

Figure 4–7 shows the timing of a SET DIRTY command and a LOCK command.

The SET DIRTY command is used by DECchip 21164-AA to inform a duplicate tag store that a cached block is changing from the not-shared clean state to the not-shared dirty state. When the CACK is received from the SYSTEM, DECchip 21164-AA will attempt to set the dirty bit. If the shared bit has been set since the original probe of the Scache, or the block has been invalidated, DECchip 21164-AA will restart the write. This will produce a new request which reflects the new state of the block. If the block is still in the not-shared clean state, the dirty bit will be set and the write completed.

The LOCK command is used by DECchip 21164-AA to pass the address of a LDx_L to the SYSTEM. A system lock register is required in any system that filters write traffic with a duplicate tag store. If the locked block is displaced from the DECchip 21164-AA caches, DECchip 21164-AA will use the value of the system lock register to determine if the LDx_L/STx_C sequence should pass or fail.

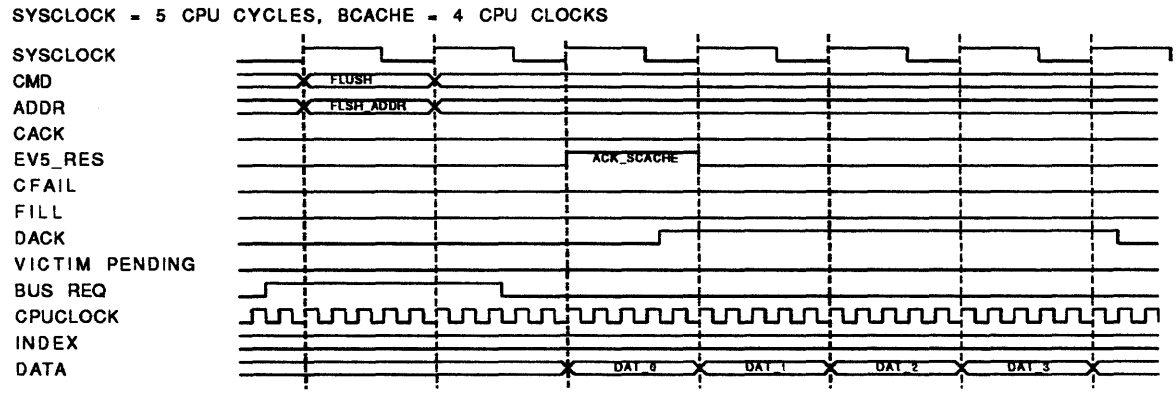
Figure 4–7: Set Dirty, and Lock



4.1.8.6 Flush

The FLUSH command can be used to remove blocks from the DECchip 21164-AA cache system. If the block is dirty, the block will be read from the caches to allow the updating of memory. Figure 4–8 shows the timing of a FLUSH transaction.

Figure 4–8: Flush

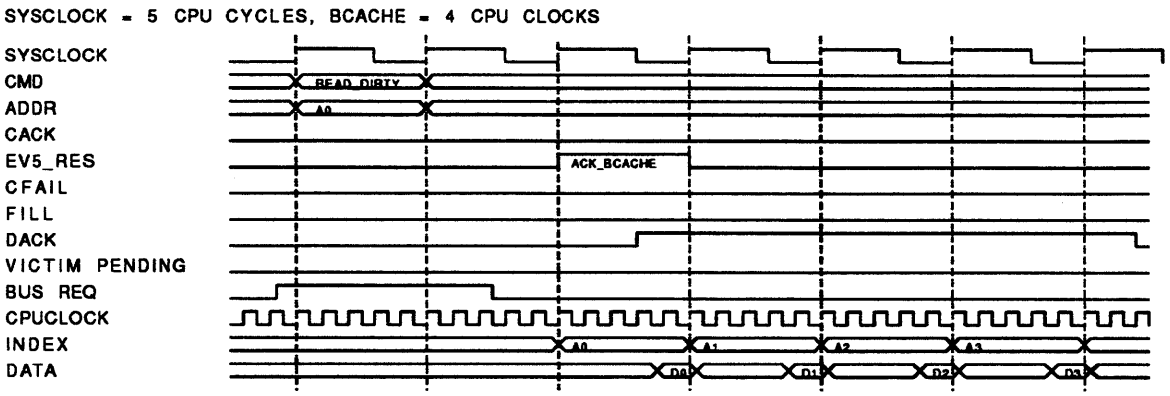


4.1.8.7 Read Dirty, and Read Dirty/INV

The READ DIRTY command is used to read modified data from the cache system. The block is also transitioned into the shared state. Figure 4–9 shows the timing of a READ_DIRTY transaction. The Scache will be probed and the data read if it is found. The state will also be set to shared. If the data is not found in the Scache, it is assumed to be in the Bcache. DECchip 21164-AA will start the read of the Bcache and write the tag to the shared state.

The READ DIRTY/INV command is identical to the READ DIRTY command except the block is transitioned to the invalid state instead of the shared state.

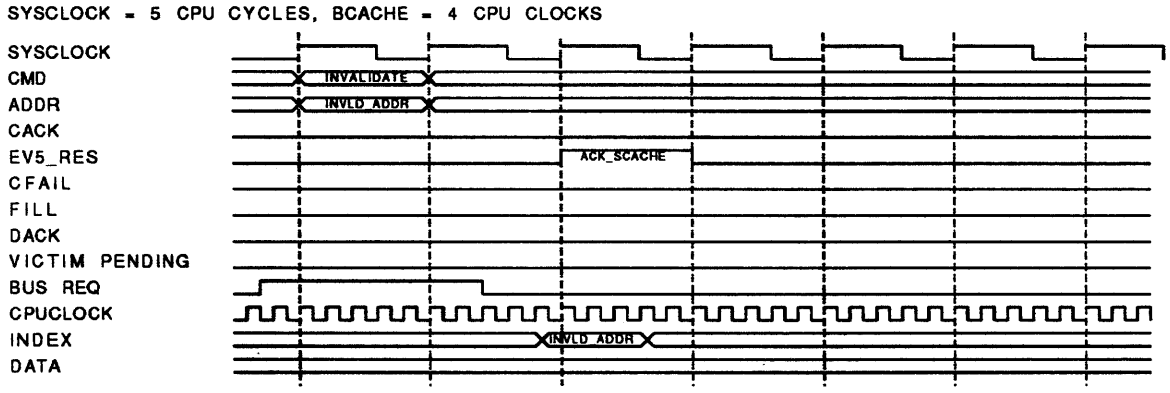
Figure 4–9: Read Dirty



4.1.8.8 Invalidate

The INVALIDATE command can be used to remove a block from the cache system. Unlike the FLUSH command, any modified data will not be read. The Scache will be probed and invalidated if the block is found. The Bcache will be invalidated without probing. Figure 4–10 shows the timing of an INVALIDATE transactions.

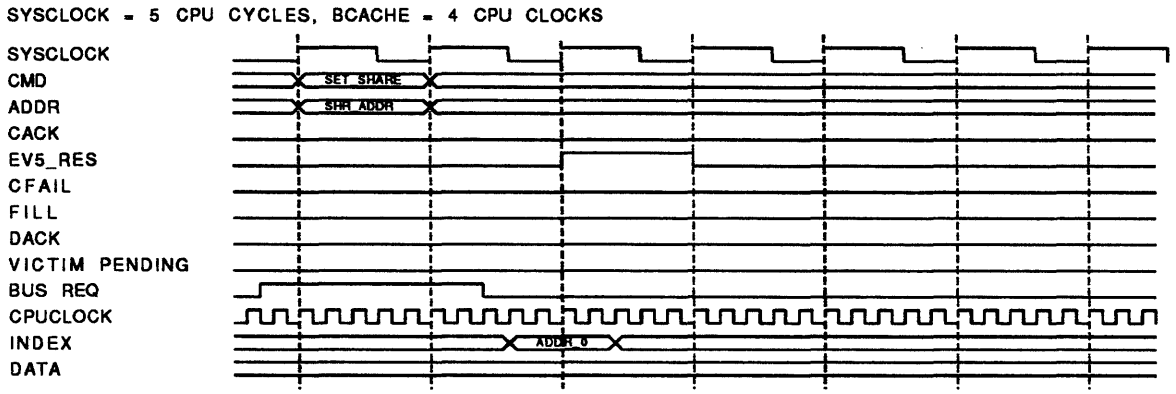
Figure 4–10: Invalidate



4.1.8.9 Set Shared

When DECchip 21164-AA revieves a SET_SHARED command, it will probe the Scache and change the state of the block to shared if it is found. DECchip 21164-AA will assume that the block is in the Bcache and write the state of the tag to shared, not-dirty. Figure 4–11 shows the timing of a SET_SHARED command.

Figure 4–11: Set Shared



4.1.8.10 Non-cached Reads

Reads to physical addresses that have bit 39 asserted will not be cached in the Dcache, Scache, or Bcache. They will be merged like any other read in the miss address file. To prevent several reads to non-cached memory from being merged into a single 32 byte bus request, software must insert MB instructions. The miss address file will merge as many Dstream reads together as it can and send the request to the BIU via the Scache. The BIU will not merge two 32 byte requests into a single 64 byte request. The BIU will request a READ MISS from the SYSTEM. DATA_VALID<3:0>_H will indicate which of the four quadwords are being requested by software. The SYSTEM should return the fill data to DECchip 21164-AA in the normal way. DECchip 21164-AA will not write the Dcache, the Scache, or the Bcache with the refill. The requested data will be written in the register file or Icache.

4.1.8.11 Non-cached Writes

Writes to physical addresses that have bit 39 asserted will not be written to any of the caches. They will be merged in the write buffer before being sent to the SYSTEM. If software does not want writes to merge it must insert MB or WMB instructions between them.

When the write buffer decides to dump data to non-cached memory the BIU will request a WRITE BLOCK. Each data cycle, DATA_VALID<3:0> will indicate which INT4s within the INT16 were really written.

4.1.8.12 Locks

The LDx_L instructions will be forced to miss in the Dcache. When the Scache is read, the Lock register in the BIU will be loaded with the physical address and the lock flag set. The BIU will send a LOCK command to the SYSTEM so it can load its lock register. The SYSTEM lock register will only be used if the locked block is displaced from the cache system. The lock flag will be cleared if any of the following things happen:

- Any write from the bus occurs to the locked block (FLUSH, INVALIDATE, or READ_DIRTY_INV).
- A STx_C by the processor.

The SYSTEM copy of the lock register is required on systems that have a duplicate tag store to filter write traffic. The direct mapped Icache, Dcache, and Bcache along with the sub-setting rules, branch prediction, and Istream prefetching can cause a lock to always fail because of constant Scache thrashing of the locked block. Each time a block is loaded into the Scache, the value of the lock register will be ANDed with the value of the SYSTEM_LOCK_FLAG signal. If the locked block is displaced from the cache system, DECchip 21164-AA will not see bus writes to the locked block, in this case the SYSTEM's copy of the lock register will correct the processor copy of the lock flag when the block is filled into the cache via the signal SYSTEM_LOCK_FLAG_H.

Systems that do not have a duplicate tag stores, and send all probe traffic to DECchip 21164-AA are not required to have a copy of the lock flag. They should wire the SYSTEM_LOCK_FLAG_H to TRUE.

When the STx_C is issued the Ibox will stop issuing memory type instructions. The store will update the Dcache in the normal way, and be placed in the write buffer by itself. It will not be merged with other pending writes. The write buffer will be flushed.

When the write buffer gets to a STx_C in cached memory, it will probe the Scache to check the block state. When the STx_C passes through the Scache, an invalidate will be sent to the Dcache. If the Lock flag is clear, the STx_C will fail. If the block is not-shared dirty, the write buffer will write the STx_C data into the Scache. Success will be written to the register file and the Ibox will begin issuing memory instructions again. If the block is in the shared state, the BIU will request a WRITE BLOCK LOCK. If the WRITE BLOCK LOCK is CACKed, the Scache will be written and the Ibox started as above. If the WRITE BLOCK LOCK is CFAILed, the STx_C will fail. No data will be written.

When the write buffer gets to a STx_C in non-cached memory it will probe the Scache to check the block state. It will miss. The state of the Lock flag will be ignored. The BIU will request a WRITE BLOCK LOCK. If the WRITE BLOCK LOCK is CACKed, the Ibox is started as above. If the WRITE BLOCK LOCK is CFAILed the STx_C will fail. No data will be written.

4.1.9 Clocks

4.1.9.1 CPU Clock

External logic will supply DECchip 21164-AA with a differential clock at twice the desired internal clock frequency via the CLK_IN_H and CLK_IN_L pins. DECchip 21164-AA divides this clock by two to generate the internal chip clock.

4.1.9.2 System Clock

The CPU clock is divided by a programmable value between 3 and 15 to generate a system clock, which is supplied to the external interface via the SYS_CLK_OUT1_H,L pins. Table 5–1 for the valid ratios of System clock to CPU clock.

SYS_CLK_OUT1 is delayed by a programmable number of CPU cycles between 0 and 7 to produce SYS_CLK_OUT2_H, L.

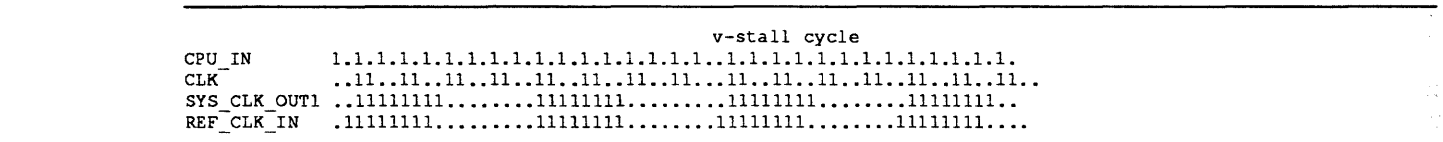
The output of the programmable divider is symmetric if the divisor is even, and asymmetric with SYS_CLK_OUT1_H and SYS_CLK_OUT2_H TRUE for one extra CPU cycle if the divisor is odd.

The false-to-true transition of the SYS_CLK_OUT1_H is the "Sysclock" used as a timing reference throughout the specification.

4.1.9.3 Reference Clock

The SYSTEM may supply a reference clock to which DECchip 21164-AA will synchronize SYS_CLK_OUT1_H. To do this the frequency of SYS_CLK_OUT1 must be ever so slightly higher than that of REF_CLK_IN. This will cause the rising edge of SYS_CLK_OUT1 to drift back towards the rising edge of REF_CLK_IN. DECchip 21164-AA will detect when the edges meet and stall the internal clock generator for one CLK_IN cycle. This will move the rising edge of SYS_CLK_OUT back in front of REF_CLK_IN. Figure 4–12 attempts to show this timing.

Figure 4–12: Reference Clock Timing



4.1.9.4 Sysclock to Bcache cycle time ratios

The Bcache cycle time may be faster, the same, or slower than the Sysclock.

Reads and writes that are private to DECchip 21164-AA and the Bcache may start on any CPU clock. There is no relation between the Sysclock and the Bcache accesses.

If the SYSTEM is involved in a Bcache transaction, each read or write will start on a Sysclock. It is up to the SYSTEM to control the rate of the Bcache transactions using the DACK wire.

The Bcache will be written during WRITE BLOCK, WRITE BLOCK LOCK, READ DIRTY, and READ DIRTY INV commands that source data from the Scache. The write of the first part of the block will start in the Sysclock that drove the command/response and address to the SYSTEM. The SYSTEM must allow enough time for the write to complete before asserting DACK. The next write will start on the Sysclock edge that DACK was asserted on.

When DECchip 21164-AA receives the fill indication for the SYSTEM it will start writing the Bcache in the N+2 Sysclock. At the end of the write time, DECchip 21164-AA will wait for the next Sysclock edge. If DACK is not asserted, the Bcache write will begin again at the same index. If DACK is asserted, the index will advance to the next part of the fill and the write will begin again. The SYSTEM must provide the data and DACK signal at the correct Sysclock edges to complete the fill correctly.

4.1.10 Tri-state Overlap

The ADDRESS/CMD bus and the DATA/TAG bus must be operated in a way that prevents more than one driver from driving the bus at a time. This section will describe the features in DECchip 21164-AA that might be used to prevent tri-state overlap.

The owner of each bus must drive the bus to some value each cycle.

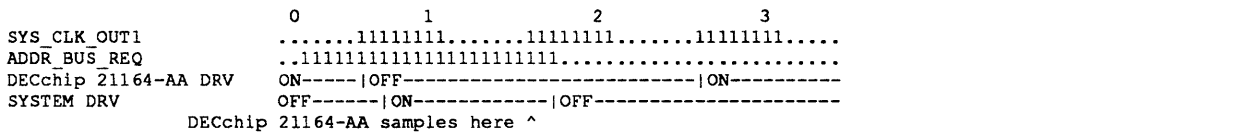
In general DECchip 21164-AA assumes that its drivers turn on and off very fast (0.5ns to 1ns range). SRAMs turn on and off slowly. System drivers fall someplace in the middle.

Figure 4–13 shows DECchip 21164-AA and the SYSTEM taking turns driving the CMD/ADDRESS bus. If ADDR_BUS_REQ is asserted at the end of a Sysclock 0, the next cycle on the CMD/ADDRESS bus belongs to the SYSTEM. DECchip 21164-AA will turn off it's drivers at the start of the Sysclock 1. The SYSTEM must turn on it's drivers during Sysclock 1, but must insure that the driver doesn't turn on before DECchip 21164-AA turns off. DECchip 21164-AA will sample the state of the CMD/ADDRESS bus at the end of Sysclock 1.

If ADDR_BUS_REQ remains asserted, the SYSTEM should continue to drive the CMS/ADDRESS bus.

To pass the bus back to DECchip 21164-AA, the SYSTEM should turn off its drivers during a Sysclock and de-assert ADDR_BUS_REQ. DECchip 21164-AA will not sample the state of the bus if ADDR_BUS_REQ is de-asserted. At the next Sysclock edge, DECchip 21164-AA will drive the bus.

Figure 4–13: Driving the CMD/ADDRESS Bus



The DATA bus can be driven by DECchip 21164-AA, the Bcache, or the SYSTEM.

For DECchip 21164-AA Bcache Writes followed by DECchip 21164-AA Bcache Reads, we assume that DECchip 21164-AA stops driving the DATA bus well in advance of the Bcache turning on.

For DECchip 21164-AA Bcache Reads followed by DECchip 21164-AA Bcache Writes, DECchip 21164-AA will insert a programmable number of CPU cycles between the read and the write. This will allow time for the Bcache drivers to turn off before turning on the DECchip 21164-AA data drivers. These rules apply to WRITE BLOCK, WRITE BLOCK LOCK, READ, READ DIRTY, and FLUSH commands as well.

DECchip 21164-AA will not prevent tri-state overlap at the start of a fill. The SYSTEM must assert IDLE_BC early enough to allow all the drivers to turn off before the SYSTEM turns on its drivers.

At the end of the Fill, DECchip 21164-AA will wait READ->WRITE programmable number of CPU cycles before starting a read or write. This time should allow the SYSTEM to turn off it's drivers. If this is not enough time, the system may assert DATA_BUS_REQ to gain addition cycles.

4.1.11 Restrictions

This section will document restrictions on the use of DECchip 21164-AA interface features.

4.1.11.1 Fills after other transactions

If the system is removing data from DECchip 21164-AA with any of the system commands, or if the system is removing a Bcache victim from the Bcache and it wants to follow any of these transactions with a fill, then the earliest assertion of the FILL signal is the Sysclock after the last DACK.

Fills followed by Fills is a special case. Fills can be pipelined back to back to use 100% of the data bus bandwidth.

This restriction may be lifted in the future.

4.1.11.2 Sending System commands

A SYSTEM can send up to TWO commands to DECchip 21164-AA. It must then wait for the assertion of the RES_H signal for the first command before it can send the third command.

4.1.11.3 CACK for WRITE BLOCK commands

When DECchip 21164-AA requests a WRITE BLOCK or WRITE BLOCK LOCK, the SYSTEM can DACK the data before asserting CACK. The SYSTEM must assert CACK no later than the last DACK.

4.1.11.4 No Bcache Systems

SYSTEMs without a Bcache must have a block size of 64 bytes and all three sets in the Scache must be enabled.

4.1.11.5 Scache duplicate tag store

SYSTEMs without a Bcache that do have an Scache duplicate tag store are also required to maintain tags for the two blocks in the DECchip 21164-AA Scache victim buffer.

NOTE

FETCH and FETCH_M commands will no longer be auto acked by DECchip 21164-AA. They will always be driven to the SYSTEM for acknowledgement.

SET DIRTY, LOCK, and MB commands have been merged in to a single command group in the BC_CONTROL<EI_OPT_CMD> ipr.

4.1.12 ECC/Parity

The chip will support INT8 ECC for the external Bcache and memory system. ECC will be provided by the CPU for each INT8 that is written into the Bcache. Fill data read from the Bcache and memory will be checked by hardware. Uncorrected data will be sent to the Dcache, and register files. Single bit errors will be corrected by hardware. The Scache and Icache will be filled with corrected data. Double bit errors will be detected. If the SYSTEM has indicated that the data should not be checked, no checking or correcting will be performed.

Each data bus cycle will deliver one INT16 worth of data. ECC is calculated as ECC(data<63:0>) and ECC(data<127:64>). This allows ECC to be calculated on each side of the chip. Figure 4–14 shows the code. Two IDT49C460 or AMD29C660 parts can be cascaded to produce this ECC code. A single IDT49C466 will also support this ECC code.

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

If the DECchip 21164-AA is in parity mode, it will generate byte parity and place it on the DATA_CHECK_H<15:0> for writes. Parity will be checked for reads. Parity for data<7:0> will be driven on DATA_CHECK_H<0> and so on.

Figure 4–14: ECC code

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

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

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

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

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

For x4 RAMs, Dave Hartwell has provide the following bit arrangement that will detect nibble errors.

Figure 4–15: x4 bit arrangement

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

4.2 Revision History

Table 4–9: Revision History

Who	When	Rev	Description of change
Pete Bannon	12/16/91	0.8	DRAFT 0.8 text
Pete Bannon	12/31/91	0.9	DRAFT 0.9 text
Pete Bannon	3/ 1/92	1.0	FILL ERROR, new non-cached read
Pete Bannon	3/27/92	1.2	New WS focus interface
Pete Bannon	3/27/92	1.3	New victim sequence
Pete Bannon	4/21/92	1.4	New ECC code
Pete Bannon	11/30/92	1.5	general update

Chapter 5

Reset and Initialization

5.1 SYS_RESET_L and DC_OK_H

The DECchip 21164-AA reset process starting from a powered off state uses two input signals, SYS_RESET_L and DC_OK_H. Until power has reached the proper operating point, DC_OK_H must be deasserted and SYS_RESET_L must be asserted. After power has reached the proper operating point, DC_OK_H is asserted. After that, SYS_RESET_L is deasserted.

From a powered on state, the reset sequence begins with SYS_RESET_L assertion. In any case, after SYS_RESET_L is deasserted, DECchip 21164-AA begins a sequence of operations: Icache BiSt, followed by an optional automatic Icache initialization via an external serial ROM interface, and finally dispatching to the RESET PALcode trap entry point.

If DC_OK_H is not asserted, SYS_RESET_L is forced asserted internally.

SYS_RESET_L forces the CPU into a known state. Chapter 3 gives the reset state of each IPR and Section 9.1 gives the reset state of the pins.

While DC_OK_H is deasserted, DECchip 21164-AA provides its own internal clock source from an on-chip ring oscillator. When DC_OK_H is asserted, the DECchip 21164-AA clock source is the differential clock input pins, CLK_IN_H and CLK_IN_L.

SYS_RESET_L must remain asserted while DC_OK_H is deasserted and for a period of time after DC_OK_H assertion which is at least TBD internal CPU cycles in length and at least TBD Sysclock cycles in length. After that, SYS_RESET_L is deasserted. SYS_RESET_L deassertion generally should be synchronous with respect to Sysclock.

ISSUE

Does DECchip 21164-AA have to support asynchronous deassertion of SYS_RESET_L?

When DECchip 21164-AA is running off the internal ring oscillator, the internal clock frequency is in the range TBD. Also the Sysclock divisor ratio is forced to TBD and the SYS_CLK_OUT2_x delay is forced to TBD. After DC_OK_H is asserted, the Sysclock divisor and SYS_CLK_OUT2_x delay are determined by input pins while SYS_RESET_L remains asserted. See Section 5.2.

5.1.1 Power Up Requirements

The DECchip 21164-AA chip uses a 3.3V power supply. This 3.3V power supply must be stable before any input or bidirectional pin rises above 4V.

The VREF_H input pin must have reached the correct stable operating point before DC_OK_H is asserted. See Chapter 7.

5.1.2 Pin State with DC_OK_H Not Asserted

While DC_OK_H is not asserted (and SYS_RESET_L is asserted), every output and bidirectional DECchip 21164-AA pin is tristated and pulled weakly to ground by a small pull-down transistor.

5.2 Sysclock Ratio and Delay

While in reset, DECchip 21164-AA reads Sysclock configuration parameters from the interrupt pins. Table 5–1 shows how the Sysclock divisor is determined and Table 5–2 shows how the SYS_CLK_OUT2_x delay is determined. These inputs should be driven with the correct configuration whenever SYS_RESET_L is asserted. When these inputs change while SYS_RESET_L is asserted, it takes TBD internal CPU cycles before the new Sysclock behavior is correct.

Table 5–1: System Clock Divisor

IRQ_H<3>	IRQ_H<2>	IRQ_H<1>	IRQ_H<0>	Ratio
L	L	H	H	3
L	H	L	L	4
L	H	L	H	5
L	H	H	L	6
L	H	H	H	7
H	L	L	L	8
H	L	L	H	9
H	L	H	L	10
H	H	H	H	15
all other values				unspecified effect

Table 5–2: System Clock Delay

SYS_MCH_CHK_IRQ_H	PWR_FAIL_IRQ_H	MCH_HLT_IRQ_H	Delay
L	L	L	0
L	L	H	1
L	H	L	2
L	H	H	3
H	L	L	4
H	L	H	5
H	H	L	6
H	H	H	7

5.3 BiSt

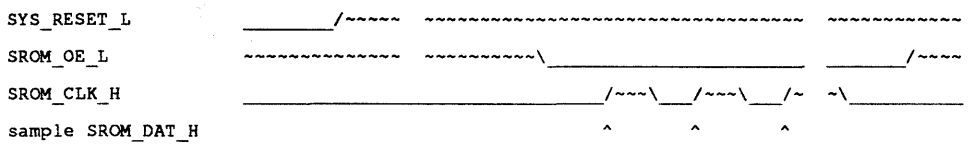
Normally upon deassertion of SYS_RESET_L, DECchip 21164-AA automatically executes Icache BiSt (Built in Self-test). If PORT_MODE_H<1> is asserted, the test port is in debug test interface mode and BiSt is bypassed. Otherwise, the Icache is automatically tested and the result is made available in ICSR and on TEST_STATUS_H<0>. Internally, the CPU chip reset continues to be asserted throughout the BiSt test process.

5.4 Serial ROM

After Icache BiSt completes, an optional serial ROM Icache load sequence begins. If SROM_PRESENT_L was not asserted when SYS_RESET_L transitioned to deasserted, the serial ROM load process is skipped, internal CPU reset is deasserted, and PALcode execution begins at the RESET trap entry point. If SROM_PRESENT_L was asserted when SYS_RESET_L transitioned to deasserted, the serial ROM load sequence is completed prior to deassertion of internal CPU reset and PALcode execution at the RESET trap entry point.

Figure 5–1 gives a timing diagram of a serial ROM load sequence. Chapter 11 describes the format of the Icache data. Every data and tag bit in the Icache is loaded by this sequence.

Figure 5–1: Serial ROM Load Timing



5.5 Cache Initialization

Regardless of whether Icache BiSt is executed, the Icache is flushed during the reset sequence prior to serial ROM load. If serial ROM load is bypassed, the Icache is initially in the flushed state.

The Scache is flushed and enabled by internal reset. This is required if serial ROM load is bypassed. The initial Istream reference after reset is location 0. Since that is a cacheable-space reference, it will probe the Scache.

The Bcache is disabled by reset.

The Dcache is disabled by reset. It is not initialized or flushed by reset.

5.6 BIU initialization

After reset, the Cbox is in the default configuration dictated by the reset state of the IPR bits which select the configuration options. (Note that the Bcache configuration registers are not initialized by reset.) The Cbox response to system commands and internally generated memory accesses will be determined by this default configuration. Systems should be compatible with ths default configuration or arrange to change it before initiating any accesses to cacheable space. Since the initial PALcode trap entry point is in cacheable space, system environmennts which are not compatible with the default configuration must utilize the serial ROM Icache load feature to initially load and execute a PALcode program which will configure Cbox IPRs as needed.

5.7 Unitialized state

A number of IPR bits are not initialized by reset. These are error reporting registers and some other IPR states. These must be initialized by initialization PALcode.

5.8 Timeout Reset

The Ibox contains a timeout timer which times out when a very long period of time passes with not one instruction completing. When this timeout occurs, an internal reset event occurs which clears sufficient internal state to allow the CPU to begin exeuting again. Registers, IPRs, and Caches are not affected. Dispatch to the PALcode MCHK trap entry point occurs immediately.

5.9 Clock Reset

A TBD method will exist which allow a chip tester to initialize the Sysclock divider logic. This allows for deterministic operation during chip test. Due to the size of internal logic propagation delays as compared to the normal speed of the internal CPU clock, it will be necessary to run the internal CPU clock at a low speed while initializing the Sysclock divider.

5.10 IEEE 1149.1 Test Port Reset

TRST_L must be asserted whenever SYS_RESET_L is asserted or DC_OK_H is deasserted.

Continuous TRST_L assertion during normal operation can be used to prevent the IEEE 1149.1 Test Port from affecting DECchip 21164-AA operation.

5.11 Revision History

Table 5–3: Revision History

Who	When	Description of change
JHE	1-March-1992	Brief statement of plan.
JHE	30-November-1992	Update

Chapter 6

Error Handling

6.1 Overview

This is an overview of DECchip 21164-AA's error handling strategy. Each internal cache (Icache, Dcache, and Scache) implements parity protection for tag and data. ECC protection is implemented for memory and Bcache data. (The implementation provides detection of all double-bit errors and correction of all single bit errors.) Correctable Istream and Dstream ECC errors are corrected in hardware without PALcode intervention. Bcache tags are parity protected. The Ibox implements logic which detects when no progress has been made for a very long time (a TBD number of CPU cycles of issue stall or infinitely repeated traps) and forces a machine check trap.

PALcode handles all error traps (machine checks and correctable error interrupts). At the time the error is handled, PALcode builds a logout frame pointed to by the HWRPB.

Where possible, the address of affected data is reported to the operating system. Most of the Istream errors are retryable by the operating system. In some other cases, the system may be able to recover from an error by terminating all processes which had access to the affected memory location.

6.2 Error Flows

6.2.1 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 set.
- Retryable.
- 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.

6.2.2 Scache data parity error - Istream

- Machine check occurs before the instruction causing the parity error is executed.
- Bad data may be written to the Icache or Icache Refill Buffer and validated.
- Retryable if there are no multiple errors.
- Recommendation: Flush the Icache to remove bad data. The Icache Refill Buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- SC_STAT: SC_DPERR<7:0> set , SC_SCND_ERR set if there are multiple errors.
- SC_STAT: CBOX_CMD is IRD
- SC_ADDR: Contains the address of the 32B block containing the error. (Note: bit4 indicates which octaword was accessed first, but the error may be in either octaword).
- Note: If the Istream parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.
- Recommendation: On data parity errors, it may be feasible for the operating system to "flush" the block of data out of the Scache by requesting a block of data with the same Bcache index, but a different tag. This may not be feasible on tag parity errors, since the tag address is suspect. If the requested block is loaded with no problems, then the "bad data" has been replaced. If the "bad data" is marked dirty, then when the new data tries to replace the old data, another parity error may result during the writeback (this is a reason not to attempt this in palcode, since a MCHK from palcode is always fatal).

6.2.3 Scache tag parity error - Istream

- Machine check occurs before the instruction causing the parity error is executed.
- Bad data may be written to the Icache or Icache Refill Buffer and validated.
- Not retryable. Probably won't be able to recover by deleting a single process because the exact address is unknown.
- Recommendation: Flush the Icache to remove bad data. The Icache Refill Buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- SC_STAT: SC_TPERR<2:0> set , SC_SCND_ERR set if there are multiple errors.
- SC_STAT: CBOX_CMD is IRD
- SC_ADDR: Contains the address of the 32B block containing the error. (Note: bit4 indicates which octaword was accessed first, but the error may be in either octaword).
- Note: If the Istream parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.

6.2.4 Scache data parity error - Dstream read/write, READ_DIRTY

- Machine check occurs. Machine state may have changed.
- Not retryable, but may only need to delete the process if data is confined to a single process and no second error occurred.
- SC_STAT: SC_DPERR<7:0> set , SC_SCND_ERR set if there are multiple errors
- SC_STAT: CBOX_CMD is DRD, DWRITE, or READ_DIRTY
- SC_ADDR: Contains the address of the 32B block containing the error. (Note: bit4 indicates which octaword was accessed first, but the error may be in either octaword).

6.2.5 Scache tag parity error - Dstream or system commands

- Machine check occurs. Machine state may have changed.
- Not retryable. Probably won't be able to recover by deleting a single process because the exact address is unknown.
- SC_STAT: SC_TPERR<7:0> set , SC_SCND_ERR set if there are multiple errors
- SC_STAT: CBOX_CMD is DRD, DWRITE, READ_DIRTY, SET_SHARED, or INVAL
- SC_ADDR: records physical address bits [39:4] of location with error.

6.2.6 Dcache data parity error

- Machine check occurs. Machine state may have changed.
- Not retryable, 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 set. LOCK set. SEO set if there are multiple errors. Note: For multiple parity errors in the same cycle, the SEO bit will not be 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 info about instruction causing parity error. Note: Fault information on other instruction in same cycle may be lost.

6.2.7 Dcache tag parity error

- Machine check occurs. Machine state may have changed.
- DCPERR_STAT: TP0 or TP1 set. LOCK set. SEO set if there are multiple errors. Note: For multiple parity errors in the same cycle, the SEO bit will not be set, but more than one error bit will be set.
- VA: Contains the virtual address of the Dcache block (hexaword) with the error.
- MM_STAT locked. Contents contain info about instruction causing parity error. WR bit set if error occurred on a store. Note: Fault information on other instruction in same cycle may be lost.
- Probably won't be able to recover by deleting a single process, because exact address is unknown, and a load may have falsely hit.

6.2.8 Istream uncorrectable ECC or data parity errors (Bcache or memory)

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache Refill Buffer and validated.
- Retryable if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache Refill Buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- EI_STAT: UNC_ECC_ERR set, SEO_HRD_ERR set if there are multiple errors.
- EI_STAT: EI_ES set if source of fill data is memory/system, clear if Bcache.
- EI_STAT: FIL_IRD is set
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- FILL_SYN: contains syndrome bits associated with the failing octaword. This register contains byte parity error status if in parity mode.
- BC_TAG_ADDR: holds results of external cache tag probe if external cache was enabled for this transaction.
- Note: If the Istream ECC or parity error occurs early in the PALcode routine at the machine check entry point, an infinite loop may result.
- Recommendation: On data ECC/parity errors, it may be feasible for the operating system to "flush" the block of data out of the Bcache by requesting a block of data with the same Bcache index, but a different tag. If the requested block is loaded with no problems, then the "bad data" has been replaced. If the "bad data" is marked dirty, then when the new data tries to replace the old data, another ECC/parity error may result during the writeback (this is a reason not to attempt this in palcode, since a MCHK from palcode is always fatal).

6.2.9 Dstream uncorrectable ECC or data parity errors (Bcache or memory)

- Machine check occurs. Machine state may have changed.
- Not retryable, but may only need to delete the process if data is confined to a single process and no second error occurred.
- EI_STAT: UNC_ECC_ERR set, SEO_HRD_ERR set if there are multiple errors.
- EI_STAT: EI_ES set if source of fill data is memory/system, clear if Bcache.
- EI_STAT: FIL_IRD is clear
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- FILL_SYN: contains syndrome bits associated with the failing octaword. This register contains byte parity error status if in parity mode.
- BC_TAG_ADDR: holds results of external cache tag probe if external cache was enabled for this transaction.

6.2.10 Bcache tag parity errors - Istream

- Machine check occurs before the instruction causing the error is executed.
- Bad data may be written to the Icache or Icache Refill Buffer and validated.
- Retryable if there are no multiple errors.
- Must flush Icache to remove bad data. The Icache Refill Buffer may be flushed by executing enough instructions to fill the refill buffer with new data (32 instructions). Then flush the Icache again.
- EI_STAT: BC_TPERR or BC_TC_PERR set, SEO_HRD_ERR set if there are multiple errors.
- EI_STAT: EI_ES clear
- EI_STAT: FIL_IRD is set
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- BC_TAG_ADDR: holds results of external cache tag probe.
- NOTE: 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 non-fatal 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.

6.2.11 Bcache tag parity errors - Dstream

- Machine check occurs. Machine state may have changed.
- Not retryable, but may only need to delete the process if data is confined to a single process and no second error occurred. Bcache hit is determined based on the tag alone, not the parity bit. The victim is processed according to the status bits in the tag, ignoring the control field parity. PALcode can distinguish fatal from non-fatal occurrences by checking for the case in which a potentially dirty block is replaced without the victim being properly written back and the case of false hit when the tag parity is incorrect.
- EI_STAT: BC_TPERR or BC_TC_PERR set, SEO_HRD_ERR set if there are multiple errors.
- EI_STAT: EI_ES clear
- EI_STAT: FIL_IRD is clear
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- BC_TAG_ADDR: holds results of external cache tag probe.

6.2.12 System command/address parity error

- Machine check occurs. Machine state may have changed.
- EI_STAT: EI_PAR_ERR set, SEO_HRD_ERR set if there are multiple errors.
- EI_STAT: EI_ES set
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- BC_TAG_ADDR: holds results of external cache tag probe if external cache was enabled for this transaction.

- When DECchip 21164-AA detects a command or address parity error, the command is unconditionally NOACKed.

6.2.13 System reads of the Bcache

- DECchip 21164-AA does not check the ECC on outgoing Bcache data. If it is bad, the receiving processor will detect it.

6.2.14 Istream or Dstream correctable ECC error (Bcache or memory)

- DECchip 21164-AA hardware corrects the data before filling the Scache and Icache. The Dcache is completely invalidated. The data in the Bcache contains the ECC error, but is scrubbed by palcode in the correctable error interrupt routine. (Using LDxL, STxC. If the STxC fails, the location can be assumed to be scrubbed.)
- A separately maskable correctable error interrupt occurs at IPL 31 (same as machine check). (Masked by clearing ICSR<crde>.)
- ISR: CRD set.
- EI_STAT: COR_ECC_ERR set.
- EI_STAT: FIL_IRD set if Istream, clear if Dstream.
- EI_STAT: EI_ES clear if source of error is Bcache, set otherwise.
- EI_ADDR: contains the physical address bits 39:4 of the octaword associated with the error.
- FILL_SYN: contains syndrome bits associated with the octaword containing the ECC error.
- BC_TAG_ADDR: Unpredictable (not loaded on correctable errors)
- Note: There will be performance degradation in systems when extremely high rates of correctable ECC errors are present due to the internal handling of this error (the implementation utilizes a replay trap and automatic Dcache flush to prevent use of the incorrect data).

6.2.15 Fill Timeout (FILL_ERROR_H)

- For systems in which fill timeout can occur, the system environment should detect fill timeout and cleanly terminate the reference to DECchip 21164-AA. If the system environment expects fill timeouts to occur, it should detect them. If it does not expect them (as might be true in small systems with fixed memory access timing), it is likely that the internal Ibox timeout will eventually detect a stall if a fill fails to occur. To properly terminate a fill in an error case, the FILL_ERROR_H pin is asserted for one cycle and the normal fill sequence involving the FILL_H, FILL_ID_H, and DACK pins is generated by the system environment.
- 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 reads of the appropriate status register(s) in the MCHK palcode.

6.2.16 System machine check

- DECchip 21164-AA has a maskable machine check interrupt input pin. It is used by system environments to signal fatal errors which are not directly connected to a read access from DECchip 21164-AA. It is masked at IPL 31 and anytime DECchip 21164-AA is in PALmode.
- ISR: MCK set.

6.2.17 Ibox timeout

- When the Ibox detects a timeout, it causes a PALcode trap to the MCHK entry point.
- Simultaneously, a partial internal reset occurs: most state except IPR state is reset. This should not be depended on by systems in which fill timeouts occur in typical use (e.g., 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 set.

6.2.18 CFAIL_H and not CACK_H

- Assertion of CFAIL_H in a sysclock cycle in which CACK_H is not asserted causes DECchip 21164-AA to immediately execute a partial internal reset.
- PALcode trap to the MCHK entry point.
- Simultaneously, a partial internal reset occurs: most state except IPR state is reset.
- ICPERR_STAT: TMR set.
- This can be used to restore DECchip 21164-AA and the external environment to a consistent state after the external environment detects a command or address parity error.
- Note: There is no internal status saved to differentiate the CFAIL_H/no CACK_H case from the timeout reset case. If necessary, systems must save this status, and include reads of the appropriate status register(s) in the MCHK palcode.

6.3 MCHK flow

- 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 (reason_for_halt = mchk_from_pal) (???need to unlock regs???).
- Read MCES.
- If MCES<mchk> set, THEN HALT (reason_for_halt = dbl_mchk) (???need to unlock regs???).
- Set MCES<mchk>.
- Issue MB to clear out Mbox/Cbox before reading Cbox registers or issuing DC_FLUSH.
- Flush Dcache to remove bad data on Dstream errors.

- Read ICSR.
- Read ICPERR_STAT.
- Read DCPERR_STAT.
- Read SC_ADDR.
- Use register dependencies or MB to ensure read of SC_ADDR finishes before subsequent read of SC_STAT.
- Read SC_STAT (unlocks sc_addr).
- Read EI_ADDR, BC_TAG_ADDR, FILL_SYN.
- Use register dependencies or MB to ensure reads of EI_ADDR, BC_TAG_ADDR, FILL_SYN finish before subsequent read of EI_STAT.
- Read EI_STAT and save (unlocks EI_ADDR,BC_TAG_ADDR,FILL_SYN).
- Read EI_STAT again to be sure it is unlocked, throw away result.
- Check for non-retryable cases. If any one of the following are true, then skip retry:
  - EI_STAT<tperr>
  - EI_STAT<tc_perr>
  - EI_STAT<ei_par_err>
  - EI_STAT<seo_hrd_err>
  - EI_STAT<unc_ecc_err> AND NOT EI_STAT<fil_ird>
  - DCPERR_STAT<lock>
  - SC_STAT<sc_scnd_err>
  - SC_STAT<sc_tperr>
  - NOT (SC_STAT<cmd> == IRD) AND SC_STAT<sc_dperr>
  - ICPERR_STAT<tmr>
  - ISR<mck>
- If none of the above are true, then either we have a retryable iread, or the source of the MCHK is a FILL_ERROR_H. Add code for query of system status.
- Set the retry flag in the logout frame if any one or several of the following are true (and none of the above conditions were true):
  - EI_STAT<unc_ecc_err> AND EI_STAT<fil_ird>
  - SC_STAT<sc_dperr> AND (SC_STAT<cmd> == IRD)
  - ICPERR_STAT<tpe>
  - ICPERR_STAT<dpe>
- Build MCHK logout frame, including the following IPRs:
  - PT0-23
  - EXC_ADDR
  - EXC_SUM
  - EXC_MASK
  - PAL_BASE
  - ISR
  - ICSR
  - ICPERR_STAT

    - DCPERR_STAT
    - MM_STAT
    - VA (read unlocks VA and MM_STAT)
    - SC_ADDR from register file
    - SC_STAT from register file
    - BC_TAG_ADDR from register file
    - EI_ADDR from register file
    - FILL_SYN from register file
    - EI_STAT from register file
    - LD_LOCK
- unlock the following iprs:
    - ICPERR_STAT (write 0x1800)
    - DCPERR_STAT (write 0x03)
    - VA, SC_STAT, and EI_STAT are already unlocked.
- Check for arithmetic exceptions:
    - Read EXC_SUM.
    - Check for arithmetic errors
    - If arithmetic error found, go to ARITH_MCHK routine (which builds stack frame and returns back here)
    - Clear EXC_SUM (unlocks EXC_MASK)
- Report the Processor Uncorrectable MCHK according to Operating System specific requirements.

6.4 Processor Correctable Error Interrupt Flow (IPL 31)

- Got here through interrupt routine because ISR<crd> bit set.
- Read EI_ADDR, FILL_SYN
- Use register dependences or MB to ensure reads of EI_ADDR, FILL_SYN finish before subsequent read of EI_STAT.
- Read EI_STAT. (unlocks EI_STAT, EI_ADDR, FILL_SYN)
- Read MCES. If MCES<pce> is set, set SECOND ERROR FLAG in the logout frame, skip frame building and go to scrub memory location routine.
- Building Logout Frame:
    - Write frame size, offsets, etc.
    - Set retry flag.
    - Set error code to be processor correctable error and write.
    - Write EI_ADDR, FILL_SYN, EI_STAT (BC_TAG_ADDR is unpredictable, so don't save it).
- Scrub the memory location by using LDQ_L/STQ_C to one of the quadwords in each octaword of the Bcache block whose address is reported in EI_ADDR. No need to scrub IO space addresses as these are non-cacheable.

- ACK the CRD Interrupt by writing a "0" to HWINT_CLR<crdc>
- If MCES<dpc> is set (logging disabled), dismiss the interrupt.
- Set MCES<pce>.
- No need to unlock any registers because conditions that would cause a lock would also cause a MCHK. VA will not be locked because DTB_MISS and FAULT pal routines will not ever be interrupted.
- Report the Processor Correctable MCHK according to Operating System specific requirements.
- NOTE: Only read EI_STAT once in the CRD flow, and then only if you KNOW there is a correctable error (ISR<crd> set). If an uncorrectable error were to occur just after a second read from EI_STAT were issued, then there could be a race between the unlocking of the register and the loading of the new error status, potentially resulting in the loss of the error status.

6.5 MCK_INTERRUPT Flow

- Got here through interrupt routine because ISR<MCK> bit set.
- Read MCES
- If MCES<mchk> set, THEN HALT (reason_for_halt = dbl_mchk) (???need to unlock regs???)
- Set MCES<mchk>
- Follow MCHK frame building flow, but do not set retry flag.
- Report the System Uncorrectable MCHK according to Operating System specific requirements.

6.6 System Correctable Error Interrupt Flow (IPL 20)

- The System Correctable Error Interrupt is entirely system specific.

6.7 Revision History

Table 6–1: Revision History

Who	When	Description of change
JHE	1-March-1992	Brief strategy statement
JEM	13-Nov-1992	Overview for new release.
JHE	19-Dec-1992	Edits for new release.
JEM	6-May-1993	Added more detailed flows.
JEM	2-Sep-1993	icperr_stat<tmr> now set on cfail/no cack errors.

Chapter 7

DC Characteristics

7.1 Overview

DECchip 21164-AA is capable of running in a CMOS/TTL environment or an ECL environment. The chips will be tested and characterized in a CMOS environment. The specifications below assume a CMOS/TTL environment. Differences for an ECL environment are noted in Section 7.2.

7.1.1 Power Supply

In CMOS mode the VSS pins are connected to 0.0V, and the VDD pins are connected to 3.3V, +/- 5%.

The VREF_H analog input should be connected to a 1.4V +/-10% reference supply.

7.1.2 Input Clocks

CLK_IN (_H,_L) is expected to be a differential signal generated from an ECL oscillator circuit, although non-ECL circuits may also be used. It may be AC coupled, with a nominal DC bias of VDD/2 set by a high-impedence (i.e. >1K) resistive network on chip. It need not be AC coupled if VDD is used as the VCC supply to the ECL oscillator. See the AC Characteristics chapter for more detail.

7.1.3 Signal pins

Input pins are ordinary CMOS inputs with standard TTL levels, see Table 7–1. Once power has been applied and VREF_H has met its hold time, the majority of input pins can be driven by 5.0V (nominal) signals without harming DECchip 21164-AA. There are some signals that are sampled before VREF_H is stable, and these signals can not be driven above the power supply. These signals are:

- DC_OK_H
- ECL_OUT_H
- TRST_L
- TDI_H
- TDO_H
- TMS_H
- TCK_H

Output pins are ordinary 3.3V CMOS outputs. Although output signals are rail-to-rail, timing is specified to standard TTL levels, see Table 7–1.

Bidirectional pins are ordinary 3.3V CMOS bidirectional. On input, they act like input pins. On output, they drive like output pins.

Once power has been applied, input (except noted above) and bidirectional pins can be driven to a maximum DC voltage of 5.5V without harming DECchip 21164-AA (it is not necessary to use static RAMS with 3.3V outputs).

Table 7–1: CMOS DC Characteristics

Parameter		Requirements			
Symbol	**Description**	**Min**	**Max**	**Units**	**Test Conditions**
TTL Inputs/Outputs					
Vih	High level input voltage	2.0		V	
Vil	Low level input voltage		0.8	V	
Voh	High level output voltage	2.4		V	Ioh = -100uA
Vol	Low level output voltage		0.4	V	Iol = 3.2mA
Power/Leakage					
Icin	Clock input Leakage	-50	50	uA	-0.5<Vin<5.5V
Iil	Input leakage current	10	10	uA	0<Vin<Vdd V
Iol	Output leakage current (three-state)	-10	-10	uA	

7.2 ECL 100K Mode

In ECL 100K mode a combination of on-chip and off-chip circuits provide ECL 100K compatible interfaces.

7.2.1 Power Supply

In ECL 100K mode the VDD pins are connected to 0.0V, and the VSS pins are connected to -3.3V, +/- 5%.

7.2.2 Reference Supply

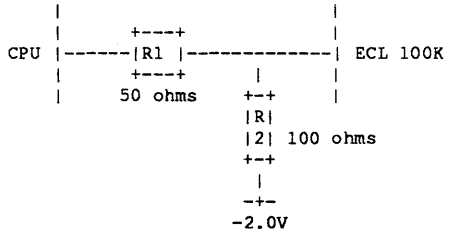
In ECL 100K mode the VREF_H input is connected to a reference supply at VDD-1.3V. The best way to generate the reference supply is to use the VBB output provided by several chips, such as the ECLinPS MC100E111.

7.2.3 Inputs

In ECL 100K mode inputs appear to be ordinary ECL 100K inputs, with the exception that they lack the pull down resistor that is normally present in ECL 100K circuits.

7.2.4 Outputs

In ECL 100K mode external resistors create the correct ECL 100K levels. The following stylized circuit is used.



7.2.5 Bidirectionals

In ECL 100K mode the bidirectional pins should be converted into unidirectional input and output busses as close to DECchip 21164-AA as possible. The DECchip 21164-AA chip bidirectional bus is buffered and driven onto the system output bus. The system input bus is driven onto DECchip 21164-AA's bidirectional bus using cut-off drivers controlled by the CPU's output enables.

The same resistor network used on output pins is used on bidirectional pins.

7.3 Power Dissipation

Table 7–2 Shows the estimated power maximum consumption at 286Mhz. Power consumption scales linearly with frequency in the frequency range 225Mhx to 312Mhz.

Table 7–2: DECchip 21164-AA Estimated Power Dissipation @Vdd=3.45V

Speed	Min	Typ	Max	Units
286 Mhz	TBD	TBD	60	Watts

7.4 Revision History

Table 7–3: Revision History

Who	When	Description of change
Pete Bannon	December 16, 1991	Include EV4 text
JHE	December 16, 1992	Updates

Chapter 8

AC Characteristics

8.1 Overview

This chapter describes the AC timing specifications for the DECchip 21164-AA running in a CMOS or TTL environment. Timing parameters are given for the nominal speed DECchip 21164-AA operating at an internal frequency of 294 MHz (3.4 ns).

8.2 Clocking Scheme

The input clock pins OSC_CLK_IN_H,_L run at 2x the internal frequency of the time base for DECchip 21164-AA. Input clocks are divided-by-two on-chip to generate a 50% duty cycle clock for internal distribution.

System designers have a choice of two system clocking schemes to run DECchip 21164-AA synchronous to the system.

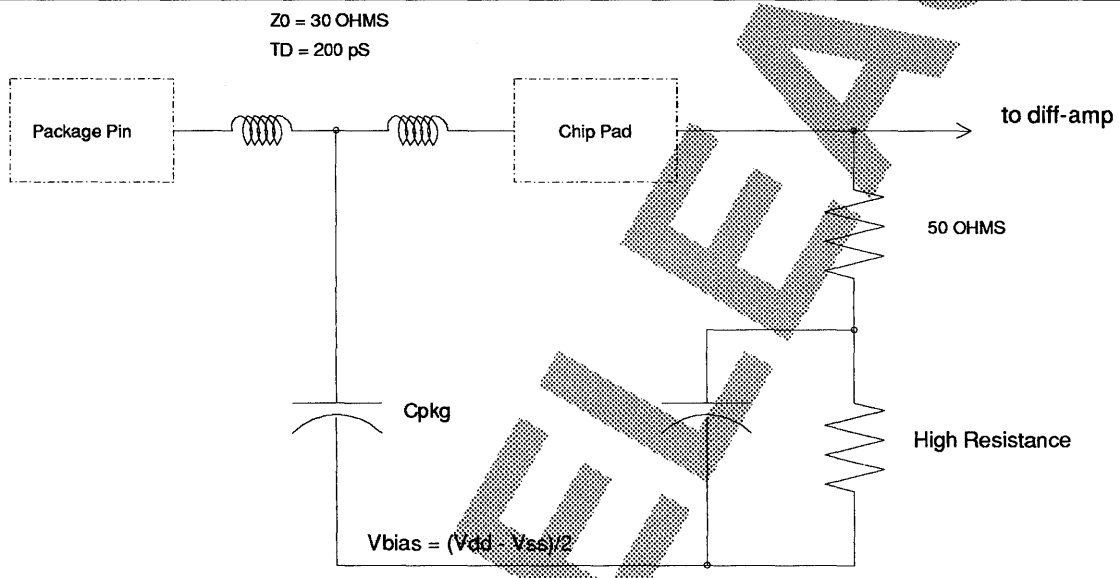
1. DECchip 21164-AA will generate and drive out a System clock, SYS_CLK_OUT1_H,_L. It runs synchronous to the internal clock at a selected ratio of the internal clock frequency.
2. DECchip 21164-AA will synchronize to a REF_CLK_IN_H supplied by the system. System clock REF_CLK_IN is running at a selected ratio of DECchip 21164-AA's internal clock frequency. Reference clock is synchronized to the internal clock by an on-chip Digital Phase Locked Loop.

See Reset and Initialization chapter for the valid ratios of DECchip 21164-AA internal clock to system clock.

8.2.1 Input Clocks

The input clocks OSC_CLK_IN_H,_L provide the time-base for DECchip 21164-AA when DC_OK_H is asserted. The terminations on these signals are designed to be compatible with system oscillators of arbitrary DC bias. The schematic equivalent is shown below:

Figure 8–1: OSC_CLK_IN_H,_L Terminations



This is designed to approximate a 50ohm termination for the purpose of impedance matching for those systems (if any) which drive input clocks across long traces. Furthermore, the high impedance bias driver allows a clock source of arbitrary DC bias to be AC coupled to DECchip 21164-AA. The peak-to-peak amplitude of the clock source must be between 0.6V and 3.0V as seen by DECchip 21164-AA. Either a "square-wave" or a sinusoidal source may be used. Note that full-rail clocks may be driven by testers.

Table 8–1: Input Clock Specification

Name	Nominal Bin	Unit
OSC_CLK_IN period min	1.7	nS
OSC_CLK_IN period max	17.0	nS
OSC_CLK_IN symmetry	50 +/- 10	percent
OSC_CLK_IN min. voltage	0.6	V (peak to peak)
OSC_CLK_IN Z input	TBD	

8.3 Pin Characteristics

All DECchip 21164-AA input pins are TTL compatable with the exception of the OSC_IN_H,_L pins and TEMP_SENSE. All output pins are TTL compatable.

8.4 Back-up Cache Loop Timing

DECchip 21164-AA can be configured to support an optional off-chip Back-up Cache (Bcache). DECchip 21164-AA initiated private Bcache read or write (Scache victims) transactions are independent of the system clocking scheme. Bcache loop timing must be an integer multiple of the DECchip 21164-AA cycle time. Outgoing Bcache index and data pins are driven off the internal clock edge and the in-coming Bcache tag and data pins are latched on the same internal clock edge.

Table 8–2: Output Driver Characteristics

Spec	40pF Load	10pF Load	Name
Maximum Driver Delay	2.6nS	1.6nS	**Tdd**
Minimum Driver Delay	1.0nS	1.0nS	**Tmdd**

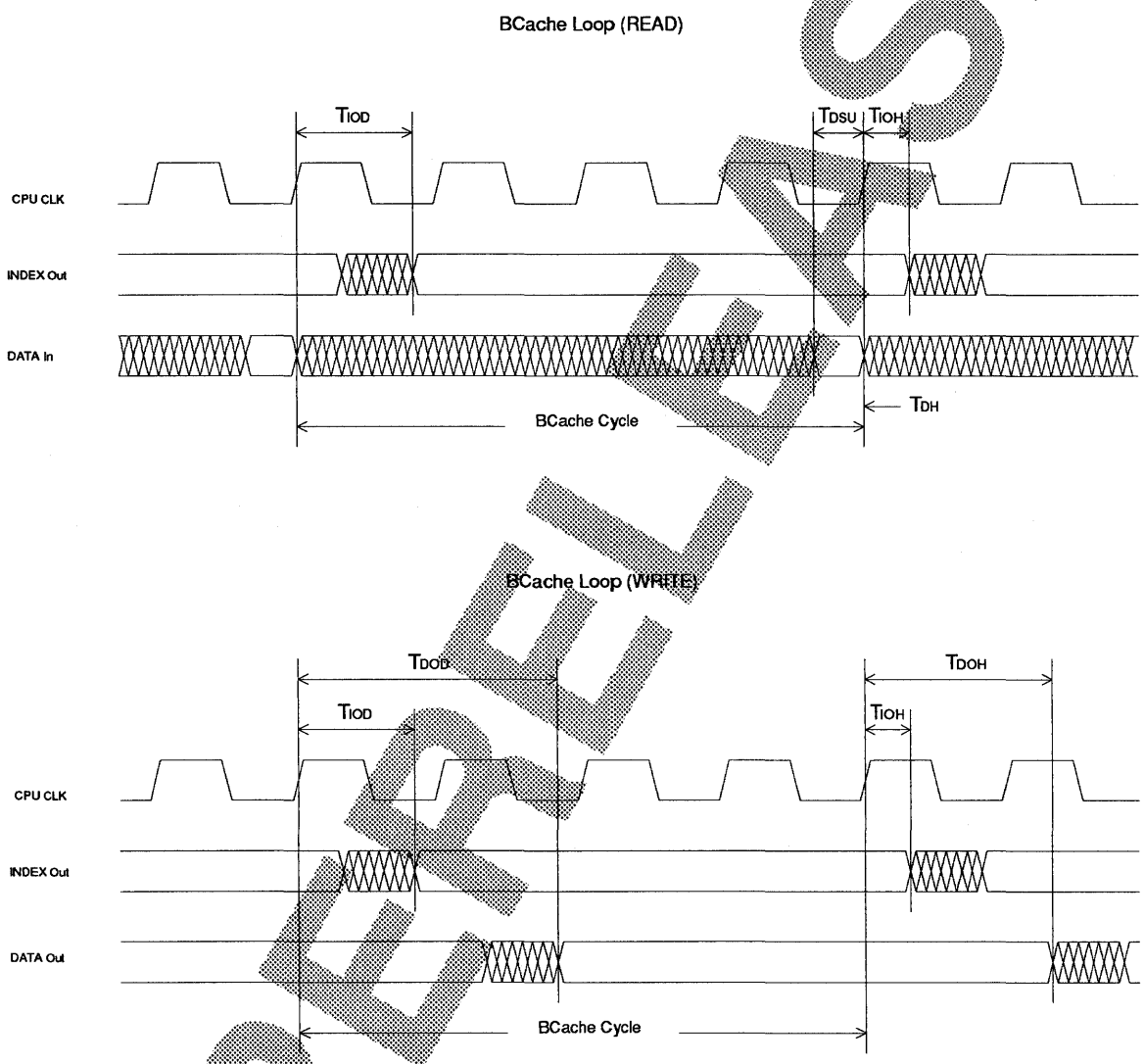
Table 8–3: Bcache Loop Timing

Pin	Spec	Value	Name	Notes
DATA<127:0>	input setup	1.1nS	**Tdsu**	
DATA<127:0>	input hold	0.0nS	**Tdh**	
INDEX<25:4>	output delay	**Tdd** + 0.4nS	**Tiod**	1
INDEX<25:4>	output hold time	**Tmdd**	**Tioh**	
DATA<127:0>	output delay	**Tdd + Tcycle** + 0.4nS	**Tdod**	1
DATA<127:0>	output hold	**Tmdd + Tcycle**	**Tdoh**	

Note:

1. 0.4nS accounts for on chip driver mismatch and clock skew

Figure 8–2: BCache Timing



8.4.1 SYS_CLK based systems

Systems which use the SYS_CLK_OUT1_H,L outputs of DECchip 21164-AA as their system clock will have the following timing. Note that all timing is with respect to the rising edge of the internal CPU CLK, this allows the setup and hold times to be specified independent of the relative capacitive loading of SYS_CLK_OUT1, ADDR, DATA and command pins. REF_CLK_IN_H must be tied to VDD for proper operation.

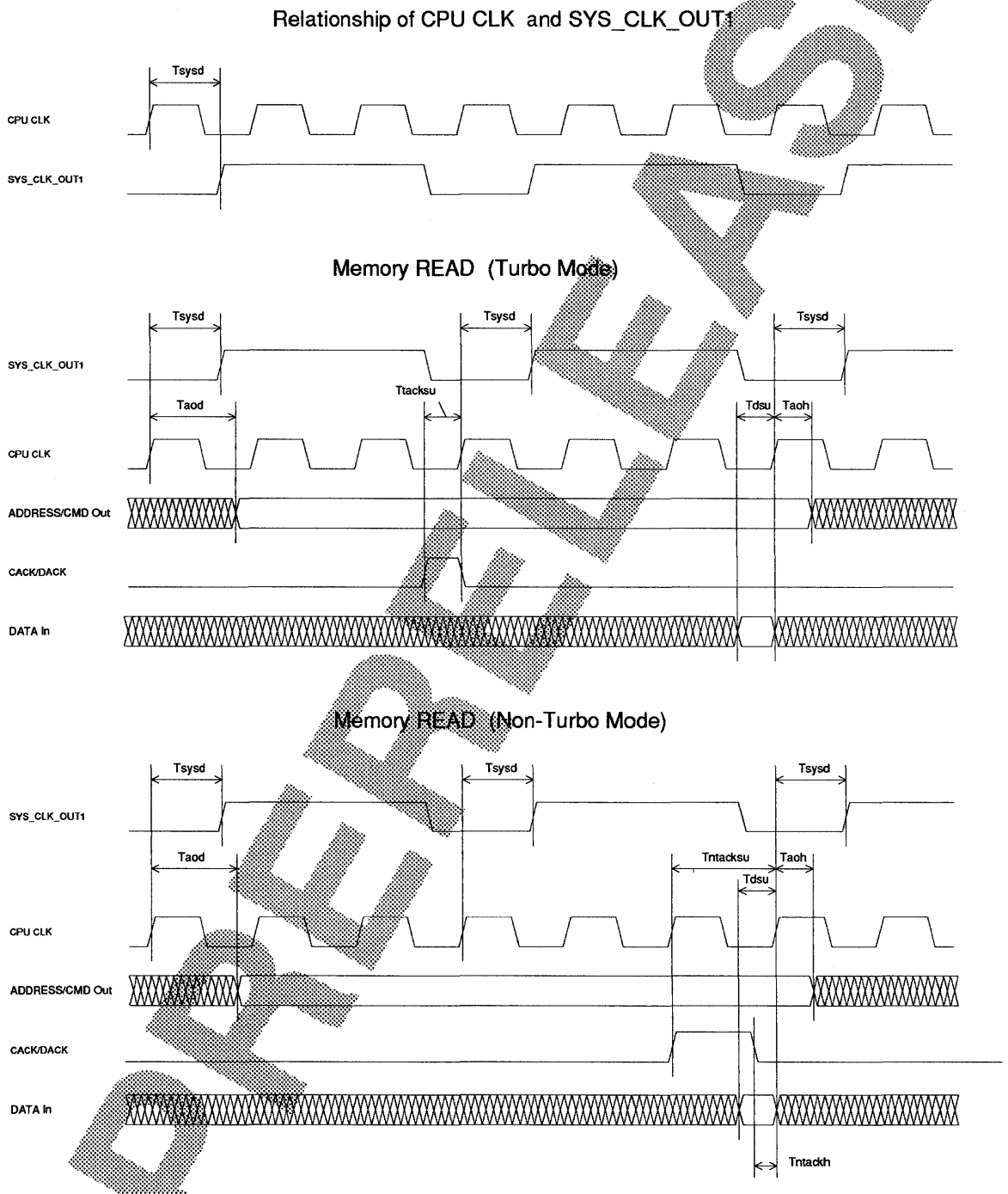
Table 8–4: Systems Using SYS_CLK

Pin	Spec	Value	Name	Notes
SYS_CLK_OUT1	output delay	**Tdd**	**Tsysd**	
SYS_CLK_OUT1	Min. output delay	**Tmdd**	**Tsysdm**	
DATA_BUS_REQ, DATA<127:0>, ADDR<39:4>	input setup	1.1nS	**Tdsu**	
DATA_BUS_REQ, DATA<127:0>, ADDR<39:4>	input hold	0nS	**Tdh**	
ADDR<39:4>	output delay	**Tdd** + 0.4nS	**Taod**	1
ADDR<39:4>	output hold time	**Tmdd**	**Taoh**	
DATA<127:0>	output delay	**Tdd** + **Tcycle** + 0.4nS	**Tdod**	1,2
DATA<127:0>	output hold time	**Tmdd** + **Tcycle**	**Tdoh**	1,2
Non-Turbo Mode				
ADDR_BUS_REQ	input setup	3.8nS	**Tabrsu**	
ADDR_BUS_REQ	input hold	-1.0nS	**Tabrh**	
CACK, DACK	input setup	3.4nS	**Tntacksu**	
CACK, DACK	input hold	-1.0nS	**Tntackh**	
Turbo Mode				
ADDR_BUS_REQ, CACK, DACK	input setup	1.1nS	**Ttacksu**	3
ADDR_BUS_REQ, CACK, DACK	input hold	0nS	**Ttackh**	3

Note:

1. The 0.4nS accounts for on chip driver mismatch and clock skew.
2. For all write transactions initiated by DECchip 21164-AA, data is driven 1 CPU cycle later.
3. In Turbo Mode, control pins are piped on-chip for one SYS_CLOCK_OUT1 before usage.

Figure 8–3: SYS_CLK System Timing



8.4.2 Reference Clocks

Systems which generate their own system clock expect the DECchip 21164-AA to synchronize its SYS_CLK_OUT1_H,L outputs to their system clock. The DECchip 21164-AA uses a Digital Phase Locked Loop (DPLL) to synchronize its SYS_CLK_OUT1 pins to the system clock applied to the REF_CLK_IN pin. The DPLL scheme requires the internal CPU CLK to run slightly faster than the clock applied to the REF_CLK_IN pin. Phase locking is accomplished by forcing the internal CPU CLK to stall for one phase whenever the rising edge of REF_CLK_IN is detected to have occured just before the rising edge of the internal CPU CLK that causes a rising edge of SYS_CLK_OUT1_H. Note that all timing is specified with respect to the rising edge of REF_CLK_IN.

Table 8–5: Systems Using REF_CLK

Pin	Spec	Value	Name	Notes
DATA_BUS_REQ, DATA<127:0>, ADDR<39:4>	input setup	1.1nS	**Tdsu**	
DATA_BUS_REQ, DATA<127:0>, ADDR<39:4>	input hold	0.5 x **Tcycle**	**Tsdadh**	
ADDR<39:4>	output delay	**Tdd** + 0.5 x **Tcycle** + 0.9nS	**Traod**	1
ADDR<39:4>	output hold time	**Tmdd**	**Traoh**	
DATA<127:0>	output delay	**Tdd** + 1.5 x **Tcycle** + 0.9nS	**Trdod**	1,2
DATA<127:0>	output hold time	**Tmdd** + **Tcycle**	**Trdoh**	2
Non-Turbo Mode				
ADDR_BUS_REQ	input setup	3.8nS	**Tntrabrsu**	
ADDR_BUS_REQ	input hold	0.5 x **Tcycle**	**Tntrabrh**	
CACK, DACK	input setup	3.3nS	**Tntracksu**	
CACK, DACK	input hold	0.5 x **Tcycle**	**Tntrackh**	
Turbo Mode				
ADDR_BUS_REQ, CACK, DACK	input setup	1.1nS	**Ttracksu**	3
ADDR_BUS_REQ, CACK, DACK	input hold	0.5 x **Tcycle**	**Ttrackh**	3

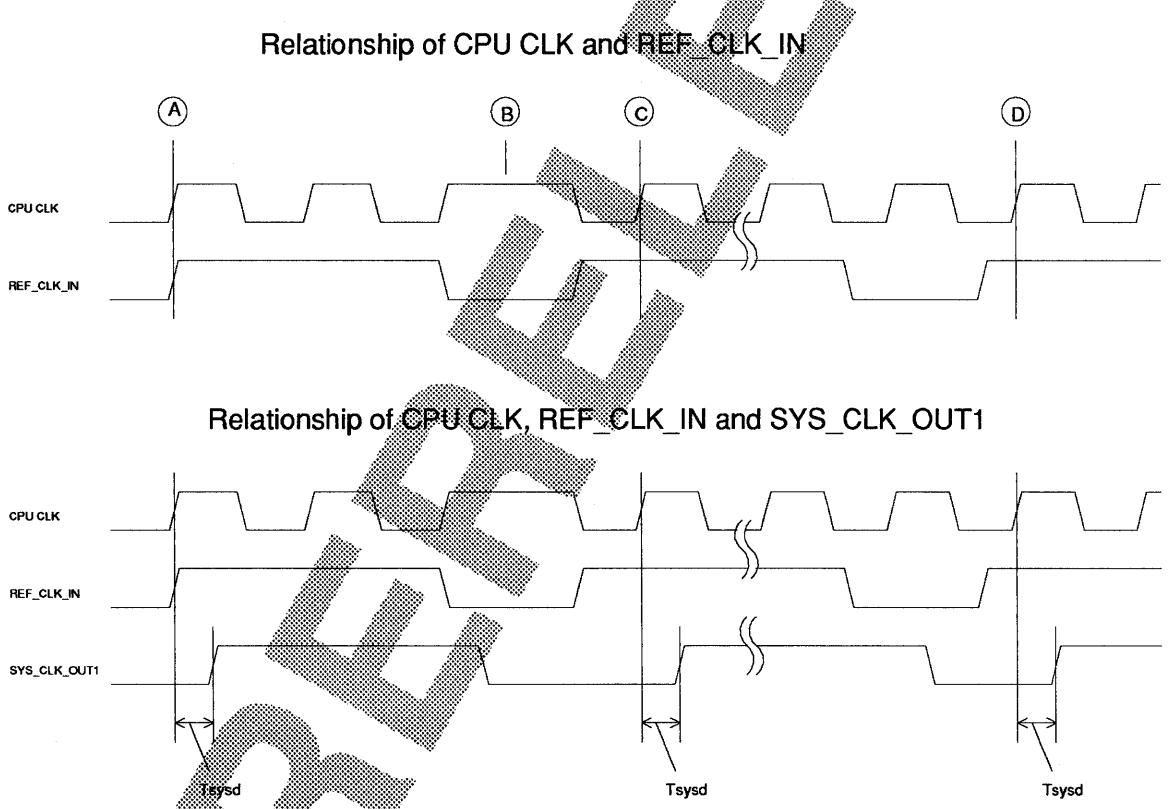
Notes:

1. The 0.9nS accounts for on chip skews which include - 0.4nS for driver mismatch and clock skew; and phase detector skews due to circuit mismatch (0.2nS) and delay in REF_CLK_IN due to the package (0.3nS).
2. For all write transactions initiated by DECchip 21164-AA, data is driven 1 CPU cycle later.
3. In Turbo Mode, control pins are piped on-chip for one SYS_CLOCK_OUT1 before usage.

The operation of the Digital Phase Locked Loop (DPLL) is shown in Figure 8–4. When the internal CPU CLK rising edge is coincident with the rising edge of REF_CLK_IN as shown at **A** in Figure 8–4, the DPPL will cause the internal CPU CLK to stretch for one phase (1 cycle of OSC_CLK_IN) as shown at **B**. This stretch will cause REF_CLK_IN to lead the internal CPU CLK by one phase (shown at **C**), however the CPU CLK is always slightly faster than the external REF_CLK_IN and will over time gain on REF_CLK_IN as shown at **D**. Eventually the gain will equal one phase and a new stretch phase will follow.

Although systems which supply a REF_CLK_IN do not use SYS_CLK_OUT1, a relationship between the two signals exists since the DECchip 21164-AA uses SYS_CLK_OUT1 internally to determine timing during system transactions just as in the SYS_CLK based systems.

Figure 8–4: REF_CLK System Timing



8.4.3 Timing - Other Pins

Table 8–6: Asynchronous Input Pins

REF_CLK_IN_H	SYS_RESET_L	PERF_MON_H
CLK_MODE_H	DC_OK_H	IRQ_H
SYS_MCH_CHK_IRQ_H	PWR_FAIL_IRQ_H	MCH_HLT_IRQ_H

Table 8–7: Timing for SYS_CLK Ratio Programming

Spec	Value
IRQ_H hold time from deassertion of SYS_RESET_L	2nS

Table 8–8: Other Pin List

Input Only		
CFAIL_H	FILL_H	FILL_ID_H
FILL_ERROR_H	FILL_NOCHECK_H	SYSTEM_LOCK_FLAG_H
IDLE_BC_H	PORT_MODE_H	TDI_H
TMS_H	TCK_H	TRST_L
SROM_PRESENT_L	SROM_DAT_H	
Output Only		
VICTIM_PENDING_H	ADDR_RES_H	INT4_VALID_H
SCACHE_SET_H	TAG_RAM_OE_H	TAG_RAM_WE_H
DATA_RAM_OE_H	DATA_RAM_WE_H	TDO_H
SYS_CLK_OUT2	TEST_STATUS_H	CPU_CLK_OUT_H
SROM_OE_L	SROM_CLK_H	
Bi-directional		
CMD_H	ADDR_CMD_PAR_H	DATA_CHECK_H
TAG_DATA_H	TAG_DATA_PAR_H	TAG_VALID_H
TAG_SHARED_H	TAG_DIRTY_H	TAG_CTL_PAR_H

Table 8–9: Other Pin Input Timing (SYS_CLK or REF_CLK based systems)

Spec	Value
input setup	1.1nS
input hold	0nS

Table 8–10: Other Pin List - Output Pin Groupings

Group A		
CPU_CLK_OUT_H	SYS_CLK_OUT2	SROM_CLK_H
SROM_OE_L	TDO_H	TEST_STATUS_H
Group B		
ADDR_CMD_PAR_H	ADDR_RES_H	CMD_H
SCACHE_SET_H	VICTIM_PENDING_H	
Group C		
DATA_CHECK_H	INT4_VALID_H	
Group D		
DATA_RAM_OE_H	DATA_RAM_WE_H	TAG_CTL_PAR_H
TAG_DATA_H	TAG_DATA_PAR_H	TAG_DIRTY_H
TAG_RAM_OE_H	TAG_RAM_WE_H	TAG_SHARED_H
TAG_VALID_H		

Table 8–11: Group Output Timing - SYS_CLK based systems

Spec	Value
Group A	
output delay	**Tdd**
output hold time	**Tmdd**
Group B	
output delay	**Taod**
output hold time	**Taoh**
Group C	
output delay	**Tdod**
output hold time	**Tdoh**
Group D (all output transactions)	
output delay	**Tdod**
output hold time	**Tdoh**

Table 8–12: Group Output Timing - REF_CLK based systems

Spec	Value
Group A	
output delay	**Tdd**
output hold time	**Tmdd**
Group B	
output delay	**Traod**
output hold time	**Traoh**
Group C	
output delay	**Trdod**
output hold time	**Trdoh**
Group D (non-BCache transaction)	
output delay	**Trdod**
output hold time	**Trdoh**
Group D (during BCache transaction)	
output delay	**Tdod**
output hold time	**Tdoh**

8.5 Clock Test Modes

8.5.1 Normal Mode

When the CLK_MODE_H<1:0> pins are not asserted the OSC_CLK_IN frequency is divided by 2. This is the normal operational mode of the clock circuitry.

8.5.2 Chip Test Mode

In order to lower the maximum frequency required to be supplied by the tester, a divide by 1 mode has been designed into the clock generator circuitry. When the CLK_MODE_H<0> pin is asserted and CLK_MODE_H<1> is not asserted the clock frequency applied to the input clock pins OSC_CLK_IN_H,_L bypasses the clock divider and is sent to the Chip Clock driver. This allows the chip internal circuitry to be tested at full speed with a 1/2 frequency (294 MHz) OSC_CLK_IN.

8.5.3 Module Test Mode

When the CLK_MODE_H<0> pin is not asserted and CLK_MODE_H<1> is asserted, the clock frequency applied to the input clock pins OSC_CLK_IN_H,_L is divided by 4 and is sent to the Chip Clock driver. The Digital Phase Locked Loop (DPLL) continues to keep the on chip SYS_CLK_OUT1 locked to REF_CLK_IN within the normal limits if a REF_CLK_IN signal is applied (ie. 0nS to 1 OSC_CLK_IN cycle after REF_CLK_IN).

8.5.4 Clock Test Reset Mode

When both the CLK_MODE_H<0> pin and the CLK_MODE_H<1> pin are asserted, The SYS_CLK generator circuit is forced to reset to a known state. This allows the tester to synchronize the chip to the tester cycle.

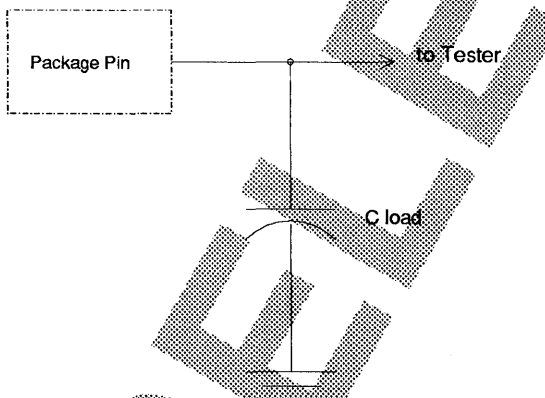
Table 8–13: Test Modes

Mode	CLK_MODE_H<0>	CLK_MODE_H<1>
Normal	0	0
Chip Test	1	0
Module Test	0	1
Clock Reset	1	1

8.6 Test Configuration

All outputs and bidirectional signals including clocks are specified with driving a standard 40pF load as shown below. All input timing is specified with respect to the crossing of standard TTL input levels of 0.8V and 2.0V, output timing is to the nominal CMOS switch point of VDD/2.

Figure 8–5: Test Configuration



8.7 Revision History

Table 8–14: Revision History

Who	When	Description of change
Andy Olesin	Oct. 18, 1993	change output spec for "other pins" list
Andy Olesin	July 27, 1993	changed test reset description
Andy Olesin	May 27, 1993	Refined the AC Spec.
Anil Jain	April 05, 1993	Defined the AC Spec.

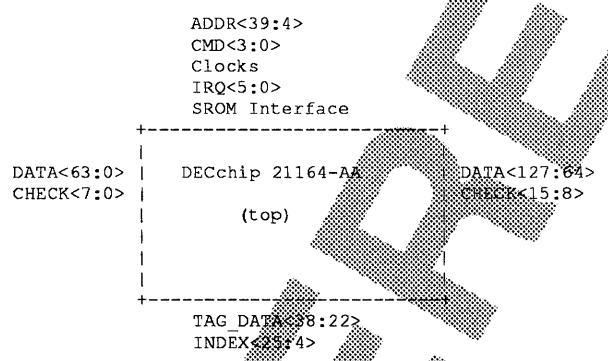
Chapter 9

Pinout

9.1 DECchip 21164-AA Pinout Overview

The DECchip 21164-AA chip is contained in the 503 pin package. 289 of these pins are used for signals, the remaining pins are used for power and ground. Looking down at the top of the chip the crude pinout of DECchip 21164-AA will look like this:

Figure 9–1: DECchip 21164-AA Pinout



9.2 DECchip 21164-AA Signal Pins

The following table is a list of the signal pins on the DECchip 21164-AA chip. In the table, all the pins listed as "O" are output only, those listed as "I" are input only, and those listed as "B" are bidirectional and tri-statable.

Table 9–1: Clock Pins

Signal Name	Type	Function	Reset State	Number
OSC_CLK_IN_H,L	I	CPU clock input	must be clocking	2
CPU_CLK_OUT_H	O	CPU clock output	clock output	1
SYS_CLK_OUT1_H,L	O	System clock output	clock output	2
SYS_CLK_OUT2_H,L	O	System clock output	clock output	2
REF_CLK_IN_H	I	System clock input	-	1
CLK_MODE_H<1:0>	I	Clock logic mode select	-	2
SYS_RESET_L	I[1]	Reset	-	1
Section Total				**11**

[1]This input may be driven asynchronously; an internal synchronizer is implemented.

Table 9–2: System Interface Pins

Signal Name	Type	Function	Reset State	Number
ADDR_H<39:4>	B	Address bus	unspecified[1]	36
CMD_H<3:0>	B	Command bus	NOP[2]	4
ADDR_CMD_PAR_H	B	Odd parity for address and CMD	NOP[2]	1
VICTIM_PENDING_H	O	This miss produced a victim	unspecified	1
ADDR_BUS_REQ_H	I	System wants to use the address and command busses	-	1
CACK_H	I	DECchip 21164-AA command taken	must be deasserted	1
CFAIL_H	I	WRITE_BLOCK_LOCK command failed or request to force Ibox timeout	must be deasserted	1
FILL_ERROR_H	I	request for machine check trap	must be deasserted	1
ADDR_RES_H<1:0>	O	DECchip 21164-AA response to CMD	NOP	2
INT4_VALID_H<3:0>	O	write data valid/INT8 read request	unspecified	4
SCACHE_SET_H<1:0>	O	Scache set allocated	unspecified	2
FILL_H	I	Fill warning	must be deasserted	1
FILL_ID_H	I	Which fill?	should be deasserted	1
DACK_H	I	Data ready or taken	must be deasserted	1
FILL_NOCHECK_H	I	Don't check ECC/parity	should be deasserted	1
SYSTEM_LOCK_FLAG_H	I	Current state of the lock flag	should be deasserted	1
IDLE_BC_H	I	No more CPU accesses to the Bcache	must be deasserted	1
DATA_BUS_REQ_H	I	System wants to use the data bus	-	1
Section Total				**61**

[1]Driven or tristated depending on ADDR_BUS_REQ_H at most recent Sysclock edge. If driven, the value is unspecified.

[2]Driven or tristated depending on ADDR_BUS_REQ_H at most recent Sysclock edge. If driven, the command is NOP.

Table 9–3: Bcache Pins

Signal Name	Type	Function	Reset State	Number
INDEX_H<25:4>	O	Bcache index	unspecified	22
DATA_H<127:0>	B	Data Bus	tristated	128
DATA_CHECK_H<15:0>	B	INT8 ECC check bits or byte parity	tristated	16
TAG_DATA_H<38:20>	B	B-cache tag (1MB min)	tristated	19
TAG_DATA_PAR_H	B	Tag parity	tristated	1
TAG_VALID_H	B	Tag valid	tristated	1
TAG_SHARED_H	B	Tag shared	tristated	1
TAG_DIRTY_H	B	Tag dirty	tristated	1
TAG_CTL_PAR_H	B	Tag V/S/D parity	tristated	1
TAG_RAM_OE_H	O	Tag RAM output enable asserted for reads	asserted	1
TAG_RAM_WE_H	O	Tag RAM write enable	deasserted	1
DATA_RAM_OE_H	O	Data RAM output enable asserted for reads	asserted	1
DATA_RAM_WE_H	O	Data RAM write enable	deasserted	1
Section Total				**194**

Table 9–4: Interrupt and Misc. Pins

Signal Name	Type	Function	Reset State	Number
IRQ_H<3:0>	I[1]	Interrupt requests	Sysclock divisor ratio input	4
SYS_MCH_CHK_IRQ_H	I[1]	System machine check interrupt	Sysclock delay input[2]	1
PWR_FAIL_IRQ_H	I[1]	Power failure interrupt	Sysclock delay input[2]	1
MCH_HLT_IRQ_H	I[1]	Halt request	Sysclock delay input[2]	1
PORT_MODE_H<1:0>	I[1]	Test port mode	-	2
TDI_H	I	IEEE 1149.1 Serial Data Input	-	1
TDO_H	O	IEEE 1149.1 Serial Data Output	-	1
TMS_H	I	IEEE 1149.1 Test Mode Select	-	1
TCK_H	I	IEEE 1149.1 Test Clock	-	1
TRST_L	I	IEEE 1149.1 Test Reset	should be asserted[3]	1
TEST_STATUS_H<1:0>	O	Test status/handshake for BiST	deasserted	2
SROM_PRESENT_L	I[1]		-	1
SROM_OE_L	O	Serial ROM output enable	deasserted[4]	1
SROM_CLK_H	O	Serial ROM clock/Tx data	deasserted[4]	1
SROM_DAT_H	I	Serial ROM data/Rx data	-	1
DC_OK_H	I[1]	Power and clocks ok	-	1
PERF_MON_H	I[1]	Performance monitor input	-	1
TEMP_SENSE	O	Temperature Sense	-	1
Section Total				**23**
Chip Total				289

[1]This(These) input(s) may be driven asynchronously; an internal synchronizer is implemented.

[2]Input for SYS_CLK_OUT2_H,L delay relative to SYS_CLK_OUT1_H,L.

[3]TRST_L can be asserted during normal operation to ensure the IEEE 1149.1 port remains inactive. The pin has special functions in test modes. See Chapter 11.

[4]If PORT_MODE_H<0> is asserted, this output is unspecified.

9.3 Revision History

Table 9–5: Revision History

Who	When	Description of change
Pete Bannon	3/22/92	New pinout
Pete Bannon	4/22/92	Change assertion of TAG_WE, TAG_OE
Pete Bannon	10/22/92	Update test pins
JHE	4-DEC-1992	Add reset information.
JHE	25-OCT-1993	edits: clk_mode, ref_clk_in_h,vref_h,ecl_out_h,srom_present,por mode

Chapter 10

The Package

TBS

This chapter is To Be Supplied.

10.1 Revision History

Table 10–1: Revision History

Who	When	Description of change

Chapter 11

Test Interface and Testability Features

11.1 Introduction

The DECchip 21164-AA CPU chip's testability features address broad issues of providing cost-effective and thorough testing of DECchip 21164-AA through its life cycle. Some specific goals supported by DECchip 21164-AA testability features include:

- Chip debug.
- Efficient and thorough testing of embedded RAM arrays.
- Built-in Self Repair (BiSr) of instruction cache (ICache) and support for reduced probe test for efficient and low cost wafer probe testing.
- High fault coverage chip manufacturing test.
- Effective burn-in test.
- Module assembly verification test via IEEE 1149.1 architecture.
- Automatic power-on Built-in Self-test (BiSt) of the ICache.
- Limited support for concurrent fault detection in fault tolerant system that employ duplicate DECchip 21164-AAs.

The testability features included on DECchip 21164-AA include ICache self-test and self-repair, internal Linear Feedback Shift Registers (LFSRs) and scan observability registers, support for reduced probe count wafer probe test, IEEE 1149.1 test access port and boundary scan register, and several other test features. DECchip 21164-AA also includes a comprehensive test interface port that permits efficient access to the chip's testability and diagnosability features during debug and manufacturing testing phases.

11.2 Test Port

Test Interface Port on DECchip 21164-AA consists of 13 dedicated pins that support three port interface modes: 1) Normal mode, 2) Manufacturing test mode, and 3) Debug test mode. Table 11–1 summarizes the test port pins and their functions in the three modes.

Table 11–1: DECchip 21164-AA Test Port Pins and Port Modes

	Normal Function		Manufacturing		Debug	
Pin Name	**Typ**	**Signal**	**Typ**	**Signal**	**Typ**	**Signal**	**Typ**
PORT_MODE_H<1>	I	LOW	I	LOW	I	HIGH	I
PORT_MODE_H<0>	B	LOW	I	HIGH	I	dbg_data_h< 8>	O
SROM_PRESENT_L	B	srom_present_l	I	test control	I	dbg_data_h< 7>	O
SROM_DATA_H	I	srom_data_h/Rx	I	srom_data_h	I	srom_data_h/Rx	I
SROM_CLK_H	O	srom_clk_h/Tx	O	obs_data_h< 8>	O	srom_clk_h/Tx	O
SROM_OE_L	O	srom_oe_l	O	obs_data_h< 7>	O	srom_oe_l	O
TDI_H	B	tdi_h	I	obs_data_h< 6>	O	dbg_data_h< 6>	O
TDO_H	O	tdo_h	O	obs_data_h< 5>	O	dbg_data_h< 5>	O
TMS_H	B	tms_h	I	obs_data_h< 4>	O	dbg_data_h< 4>	O
TCK_H	B	tck_h	I	obs_data_h< 3>	O	dbg_data_h< 3>	O
TRST_L	B	trst_l	I	obs_data_h< 2>	O	dbg_data_h< 2>	O
TEST_STATUS_H<0>	O	test status	O	test status / obs_ data_h< 1>	O	dbg_data_h< 1>	O
TEST_STATUS_H<1>	O	test status	O	test status / obs_ data_h< 0>	O	dbg_data_h< 0>	O

11.2.1 Normal Test Interface Mode

The test port is in normal test interface mode when the PORT_MODE_H<1:0> are tied to 00. This is the default mode. In this mode the test port supports a serial ROM interface, a serial diagnostic terminal interface, and an IEEE 1149.1 test access port.

11.2.1.1 SROM Port

SROM_PRESENT_L, SROM_DATA_H, SROM_OE_L, SROM_CLK_H constitute the SROM interface.

If serial ROMs (such as an AMD Am1736) are present in the system, the pin SROM_PRESENT_L may be pulled down on the board. DECchip 21164-AA samples this pin during the system reset. If the pin is pulled down during the system reset, then the DECchip 21164-AA's reset sequence automatically loads its ICache from serial ROMs before executing its first instruction. If SROM_PRESENT_L is pulled-up during system reset, the SROM load is disabled. In this case the ICache valid bits are cleared by the reset sequence, causing the first instruction fetch to miss the ICache and seek the instructions from the off chip memory.

During SROM load:

- SROM_OE_L signal supplies the output enable to the serial ROM, serving both as an output enable and as a reset (refer to the serial ROM specifications for details).

  DECchip 21164-AA asserts this signal low for the duration of ICache load from serial ROM. Once the load is complete, the signal is deasserted.

- SROM_CLK_H output signal supplies the clock to the ROM that causes it to advance to the next bit. The cycle time of this clock is 128 times the cpu clock rate.
- SROM_DATA_H pin reads the serial ROM data.

The serial ROMs can contain enough ALPHA code to complete the configuration of the external interface (e.g. setting the timing on the external cache RAMs, and diagnose the path between the CPU chip and the real ROM).

DECchip 21164-AA is in PALmode following the deassertion of system reset and the conclusion of ICache self-test - this gives the code loaded into the ICache access to all of the visible state within the chip.

See Section 11.4 for the details of the ICache fill operation from SROMs.

11.2.1.2 Serial Terminal Port

Once the data in the serial ROM has been loaded into the ICache, the three SROM Port pins turn into a simple parallel I/O pins that can be used to drive a diagnostic terminal such a RS422.

When the serial ROM is not being read, the SROM_OE_L output signal is false. The serial diagnostic terminal port is enabled if this pin is wired to the active high enable of an RS422 (or 26LS32) receiver driving onto SROM_DATA_H and to the active high enable of an RS422 (or 26LS31) driver driven from srom_clk_h pin. The CPU allows SROM_DATA_H to be read and SROM_CLK_H to be written by PALcode. This supports a bit-banged serial interface.

IPRs associated with this interface are described in the chapter on PAL Code and IPRs.

11.2.1.3 IEEE 1149.1 Test Access Port

TDI_H, TDO_H, TCK_H, TMS_H and TRST_L make up the IEEE 1149.1 test access port. This port accesses DECchip 21164-AA chip's boundary scan register and chip tri-state functions for board level manufacturing test. The port also allows access to the die identification code. The port is compliant with all requirements of IEEE 1149.1 test access port. See IEEE Std. 1149.1 "A Test Access Port and Boundary Scan Architecture" for the full description of the specification.

Figure 11–1 shows the user-visible features from this port.

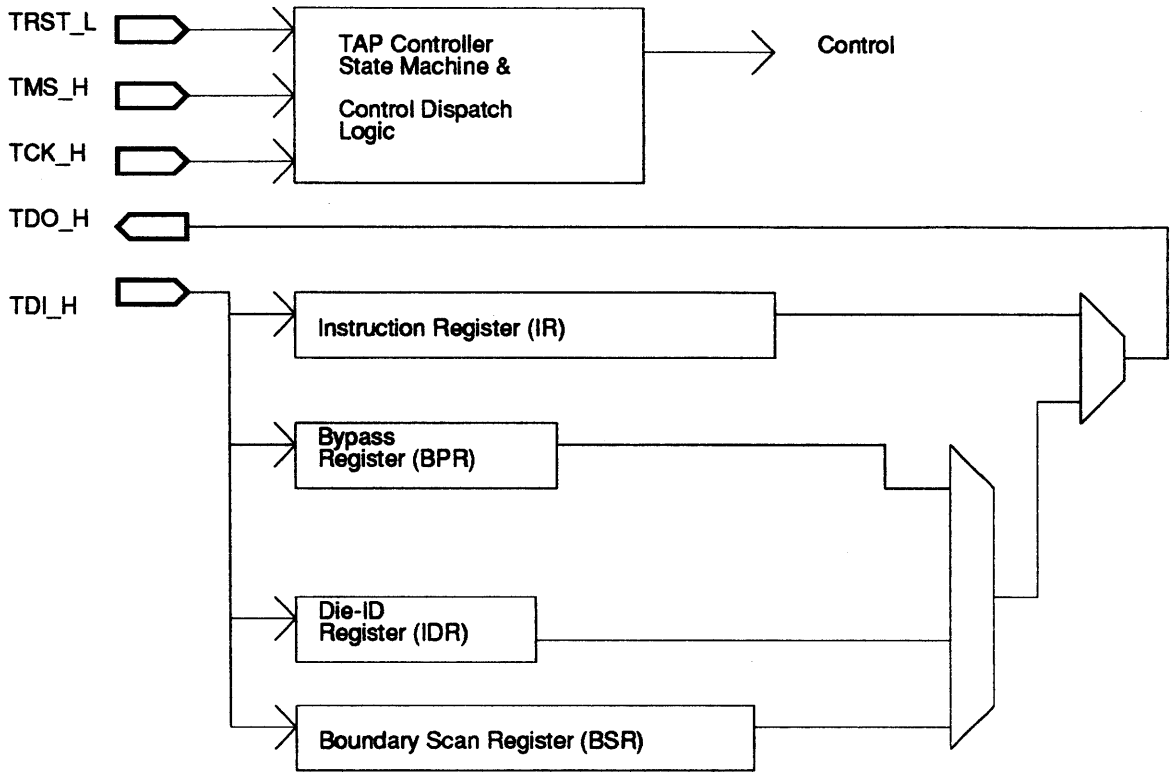
TAP Controller

The TAP Controller contains a state machine. It interprets IEEE 1149.1 protocols received on TMS_H signal and generates appropriate clocks and control signals for the testability features under its jurisdiction.

Bypass Register

The Bypass Register is a 1-bit shift register. It provides a short single-bit scan path through the port (chip).

Figure 11–1: IEEE 1149.1 Test Access Port



Instruction Register

The Instruction Register (IR) is 3-bits wide. It supports EXTEST, SAMPLE, BYPASS, HIGHZ and DIE_ID instructions. Table 11–2 summarizes the instructions and their functions.

During the capture operation, the shift register stage of IR is loaded with '001'. This automatic load feature is useful for testing the integrity of the IEEE 1149.1 scan chain on module.

Table 11–2: Instruction Register

IR< 2:0>	Name	Scan Register Selected	Remarks
111	BYPASS	Bypass Register	Default.
110	HIGHZ	Bypass Register	Tristates all I/O and output pins
101	BYPASS	Bypass Register	Duplicate BYPASS
100	HIGHZ	Bypass Register	Duplicate HIGHZ
011	DIE_ID	Die ID Register	
010	SAMPLE	Boundary Scan	
001	DIE_ID	Die ID Register	Duplicate DIE_ID
000	EXTEST	Boundary Scan Register	BSR drives chip I/O and output pins

Note that the SAMPLE, BYPASS and DIE_ID instructions are non-intrusive. That is, they could be operated while chip is doing its normal functions. EXTEST and HIGHZ instructions force chip's internal logic to a reset state.

Die-ID Register

Die-ID Register is 32-bit scan register. It shifts out fuse-programmed die information. The format and content of the information to be programmed will be determined by the manufacturing.

Boundary Scan Register

Boundary Scan Register on DECchip 21164-AA is approx. 286 **TBD** bits long. It supports SAMPLE and EXTEST instructions. See Section 11.9 for the organization of this register.

Effects of EXTEST and HIGHZ Instruction

The effect of EXTEST or HIGHZ instruction on DECchip 21164-AA chip is as follows

- EXTEST instruction allows the boundary scan register to have complete control over the output and bidirectional pins. HIGHZ instruction forces all output and bidirection pins to a high impedance state.
- The effect on clock input and output pins is **TBD**.
- The internal chip logic is forced to a reset state. This prevent the cpu from reacting to irrelevant test data that may appear at the chip's inputs.

11.2.1.4 Test Status Pins

Two test status pins TEST_STATUS_H<1:0> pins are used for extracting of test status information from the chip. System reset drives both test status pins low.

- During ICache BiSt

  TEST_STATUS_H<<0> is asserted high to indicates that the ICache BiSt has failed. TEST_STATUS_H<1> is asserted high to indicate presence of more than two failing rows.

The start of ICache BiSt forces TEST_STATUS_H<0> pin to go high. If the ICache BiSt passes, TEST_STATUS_H<0> is deasserted, otherwise it remains asserted. TEST_STATUS_H<1> is asserted as high as soon as third bad ICache row is detected. This nay be used to detect unrepairable ICache early, thus reducing average test time. System users may ignore this pin.

- During On-Line LFSR mode

  When the internal LFSRs are turned on in on-line mode (ON_OBL_1 command described later), the TEST_STATUS_H<0> outputs the quotient generated by the observability LFSRs. A new quotient bit is observed with every system clock rising edge. This feature is useful to people implementing fault tolerant systems. Also, the feature can be exploited for the burn-in and life test for monitoring failures. See Section 11.6.2 for more details.

- IPR Read/writes to Test Status Pins

  PALcode can write to TEST_STATUS_H<1> and can read the TEST_STATUS_H<0> via hardware IPR access. See Chapter 3.

The default operation for TEST_STATUS_H<0> pin is to output the BiSt result. The default operation for TEST_STATUS_H<1> pin is to output the IPR written value.

11.2.2 Manufacturing Test Interface Mode

The DECchip 21164-AA test port is in Manufacturing Test Interface Mode when PORT_MODE_H<1:0> are tied to 01 (binary). This mode allows control of ICache test features, internal LFSR and Scan Observability Registers, and efficient byte-serial read-out of observability features, including ICache bit map. Figure 11–2 shows the user-visible features during manufacturing test interface mode.

The SROM_PRESENT_L pin is used for test control. Asserting a high on this pin initiates a test operation state. In this state, DECchip 21164-AA chip automatically loads the 8-bit Test Command Register and executes all required test actions, including any additional shift operations. Input test data is serially fed at the SROM_DATA_H pin. Test results from chip are shifted out byte-serially (9 bits at a time) on the test pins.

The SROM_PRESENT_L pin may be returned to low once test shift operation has been initiated. A new test command may be loaded by once again asserting a high on SROM_PRESENT_L pin after all actions of the previous command have been completed.

When the manufacturing test interface mode is activated, all inputs to the IEEE 1149.1 port are driven with their default values.

Test Command Register (TCR)

Test Command Register is 8 bits wide. Table 11–3 summarizes the test commands and their actions.

Figure 11–2: Manufacturing Test Interface Mode

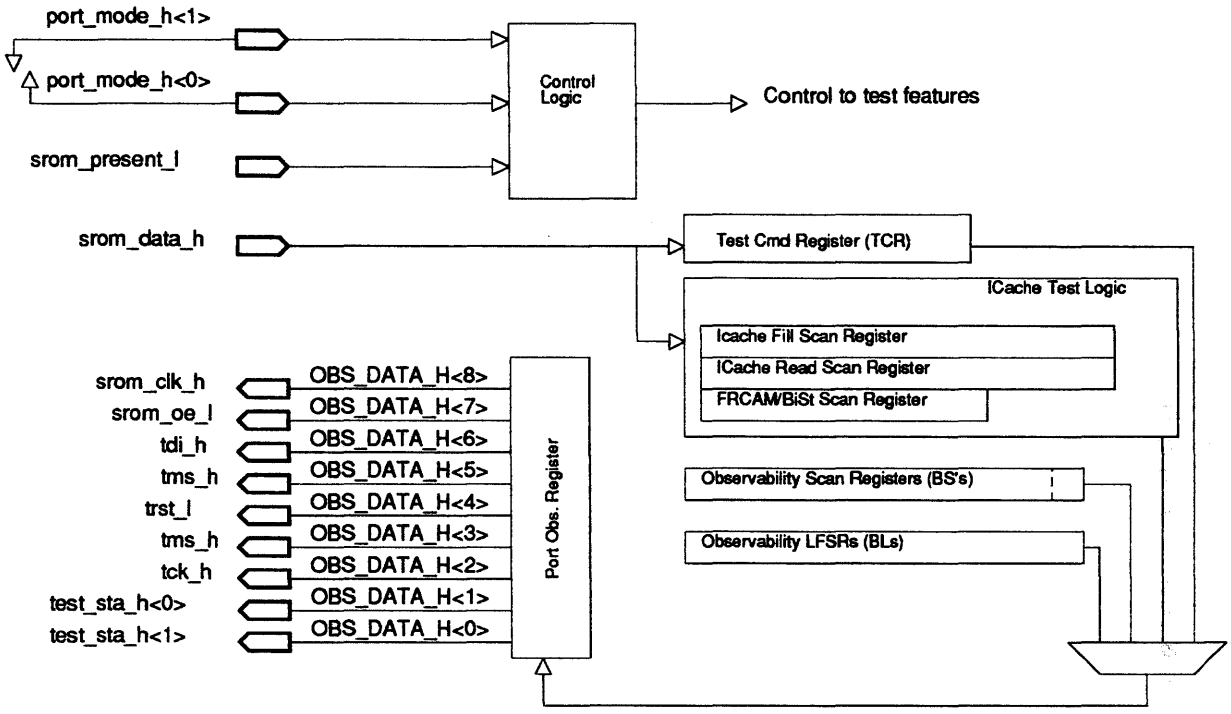


Table 11–3: Test Command Register

TCR< 7:0>	Command Mnemonic	Action
00 000 XXX	RD_ICache	Reads out ICache contents on test port. Useful for debug/bit mapping etc.
00 001 00X	WR_IC_F0	Writes ICache serially. Data shifts at system clock rate. Internal chip reset extended. Used for subsequent read out for ICache test purposes.
00 001 01X	WR_IC_F1	Writes ICache serially. Data shifts at system clock rate. Internal chip reset NOT extended. Used for speedier ICache fill during manufacturing.
00 001 10X	WR_IC_S0	Writes ICache serially. Data shifts at slow rate. (cycle time = cpu clock cycle * 128) rate. Internal chip reset extended.
00 001 11X	WR_IC_S1	Writes ICache serially. Data shifts at slow rate. (cycle time = cpu clock cycle * 128) rate. Internal chip reset not extended. This instruction is forced by CPU during power-on/reset sequence to automatically load from SROM.
00 100 XXX	LD_BKG	Loads ICache fill Scan path with background pattern. This instruction is forced by the BiSt logic.
00 101 XXX	SC_FRCAM	Scans out Failing Row CAM on test port. This instruction is forced by the BiSt logic.
00 110 XXX	SC_BIST	Scans out portions of BiSt logic for testing the BiSt logic.
00 010 XXX	RU_BIST	Runs ICache BiSt. This instruction is forced by the power-on/reset sequence.
00 011 XXX	RU_RETENT	Runs ICache Retention BiSt.
00 111 XXX	IC_NOP	No ICache action. However, forces internal chip reset.
01 ss dddd	SC_src_delay	Scans out selected register. src = 0X selects LFSR scan path. src = 1X selects internal scan register. delay selects cycle (0 to 15) to be captured for observation. The command performs the capture-scan out sequence continuously, until another test command is loaded.
10 X0 0XXX	OFF_OBL	Turns off Observability LFSR data compression mode.
10 X0 10XX	ON_OBL_0	Turns On the Observability LFSR data compression in off-line mode.
10 X0 11XX	ON_OBL_1	Turns On the Observability LFSR data compression in On-line mode.
10 X1 0XXX	OFF_CBL	Turns Off (if previously turned on) the intrusive controllability features, LFSR pattern generators, etc on the chip.
10 X1 1XXX	ON_CBL	Turns On the intrusive controllability features (such as LFSR pattern generators) on the chip.
11 XX XXXX		Reserved.
11 11 1111	PRT_NOP	No test port actions.

Notes:

1. The internal chip logic is forced to reset during all ICache test commands.
2. The cycle time for shifting data during SROM load is 128 * cpu clock cycle. Assuming the fastest DECchip 21164-AA cycle time of 3ns, this translates to the fastest shift rate of 384ns..
3. The scan and LFSR observability registers can be operated and read out without interfering with normal system operation.

ICache Fill Scan Register

This is a 200-bit long scan register used for filling the ICache serially from SROMS or tester. See section Section 11.4 for the details of the serial fill operation.

ICache Read Scan Register

This is 100-bit long read scan register path used for dumping the ICache contents. See section Section 11.4 for the details of the serial read operation.

Observability LFSRs

This is TBD-bit register used for enhancing fault coverage of manufacturing test. See section Section 11.6 for details.

Observability Scan Registers

This is TBD-bit register used primarily for chip debug. See section Section 11.7 for details.

Controllability Features

Test Port also has the provision for supporting internal controllability features. If these features are provided, they are turned on and off via the ON_CBL and OFF_CBL test commands.

FRCAM Scan Register

This scan register is 13-bit long. It consists of 12 bits of failing row CAM and unrepairable_ic flag. See Section 11.3 for more details.

Port Observability Register

This is 9-bit serial-in Parallel-out observability register. The parallel outputs of the register update the corresponding test port pins every system clock cycle. This allows tester to observe 9-bits of scan data simultaneously. This reduces the vector depth requirement on chip tester's failure capture memory (DFM) by a factor of eight.

The internal observability LFSRs and the Internal Scan Registers shift at the chip's internal clock rate. The scan paths in ICache test logic shift at the system clock rate.

11.2.3 Debug Test Interface Mode

DECchip 21164-AA test port is in Debug Test Interface Mode when PORT_MODE_H<1> is tied to 1. Debug Test Interface Mode allows the critical chip nodes to be monitored in parallel.

Signals to be observed on parallel port are selected by TBD IPR bits. (See chapter on PALCode and IPRs for the details.)

Restrictions of parallel debug test port

1. When parallel debug port is activated, all inputs corresponding to the normal test input pins are fed with their default values.
2. The PORT_MODE_H<1> pin allows to switch back and forth between the normal test port and the parallel debug port.
3. **Parallel debug port is designed to support chip/system debugging in prototype system environments only.** Some small logic may be required to ensure that there is no interference with other chips connected to the test port.

11.2.4 Activating Debug/Manufacturing Port Modes in a System

Both Debug and Manufacturing port modes can be activated in a system by incorporating a few jumpers, and if necessary, some support logic. Jumpers are required as some of the test pins are shared for outputing the debug/observability information from the chip. Jumpers prevent observability data from interfering with the operation of the other chips connected to the shared test pins. Support logic is required only if system wants to load test commands automatically through the manufacturing test port mode, for example, to turn on/off the observability LFSRs in on-line mode.

Figure 11–3 shows a typical module and the places where jumpers may be necessary to activate the debug and manufacturing test port modes.

11.3 ICache BiSt

The DECchip 21164-AA ICache is tested by Built-in Self-test that implements a full march algorithm. The self test logic covers all three (Data, BHT, and TAG) ICache arrays.

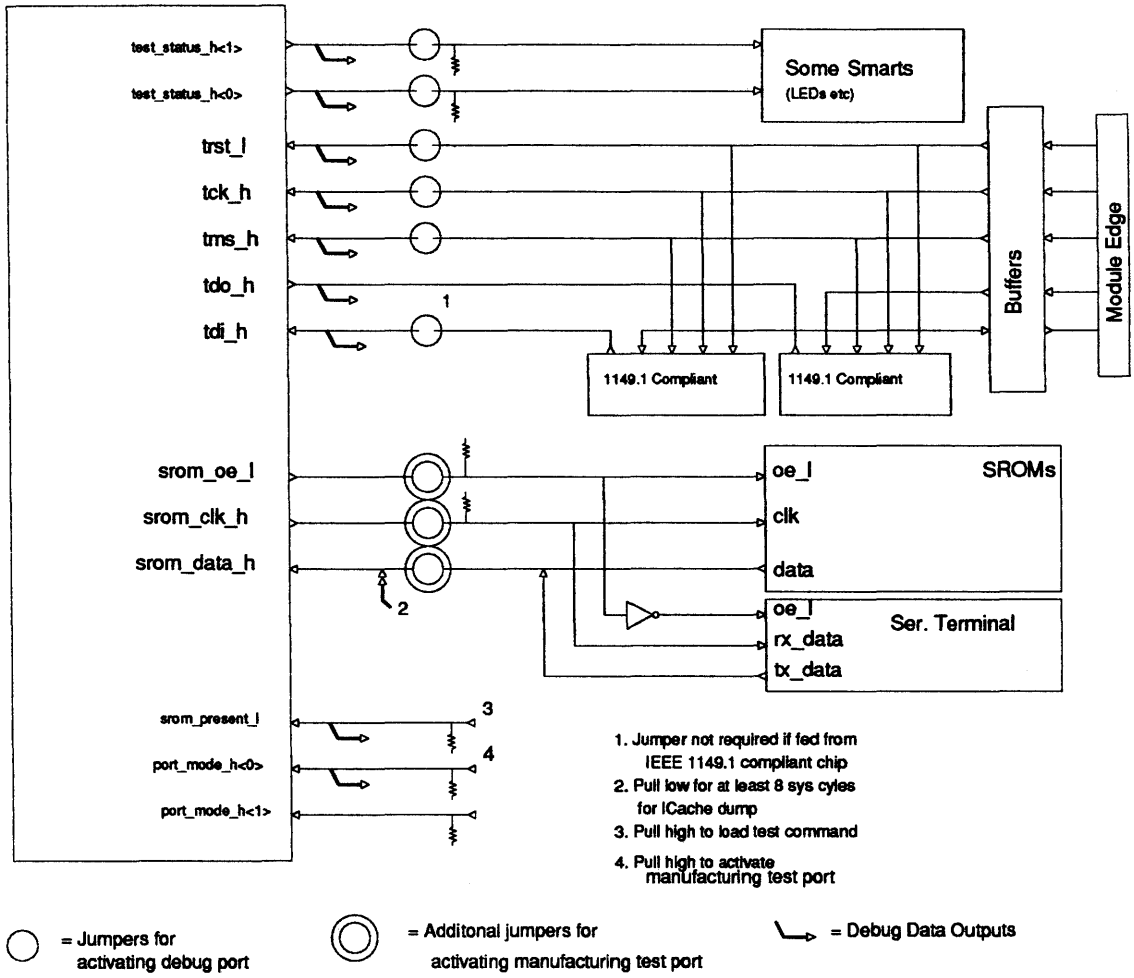
ICache BiSt is invoked automatically upon deassertion of system reset if the BiSt is not bypassed. BiSt is bypassed if the PORT_MODE_H<1> pin is asserted high during system reset.

BiSt Bypass feature allows ICache BiSt as well as the Built-in Self Repair to be bypassed during debug and in between pattern runs on testers, if so desired.

BiSt runs for **TBD** cycles.

The Go/NoGo result of BiSt is made available on TEST_STATUS_H<0> pin. TEST_STATUS_H<0> is forced low by the system reset and high with the start of the BiSt. If at the end of BiSt, any of the ICache rows are bad, the pin remains asserted high, otherwise it is deasserted. Software can read this status through an IPR. If ICache fails in more than two rows, TEST_STATUS_H<1> is asserted high. This pin is cleared by the the system reset.

Figure 11–3: Tes Port Connections on Module



Built-in Self-Repair

When the BiSt is invoked on wafers that have not gone through the fuse repair process, the ICache BiSt sequence automatically performs the following steps.

- Perform the BiSt. Store up to two the failing row addresses in the failing row CAMs.
- Self repair the ICache data array.
- Repeat BiSt.
- Dump the content of the failing row CAMs on the test port.

The repair information shifted out consists of the following bits.

Table 11–4: FRCAM Scan Register Organization

Field Name	Extent	
unrepairable_ic_flag	0	High = unrepairable cache
CAM_0 Valid Flag	1	High = 1st repair address valid
CAM_0 Reg	2:6	1st repairable row-pair address
CAM_1 Valid Flag	7	High = 2nd repair address valid
CAM_1 Reg	8:12	2nd repairable row pair address

Notes:

- The automatic BiSt and BiSR run identically under the normal and the manufacturing test interface modes.
- Built-in self-repair feature is available only prior to laser repair process. BiSt logic uses a fuse programmed internal signal to determine whether the BiSR is required.

11.4 ICache Serial Write and Read Operations

Serial Write Operation

The ICache can be written serially from the SROM or for testing purposes from the SROM port pins. On DECchip 21164-AA, all ICache bits, including each block's tag, ASN, ASM, valid and branch history bits can be loaded serially. Once the serial load has been invoked (either automatically by the chip reset sequence, or via the IC_WR_xx command from the manufacturing test port), the entire cache is loaded automatically from the lowest to the highest addresses.

The serial bits are received in a 200-bit long Fill Scan Path from which they are written in parallel into the ICache address. The Fill scan path is organized as shown in Figure 11–4. The farthest bit (tag< 42>) is shifted in first and the nearest bit (BHT< 7>) is shifted in last. Note that the data and predecode bits in the data array are interleaved.

The automatic serial fill invoked by the chip reset sequence occurs at the slower SROM clock rate (period = cpu clock rate * 128). The serial invoked by IC_WR_xx can occur at the SROM rate or at the system clock rate. In either case, the ICache fill operation is automatic.

Serial Read Operation

All three ICache arrays can be read out serially for testing purposes. Manufacturing port's IC_RD command initiates the serial read operation. Arrays are dumped from the lowest to the highest address. The data is first received into a READ Scan Path (RSP), from which it is serially shifted to the test port's Port Observability Register at the system clock rate. The data can be read out at the test port pins 9-bits at a time.

Figure 11–4: SROM Fill Scan Path Bit Order

```
SROM_DATA_H          serial input ->
BHT Array             7 ->   6 -> ... ->   0 ->
Data                127 ->  95 -> 126 ->  94 -> ... ->  96 ->  64 ->
Predecodes           19 ->  14 ->  18 ->  13 -> ... ->  15 ->   9 ->
Data parity           b ->
Predeocde parity      b ->
Predecodes            9 ->   4 ->   8 ->   3 -> ... ->   5 ->   0 ->
Data                 63 ->  31 ->  62 ->  30 -> ... ->  32 ->   0 ->
Tag Parity            b ->
Tag Valids            0 ->   1 ->
TAG ASM               b ->
TAG ASN               0 ->   1 -> ... ->   7 ->
TAGs                 13 ->  14 -> ... ->  42

   b = Single bit signal
```

Figure 11–5: Read Scan Path Bit Order

```
Serial out           serial out <-
BHT array leader     dmy <- err <- rf1 <- rf0 <-
BHT Bits               7 <-   6 <- ... <-   0 <-
Data array leader    dmy <- err <- rf1 <- rf0 <-
Data Bits            d37 <-  36 <- ... <-   0 <-
Tag array leader     dmy <- err <- rf1 <- rf0 <-
Tag Parity             b <-
Tag Valids             0 <-   1 <-
TAG ASM                b <-
TAG ASN                0 <-   1 <- ... <-   7 <-
TAGs                  13 <-  14 <- ... <-  42

    b        = Single bit signal
    dmy      = Dummy bit. Makes RSP for the array even bit length
    err      = Error bit. Useful for BiSt logic testability
    rf1,rf0  = Used by BiSt logic to store reference patterns
```

The RSP is 100-bits long and consists of three segments: 12-bit BHT segment, 42-bit Data array segment, and 46-bit Tag array segment. Besides the bits that capture data from ICache array, each segment has 4 extra bits used by the BiSt logic.

The 150 bits of data from the data array are read into the 38 bits of Read Scan Path via a multiplexer which selects one of the four physically adjacent data bits. The entire array is read by making four passes through the ICache addresses. (Note that this causes the BHT and tag arrays to be read four times!) This necessitates that the data dumped by the serial ICache read operation must be carefully reconstructed before interpreting them.

The organization of the bits in the read scan path is shown in Figure 11–5.

11.5 SCache/DCache Test Features

See PALCode and IPR chapter and the cache section in chapter on DECchip 21164-AA Microarchitecture. Also, see Section 11.7 for description of SCache scan chain.

SCache Test and Repair Algorithm

TBD.

11.6 Observability LFSRs (OBLs)

11.6.1 Organization

DECchip 21164-AA implements several Observability LFSRs (OBLs) to enhance the fault coverage. The OBLs are turned and off on by the ON_OBL_x and OFF_OBL test commands described in Section 11.2.2. LFSRs also operate as ordinary scan registers. They are read out by the SC_src_delay test command.

All LFSRs in DECchip 21164-AA are accessed from a single scan chain. Figure 11–6 summarizes the LFSR organization. The details of the signals captured and the LFSR design (feedback taps) are given in Table 11–5. .

Figure 11–6: LFSR Chain Organization

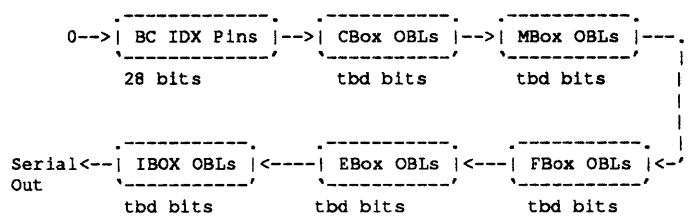


Table 11–5: Observability LFSR Organization

LFSR Name: Backup Cache Index Pins

Size: 28 bits

Feedback polynomial: 2200000001(Octal, Taps bits 28 and 25)

Access Chain Number: ....

Bit #	Signal name	Remarks
28	feedback	
27:26	p%ev5_sc_set_h< 01:1>	unprobed outputs
25	feedback	
24:3	p%bc_index_h< 4:25>	Unprobed outputs
2:1	p%ev5_adr_res_h< 1:0<	Unprobed outputs

Table 11–5 (Cont.): Observability LFSR Organization

0	available

LFSR Name:
Size:
Feedback polynomial: ....
Access Chain Number: ....

Bit #	Signal name	Remarks
0	Tbd	TBD

(As the design work progresses, more details of LFSR operation will be added here.)

11.6.2 On-line LFSR Operation

DECchip 21164-AA supports an on-line testing mode via its observability LFSRs. The quotient bit generated by the observability LFSR in IBOX is brought out to the TEST_STATUS_H<0> pin when the LFSRs are turned on in an on-line mode (ON_OBL_1 test command). Monitoring and comparing this pin with the expected serial stream can provide an indication of DECchip 21164-AA health on the fly.

This feature can be exploited by the fault tolerant systems that employ multiple redundant DECchip 21164-AAs. They can compare the TEST_STATUS_H<0> on two or more DECchip 21164-AAs performing identical tasks. The same principle can be extended in other test applications such as burn-in test for monitoring failures.

During the on-line test mode, a new quotient bit is observed with every system clock rising edge. Since the observability LFSRs work at the CPU clock rate, not every quotient bit is observed on TEST_STATUS_H<0>. This is generally acceptable since typically an error in an input to an LFSR produces a multitude of erroneous quotient bits.

Note that the LFSRs must be turned on only after DECchip 21164-AA initialization has been completed.

11.7 Observability Scan Registers (OBSs)

Internal Scan Registers offer observability of debug-critical signals. They are accessed from the test port under the manufacturing test interface mode as described in Section 11.2.2. The capture action of internal scan register occurs TBD cpu cycles after the Test Command Register is loaded with the appropriate SCAN command. Table 11–6 gives organization of the DECchip 21164-AA's Observability Scan Registers.

Table 11–6: Observability Scan Register Organization

Scan Chain Name: SCache

Size: 164 bits

Scan Chain for Part 1 of SCache:

Bit #	Signal name	Remarks
0	S%IFB_PAR_H<0>	LW Parity for Data<31:0>
1:32	S%IFB_H<0:31>	Data<0:31>
33	S_DIR_CTL%LSEL_WSC_H	LW write enable for Data<31:0>
34:44	S_DCR%ADDR_7A_L<14:4>	Address driven to SCache
45	S_DCR%HIT_H<2>	SET_HIT signal, set 2
46	S_DCR%HIT_H<1>	SET_HIT signal, set 1
47	S_DCR%HIT_H<0>	SET_HIT signal, set 0
48	S_DIR_CTL%RSEL_WSC_H	LW write enable for Data 63:32
49:80	S%IFB_H<32:63>	Data<32:63>
81	S%IFB_PAR_H<1>	LW Parity for Data<63:32>

Scan Chain for Part 2 of SCache:

Bit #	Signal name	Remarks
0	S%IFB_PAR_H<2>	LW Parity for Data<95:64>
1	S%IFB_H<64:95>	Data<64:95>
33	S_DIL_CTL%LSEL_WSC_H	LW write enable for Data<95:64>
34:44	S_DCL%ADDR_7A_L<14:4>	Address driven to SCache
45	S_DCL%HIT_H<2>	SET_HIT signal, set 2
46	S_DCL%HIT_H<1>	SET_HIT signal, set 1
47	S_DCL%HIT_H<0>	SET_HIT signal, set 0
48	S_DIL_CTL%RSEL_WSC_H	LW write enable for Data<127:96>
49:80	S%IFB_H<96:127>	Data<96:127>
81	S%IFB_PAR_H<3>	LW Parity for Data<127:96>

11.8 Controllability Features

TBD.

11.9 Boundary Scan Register

DECchip 21164-AA Boundary Scan Register is approx. 286 bits long. Table 11–7 gives the boundary scan register organization. The Boundary scan register begins at the TDI_H pin and traverses in clock-wise direction and ends at TDO_H pin.

NOT FINAL

The list below is based on the DECchip 21164-AA die size and pad assignments as of 11/23/92.

Table 11–7: Boundary Scan Register Organization

Signal Name	Type	Count	BSR Cell	Remarks
P%TDI	B	1	None	
P%SROM_OE_L	O	1	out_bcell	
P%SROM_CLK_H	O	1	out_bcell	
P%SROM_DATA_H	B	1	in_bcell	
P%SROM_PRESENT_L	B	1	in_bcell	was SROM_DISABLE
P%PORT_MODE_H< 0:1>	I	2	in_bcell	
P%SYS_RESET_L	I	1	in_bcell	
P%DC_OK_H	I	1	in_bcell	
P%SYS_MCH_CHK	I	1	in_bcell	
P%PWR_FAIL_IRQ	I	1	in_bcell	
P%MCH_HALT_IRQ	I	1	in_bcell	
P%IRQ< 3:0>	I	4	in_bcell	
P%CLK_IN_H, _L	I	2	in_bcell	
P%CPU_CLK_OUT_H	O	1	out_bcell	
P%SYS_CLK_OUT_H, _L	O	2	out_bcell	
P%SYS_CLK2_OUT_H, _L	O	1	out_bcell	
P%ECL_OUT_H	I	1	in_bcell	
P%VREF_H	I	1	in_bcell	
P%REF_CLK_IN_H, L	I	2	in_bcell	
P%PERF_MON< 0>	I	1	in_bcell	
P%ADDR< 21:5>	B	17	io_bcell	U-R corner
P%ADDR< 4>	B	1	io_bcell	U-R corner
P%DATA< 063:0>	B	64	io_bcell	
P%DATA_CHECK< 0:7>	B	8	io_bcell	
P%DATA_VALID< 1:0>	O	2	out_bcell	
P%EV5_SC_SET< 1:0>	O	2	out_bcell	L-R Corner
P%BC_INDEX< 25:4>	O	22	out_bcell	L-R Corner

Table 11–7 (Cont.): Boundary Scan Register Organization

Signal Name	Type	Count	BSR Cell	Remarks
P%EV5_ADDR_RES< 1:0>	O	2	out_bcell	
P%IDLE_BC	I	1	in_bcell	
P%SYS_LCK_FLG	I	1	in_bcell	
P%DATA_BUS_REQ_H	I	1	in_bcell	
P%ADDR_BUS_REQ_H	I	1	in_bcell	
P%FILL_NOCHK	I	1	in_bcell	
P%FILL_ERR	I	1	in_bcell	
P%FILL_ID_H	I	1	in_bcell	
P%FILL_H	I	1	in_bcell	
P%DACK_H	I	1	in_bcell	
P%CFAIL_H	I	1	in_bcell	
P%CACK_H	I	1	in_bcell	
P%ADDR_CMD_PAR_H	B	1	io_bcell	
P%VTM_PENDING	O	1	in_bcell	
P%DATA_RAM_WE	O	1	out_bcell	
P%DATA_RAM_OE	O	1	out_bcell	
P%TAG_RAM_WE	O	1	out_bcell	
P%TAG_RAM_OE	O	1	out_bcell	
P%EV5_CMD< 0:3>	B	4	io_bcell	
P%TAG_DAT_PAR	B	1	io_bcell	
P%TAG_CTL_PAR	B	1	io_bcell	
P%TAG_DIRTY	B	1	io_bcell	
P%TAG_SHARED	B	1	io_bcell	
P%TAG_VALID	B	1	io_bcell	
P%BC_TAG< 20:38>	B	19	io_bcell	L-L Corner
P%DATA_VALID< 2:3>	O	2	out_bcell	
P%DATA_CHECK< 15:8>	B	8	out_bcell	
P%DATA< 064:127 >	B	64	io_bcell	
P%ADDR_H< 39:37>	B	3	io_bcell	U-L Corner
P%ADDR_H< 36:22>	B	15	io_bcell	U-L Corner
P%spare		1	io_bcell	Captures zero
P%TEST_STATUS_H< 1:0>	O	2	out_bcell	
P%TRST_L	I	1	None	
P%TCK	B	1	None	
P%TMS	B	1	None	
P%TDO	O	1	None	

Table 11–7 (Cont.): Boundary Scan Register Organization

Signal Name	Type	Count	BSR Cell	Remarks
en_for_left_data	sig	1	out_bcell	tbd
en_for_right_data	sig	1	out_bcell	tbd
en_for_bc_tag	sig	1	out_bcell	tbd
en_for_??	sig	1	out_bcell	tbd

11.10 Testability IPRs

The following is the list of IPRs connected to testability features. See chapter on PALCode and IPRs for more details.

1. TEST_STATUS_H<0> read and TEST_STATUS_H<1> write (ICSR).
2. Debug port visibility select bits in IPRs (TBD).
3. Serial Terminal Port IPRs (SL_RCV, SL_XMIT)
4. Scache IPRs (SC_CTL, SC_ADDR)
5. Dcache IPRs (DC_MODE, DC_TEST_CTL, DC_TEST_TAG_TEMP)
6. Bcache IPRs (BC_CONTROL, BC_TAG_ADDR)

11.11 Test Feature Reset and Initialization

Reset, initialization and defaults of testability features are described through-out this chapter and in the chapter on Reset and Initialization. For convenience, this section summarizes the power-on reset sequence, as it pertains to the testability features for the normal operation. The sequence of events is as follows:

1. SYS_RESET_L is asserted.
2. The values on the SROM_PRESENT_L and PORT_MODE_H<1> pins are sampled on SYS_RESET_L deassertion.
3. If BiSt is bypassed (indicated by a '1' sampled on PORT_MODE_H<1>), go to the next step.

   If the BiSt is not bypassed, keep rest of the chip in reset state. Perform ICache BiSt (and BiSR, if BiSR is required). Clear ICache Tag valid bits at the end of BiSt.
4. If SROMs are not present, (indicated by '1' sampled on SROM_PRESENT_L), go to the next step.

   If SROMs are present, keep rest of the chip in reset state. Load ICache from the SROMs.
5. Deassert internal reset. Fetch the first instruction.

Table 11–7 (Cont.): Boundary Scan Register Organization

Signal Name	Type	Count	BSR Cell	Remarks
en_for_left_data	sig	1	out_bcell	tbd
en_for_right_data	sig	1	out_bcell	tbd
en_for_bc_tag	sig	1	out_bcell	tbd
en_for_??	sig	1	out_bcell	tbd

11.10 Testability IPRs

The following is the list of IPRs connected to testability features. See chapter on PALCode and IPRs for more details.

1. TEST_STATUS_H<0> read and TEST_STATUS_H<1> write (ICSR).
2. Debug port visibility select bits in IPRs (TBD).
3. Serial Terminal Port IPRs (SL_RCV, SL_XMIT)
4. Scache IPRs (SC_CTL, SC_ADDR)
5. Dcache IPRs (DC_MODE, DC_TEST_CTL, DC_TEST_TAG_TEMP)
6. Bcache IPRs (BC_CONTROL, BC_TAG_ADDR)

11.11 Test Feature Reset and Initialization

Reset, initialization and defaults of testability features are described through-out this chapter and in the chapter on Reset and Initialization. For convenience, this section summarizes the power-on reset sequence, as it pertains to the testability features for the normal operation. The sequence of events is as follows:

1. SYS_RESET_L is asserted.
2. The values on the SROM_PRESENT_L and PORT_MODE_H<1> pins are sampled on SYS_RESET_L deassertion.
3. If BiSt is bypassed (indicated by a '1' sampled on PORT_MODE_H<1>), go to the next step.

   If the BiSt is not bypassed, keep rest of the chip in reset state. Perform ICache BiSt (and BiSR, if BiSR is required). Clear ICache Tag valid bits at the end of BiSt.
4. If SROMs are not present, (indicated by '1' sampled on SROM_PRESENT_L), go to the next step.

   If SROMs are present, keep rest of the chip in reset state. Load ICache from the SROMs.
5. Deassert internal reset. Fetch the first instruction.

11.12 Open Issues

1. Should we make chip run w/o external oscillator and with internal PLL during EXTEST and HIGHZ instructions?
2. Details of bits in OBL and OBS chains to be defined.
3. Details of signals brought to the parallel debug port need to be defined.
4. The following additional test feature enhancements on boundary scan are currently being considered
   - CLAMP_IO Instruction.
   - A ring oscillator mode for the boundary scan.

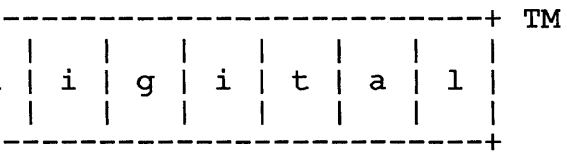
11.13 Revision History

Table 11–8: Revision History

Who	When	Description of change
Dilip Bhavsar	2/13/92	Working draft
Dilip Bhavsar	6/25/92	Working draft
Dilip Bhavsar	7/1/92	Working draft
Dilip Bhavsar	9/16/92	Rev 0.1 Changes: Second test_status_h pin added. TCR size changed from 4 to 8 bits to program cpu cycle to be captured during scan. Opcodes redefined
Dilip Bhavsar	11/23/92	Rev 1.0 Clean-up and updates

ROCK::BANNON "Pete Bannon HLO2-3/J12 225-5249 02-Dec-1993 1743 +0000"
993 18:32:10.40
@SPECHOLDERS.DIS,@EV5_CBOX_TEAM

EV5 interface update



To: @spec_holders

Date: 2-December 1993
From: Peter Bannon
Dept: SEG/HPC
DTN: (8)225-5249
Loc/Mail: HLO2-3/D11
Net mail: ROCK::BANNON

TITLE: EV5 Interface Update

1 INTRODUCTION

The purpose of this note is to update everyone on changes to the EV5 interface since Rev 1.9 of the spec. This note was first released on February 24, 1993. Change bars mark updates since that release.

2 KNOWN BUGS IN PASS1

The following bugs have been found in EV5 and will not be fixed for PASS1. Systems should be careful to avoid these problems.

2.1 WRITE_BLOCK_LOCK

A WRITE_BLOCK_LOCK is caused by a store conditional instruction to I/O space. Two octawords of data will be provide by EV5, each requiring a DACK. If the system asserts DACK for the first octaword, and asserts CACK and CFAIL at the time, and the sysclk ratio is 3, EV5 will hang.

If DACK, CACK, and CFAIL are asserted for the second cycle the write will be failed correctly.

If CACK and CFAIL are asserted at any time without DACK, the write will be failed correctly.

If the sysclk ratio is something other than 3, any legal combination of DACK, CACK, and CFAIL will cause the write to fail correctly.

2.2 WRITE_BLOCK

When doing WRITE_BLOCK, EV5 should align the address so ADDR_H<5:4> is zero for 64 byte block systems or ADDR_H<4> is zero for 32 byte block systems. This works correctly for shared writes and writes to I/O space.

However, if the Bcache is off, and EV5 is using a WRITE_BLOCK command to write back an Scache victim to memory, EV5 may not correctly align ADDR_H. The data will be written as if the alignment had been done. To avoid this problem, system should ignore ADDR_H<5:4> (ADDR_H<4> for 32 byte systems) when the address for a WRITE_BLOCK is in cachable space (ADDR_H<39>=0).

If the Bcache is enabled this problem will not occur.

3 IPR CHANGES

Please consult the October pre-release of the IPR chapter for details. A copy can be obtained by contacting John Edmondson, ROCK::EDMONDSON.

4 UNUSED PINS

Unused pins may be left unconnected.

5 PRIVATE READ TIMING

There have been two changes to private read timing. The addition of wave pipelining and changes to the timing of output enable.

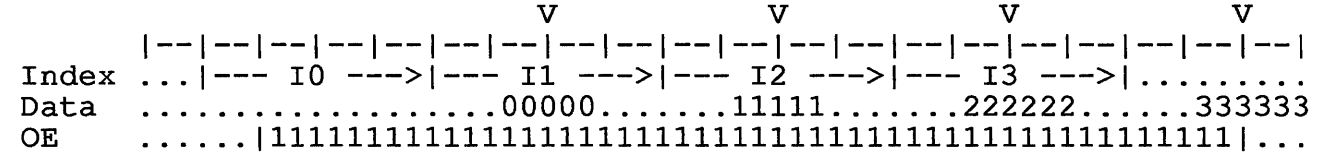
5.1 Wave Pipelining

Wave pipelining has been added for 64B block size caches. Wave pipelining is not supported for systems that have a 32B block size.

To enable wave pipelining, the BC_RD_SPD should be set to the latency of the Bcache read. BC_CONTROL<18:17> should be set to the number of cycles to subtract from BC_RD_SPD to get the Bcache repetition rate. For example, if BC_RD_SPD is set to 7 and BC_CONTROL<18:17> is set to 2, it will take 7 cycles for valid data to arrive at the pins, but a new read will start every 5 cycles.

The read repetition rate must be greater than 3. For example it is not permitted to set BC_RD_SPD to 5 and BC_CONTROL<18:17> to 2.

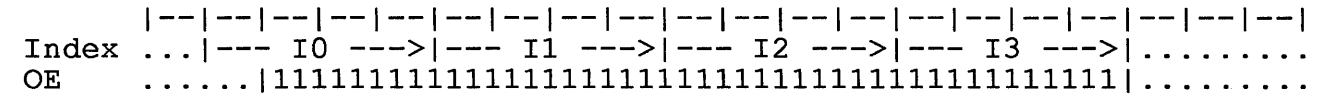
BC_RD_SPD=6, BC_CONTROL<18:17>=2:
V - shows were EV5 would clock the Bcache data into the pad ring



The value of BCCONTROL<18:17> should be added to the normal value of BC_CNFG<14:12> to increase the time between reads and writes. This will prevent a write from starting before the last data of a read is received.

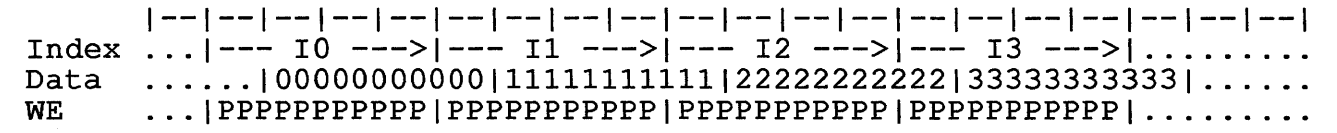
5.2 Output Enable

TAG_RAM_OE_H and DATA_RAM_OE_H will not assert during the first CPU cycle of a private Bcache read.



5.3 Private Write Timing

To improve the data hold time at the end of the write, the data will be driven one CPU cycle after the index. It will be de-asserted one cycle after the index. For private writes to the Bcache the write pulse can be programmed in each cycle.



P - programmable write pulse

Note that the minimum value of BCCONFIG<18:16>, FILL_WE_OFFSET is still 1, which will add one CPU cycle of disabled write pulse at the start of each fill.

6 CALCULATING IDLE_BC

The rule for calculating IDLE_BC has changed. The new equation

is:

```
if (block_size == 32b)
  x = 2
else
  x = 0

read_hit_idle  = 2 + x + (block_size/16)*BC_RD_SPD +
                   tri-state_ram_turn_off - 3*wave_pipelining
read_miss_idle = 7 + BC_RD_SPD + Sysclk_ratio + tri-state_RAM_turn_off
write_idle     = 4 + (block_size/16)*BC_WRT_SPD + tri-state_EV5_turn_off
```

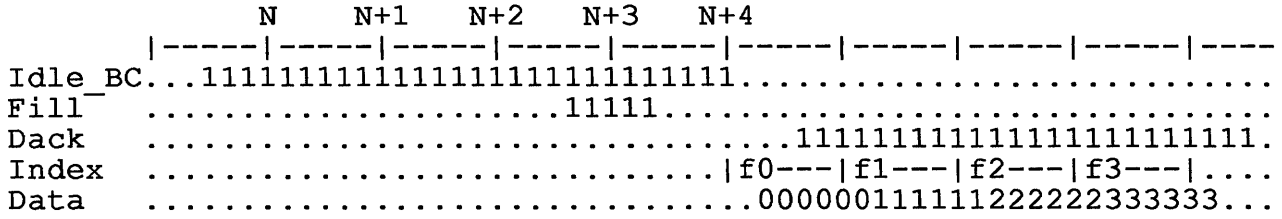
Take the largest of the three times and then round up to the next Sysclk boundary.

When determining the tri-state_turn_off times if the System will not turn on it's drivers for some number of nanoseconds after EV5 starts driving the Bcache index, this time can be used to reduce the tri-state_turn_off time.

For example if the sysclk ratio is 6, 64b block, Bcache read/write speed of 5, with no wave pipelining, 2 cycles for tri-state_read, 0 cycles for tri-state_write, the equations would work out to:

```
read_hit_idle  = 2 + 0 + (64/16)*5 + 2 - 3*0 = 24
read_miss_idle = 7 + 5 + 6 + 2 = 20
write_idle     = 4 + (64/16)*5 + 0 = 24

   MAX(24/6,20/6,24/6) = 4
```



If EV5 receives IDLE_BC asserted at sysclk edge N, the FILL command can be received at sysclk edge N+3. EV5 will drive the index to fill the Bcache on sysclk edge N+4.

7 FILLS

During a fill from memory, the Bcache tag store is written for each INT16 of data. The system is required to drive the share, dirty, and parity bits with the correct value for the entire fill.

8 TRI-STATE

8.1 Private Read Or Write To FILL

The time required to tri-state the EV5 drivers at the end of a write, or the the Bcache drivers at the end of a read is part of the IDLE_BC equation.

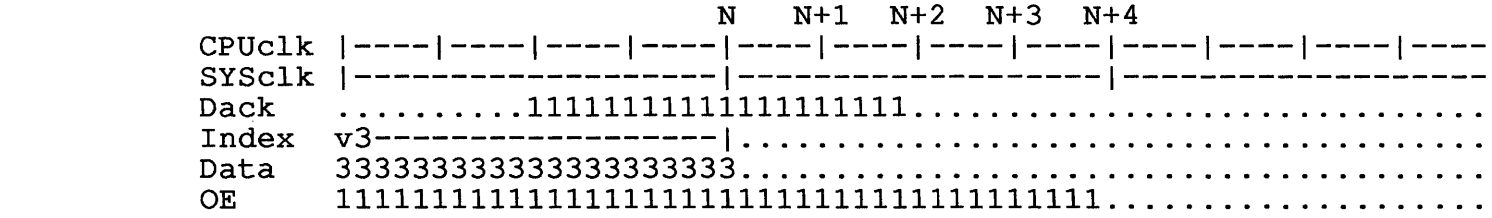
8.2 Bcache Victim To FILL

There are two read miss with victim cases to consider, one if the READ_MISS will be first, and another if the READ_MISS will be second.

8.2.1 Read Miss First -

The time to turn off the Bcache drivers at the end of a Bcache victim is fixed by the EV5 design. The system must allow for this time before starting a fill.

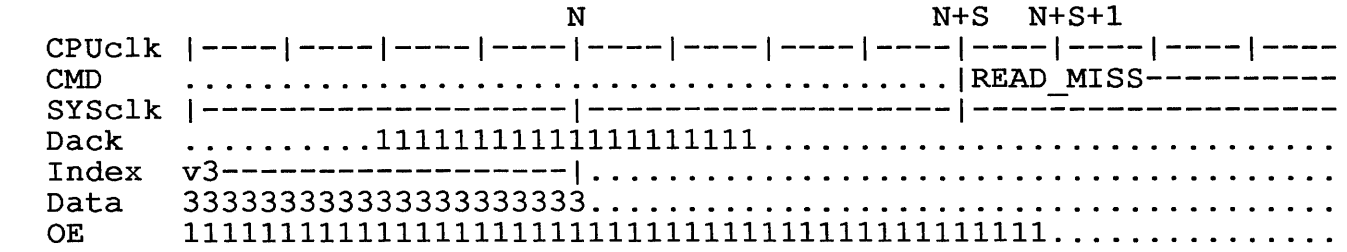
The final DACK will be received by EV5 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, OE will de-assert at the rising edge of CPU clock N+4.



8.2.2 Read Miss Second -

The time to turn off the Bcache drivers at the end of a Bcache victim is fixed by the EV5 design. The system must allow for this time before starting a fill.

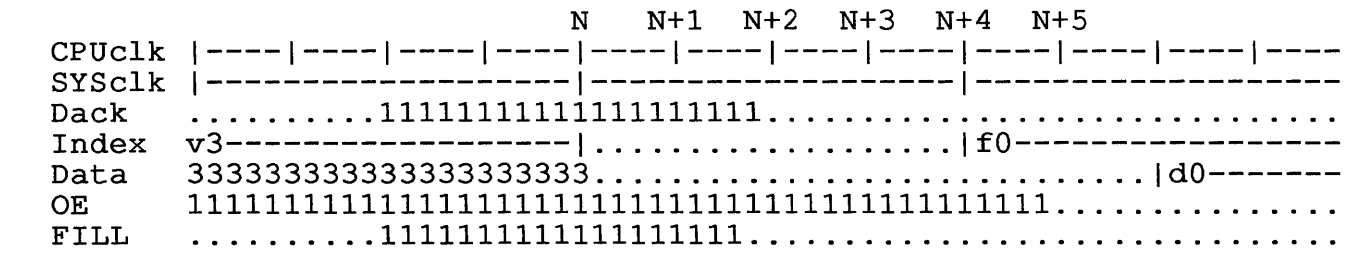
The final DACK will be received by EV5 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, then the READMISS command will arrive on the next sysclk edge, and the OE will de-assert at the rising edge of CPU clock N+S+1, where S is the sysclk ratio. If the sysclk ratio is 3, it will take an extra sysclk to send the READMISS command, so the OE will de-assert at N+2S+1.



8.3 System Command To FILL

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

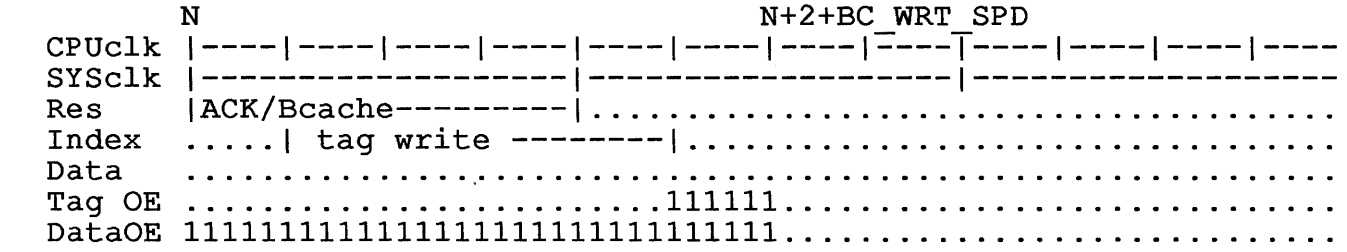
The final DACK will be received by EV5 on the rising edge of sysclk. If the corresponding rising CPU clock edge is labeled N, OE will de-assert at the rising edge of CPU clock N+5.



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

If the system command was a set shared or an invalidate command, the system must allow time for EV5 to complete the Bcache tag write and then for the drivers to turn off before driving the TAG_SHARED, TAG_DIRTY, and TAG_CTL_PAR wires.

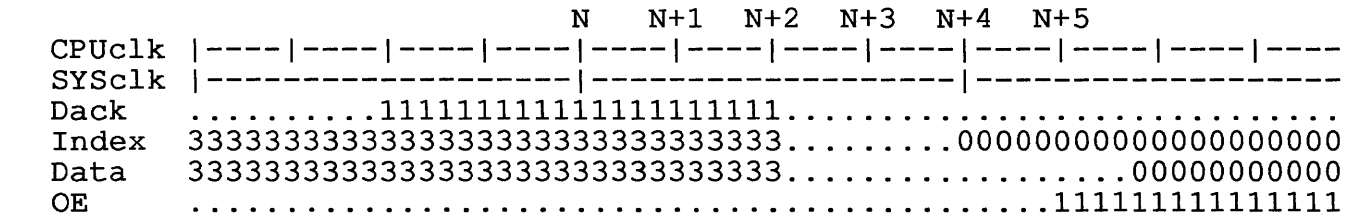
EV5 will begin the tag write one CPU cycle after the response is sent to the system. The write will take BC_WRT_SPD cycles to complete. During the write the DATA_RAM_OE will be asserted. TAG_RAM_OE will not. At the end of the write TAG_RAM_OE will pulse for one CPU cycle and then both will go off. In the picture below, if the response is is driven at the rising edge of CPU clock N, then the OE will fall at N+2+BC_WRT_SPD, or N+6 for a 4 cycle write speed.



8.4 FILL To Private Read Or Write

At the end of the fill, EV5 will not begin to drive the data bus until the 5th CPU cycle after the Sysclk that loads the last DACK. EV5 will not assert OE until the 5th cycle after the Sysclk that loads the last DACK.

If systems require more time to turn off their drivers, they must use IDLE_BC and DATA_BUS_REQ at the end of the fill to stop EV5 requests and not send any system requests.



9 COMMAND ENCODINGS

Two new commands have been added to support write lockout on TurboLaser systems. Other systems should respond to the READ_MS_MOD_LKn commands as if it were are regular READ_MISS_MODn.

```
        EV5_CMD__NOP                    0
        EV5_CMD__LOCK                   1
        EV5_CMD__FETCH                  2
        EV5_CMD__FETCH_M                3
        EV5_CMD__MB                     4
        EV5_CMD__SET_DIRTY              5
        EV5_CMD__WRITE_BLOCK            6
        EV5_CMD__WRITE_BLOCK_LOCK       7
        EV5_CMD__READ_MISS0             8
        EV5_CMD__READ_MISS1             9
        EV5_CMD__READ_MISS_MOD0         10
        EV5_CMD__READ_MISS_MOD1         11
        EV5_CMD__BCACHE_VICTIM          12
|       EV5_CMD__RESERVED              13
        EV5_CMD__READ_MS_MOD_STx0       14
        EV5_CMD__READ_MS_MOD_STx1       15
```

See John Edmondson's memo on write timeliness, which is append to end of this note.

10 EV5 RESPONSES TO SYSTEM REQUESTS

The following table shows the responses that EV5 will generate for each system request.

	BCACHE	SCACHE	RESPONSE
FLUSH	None	Sc_Miss	Noack
	None	Sc_Hit, not dirty	Noack
	None	Sc_Hit, dirty	Ack_Sc
	None	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Miss	Sc_Miss	Noack
	Bc_Miss	Sc_Hit, not dirty	Noack
	Bc_Miss/Hit	Sc_Hit, dirty	Ack_Sc
	Bc_Miss/Hit	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Hit, not dirty	Sc_Miss/Hit, not dirty	Noack
	Bc_Hit, dirty	Sc_Miss/Hit, not dirty	Ack_Bc
INVALIDATE	None	Sc_Miss	Noack
SET_SHARED	None	Sc_Hit	Ack_Sc
	Bc_Miss/Hit	Sc_Miss/Hit	Ack_Bc
READ	None	Sc_Miss	Noack
	None	Sc_Hit	Ack_Sc
	None	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Miss	Sc_Miss	Noack
	Bc_Miss/Hit	Sc_Hit	Ack_Sc
	Bc_Miss/Hit	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Hit	Sc_Miss	Ack_Bc
READ_DIRTY	None	Sc_Miss	Noack
RD_DIRTY_INV	None	Sc_Hit, not dirty	Noack
	None	Sc_Hit, dirty	Ack_Sc
	None	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Hit, dirty	Sc_Hit, dirty	Ack_Sc
	Bc_Hit, dirty	hit in Sc_Victim_Buffer	Ack_Sc
	Bc_Hit, dirty	Sc_Miss/Hit, not dirty	Ack_Bc

10.1 Scache Tag/Data Par_err - Any Command

Should this occur while Cbox is processing an incoming system command, it will be completed as normal and the usual response transmitted. However, the parity error will eventually be logged in the SC_ADDR and SC_STAT Cbox ipr's, and a machine check is generated by Ibox.

10.2 System_Address_Command Par_err - Any Command

Same as above, except the parity error is logged in the EI_ADDR and EI_STAT ipr's, and Cbox will terminate the incoming system

command and respond with NOACK.

11 SENDING SYSTEM REQUEST TO EV5

The rules for sending a request from the system to EV5 have changed some. The new rules follow this model:

```
if (init)
      count = 0;

if (cmd && (count < 2)) {
      count ++;

      /* ADDR_BUS_REQ can already be asserted */
      send(cmd)
}

if (ev5_res == ack/scache) {
      if (cmd == read_dirty | cmd == read_dirty_inv
          cmd == read       | cmd == flush) {

            /* first, receive all the data */

            count --;
      }
      else {
            count --;
      }
}

if (ev5_res == ack/bcache |
    ev5_res == noack) {
      count --;
}
```

12 SEQUENCING CPU AND SYSTEM REQUEST THROUGH THE BCACHE

This section goes over the rules for determining the order in which the BIU processes EV5 and System requests. In general the order of processing is determined by the system using EV5_CMD, IDLE_BC, and the FILL.

1. If IDLE_BC is not asserted and there are no valid requests in the system command buffer, then EV5 is free to do any CPU request.

2. If a FILL is pending, EV5 will only produce another read miss, with a possible Bcache victim. It will not attempt any other command.

3. The assertion of IDLE_BC, or the sending of a valid non-NOP system command to EV5 will cause the BIU to idle. If the BIU has a command loaded in the pad ring, it will remove the command and replace it with a NOP. The state of the command/address bus is unpredictable until the idle condition goes away.

4. The idle condition ends when EV5 recieves a de-asserted IDLE_BC, and EV5 has responded to all the system commands that were sent.

5. The System must not assert CACK during the idle condition.

6. There is one exception to rules 3, 4, and 5. If IDLE_BC or a system command arrives while EV5 is reading the Bcache, and that read turns into a miss, and it does not produce a victim, then EV5 will load the miss into the pad ring. The system may CACK this read miss request at any time.

7. If CACK is asserted at the same time as IDLE_BC or a valid system request, CACK wins and the command is taken by the system. CACK should not be asserted if IDLE_BC has been asserted or a valid System command is underway.

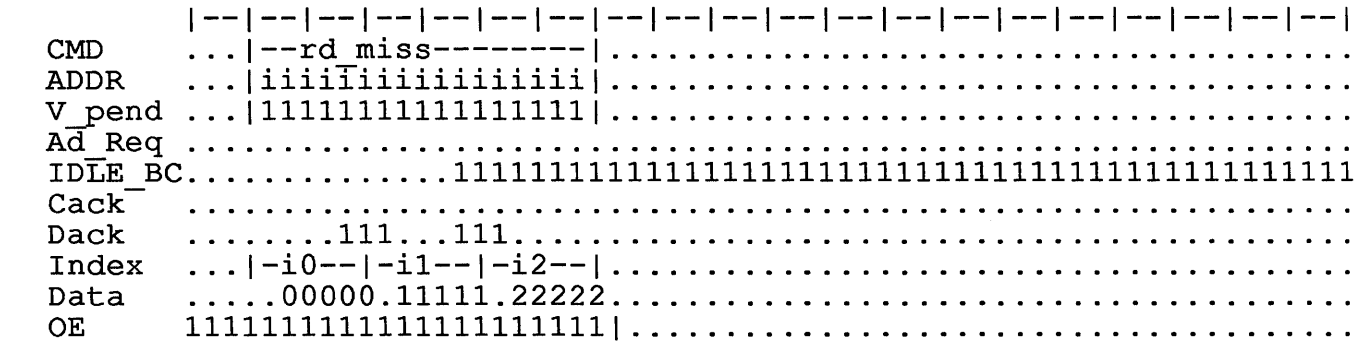
8. A read miss with a victim is treated as a pair. The order, read miss then victim or victim then read miss, is programmable. Either way, if the first command is CACKed, then both commands must be CACKed and all the data DACKed, before EV5 will respond to any other request.

9. The CACK for a WRITE_BLOCK or BCACHE_VICTIM must be received by EV5 with or before the last DACK of the data. For WRITE_BLOCK and BCACHE_VICTIM, it is possible to DACK all but the last data, and then decide to do something else.

10. The CACK for a READ_MISS must arrive with or before the last DACK for the requested fill.

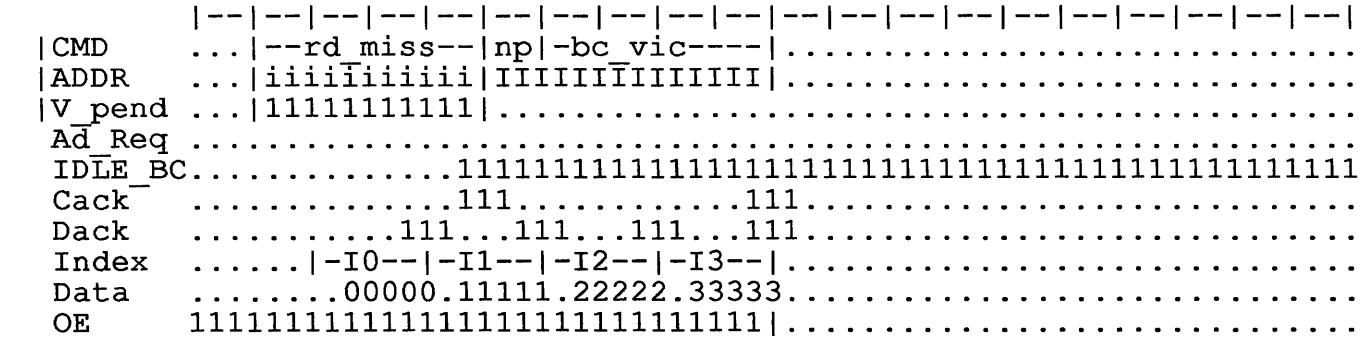
12.1 Read Miss With Victim Example

In this example EV5 asserts a read miss with a victim. The system Dacks two datas from the Bcache and then asserts IDLE_BC. This causes EV5 to remove the read miss with victim pending. EV5 will reassert the read miss and victim, if needed, at a later time.



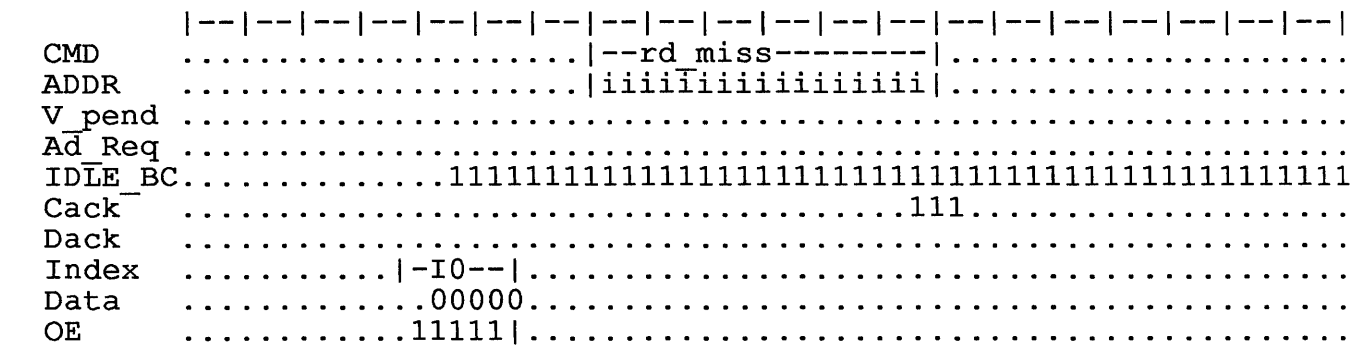
12.2 IDLE_BC And CACK Race Example

In this example, IDLE_BC and Cack are asserted in the same Sysclk. This means the the System will take the read miss and victim before doing anything else. Note that the last Dack for the victim is in the same cycle as the Cack for the victim command, meeting the requirement that the Cack arrive before or with the last Dack.



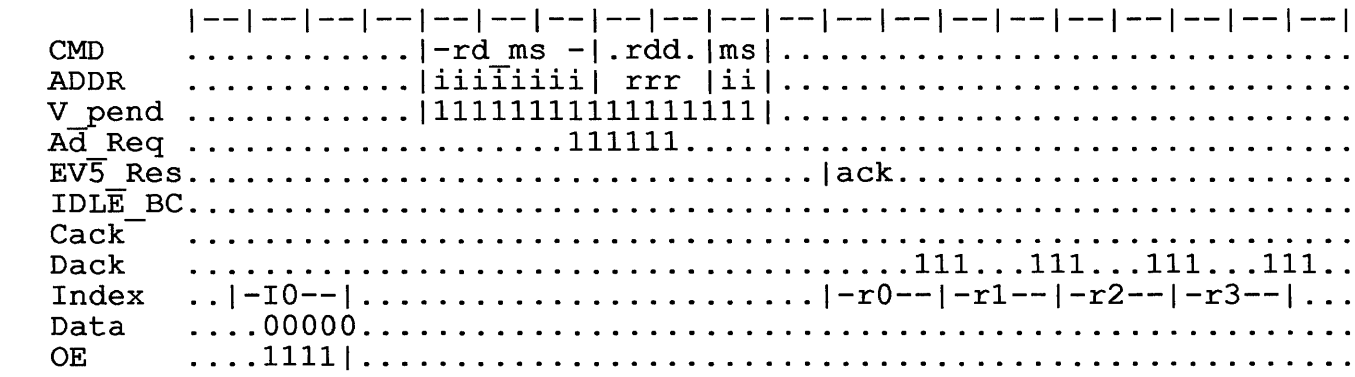
12.3 Read Miss With IDLE_BC Asserted Example

In this example, EV5 has started a Bcache read that misses. IDLE_BC is asserted, but no victim was created, so the read miss request is loaded into the pad ring. The system then takes the request.



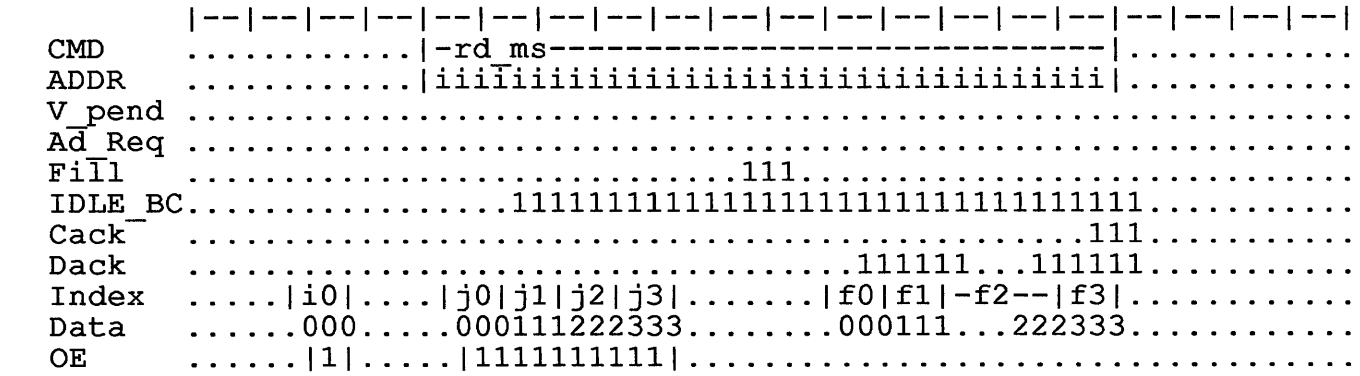
12.4 Read Miss With Victim Abort Example

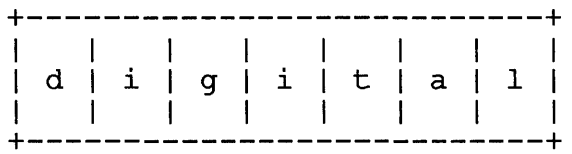
In this example, EV5 produces a read miss with a victim and is waiting for the system to take it when the system takes the bus and requests a read dirty. EV5 drives the miss request for one more cycle after it gets the bus back and then removes the request. EV5 then responds to the read dirty and drive the index to read the Bcache. Not shown in the picture is EV5 restarting the Bcache read, requesting the read miss with victim. If the victim block was invalidated by the system request, EV5 will produce a clean read miss.



12.5 Bcahe Hit Under Read Miss Example

In this example, EV5 produces a read miss and requests a fill from the system. A Bcache hit to index j take place while we wait for the fill. The system then return the requested data in two bursts, asserting CACK at the same time as the last Dack.


CMD
ADDR
V_pend
Ad_Req
Fill
IDLE_BC
Cack
Dack
Index
Data
OE
-rd_ms
i0
j0
j1
j2
j3
f0
f1
-f2--
f3
000
000111222333
000111...222333
1
1111111111
111
111111...111111



TO: Distribution

DATE: 13-JAN-1993
FROM: John Edmondson
DEPT: SEG/HPC
EXT: 225-6249
LOC/MS: HLO2-3/D11
ENET: ROCK::Edmondson
or evsrv2::jhe

SUBJECT: Proposed Solution to Write Timeliness Problem

EV5 is prepared to make a pin-bus command encoding change, provided the Turbo-Laser team confirms they want it. The change is intended to solve the problems that arise from the difference between the EV5 cache-coherence protocol and the Turbo-Laser bus cache-coherence protocol. The problems are write timeliness and EV5 hanging due to write operation livelock. I describe in detail the proposed EV5 pin-bus changes and the proposed external logic which utilizes this change to solve the stated problems.

Description of the Protocol Bug

This is a description of the protocol bug which results from the difference between the EV5 cache-coherence protocol and the Turbo-Laser bus cache-coherence protocol. The bug occurs when one processor (CPU A) repeatedly writes to block X and another processor (CPU B) has to write block X too. CPU B's write misses in its caches and causes a READ-MISS-MOD. That READ-MISS-MOD hits dirty in CPU A's cache and causes the block to become SHARED in CPU A's cache. After CPU B's cache receives the fill of block X, CPU B tries to send a WRITE-BLOCK to its bus interface. Meanwhile, a subsequent write by CPU A is broadcast on the bus (CPU A wins arb since CPU B just used the bus and takes a certain amount of time to respond to the fill and produce the WRITE-BLOCK command). Since CPU A's write won arb on the bus, CPU B's bus interface has to force CPU B off its command bus (via ADDR_BUS_REQ_H) and sent CPU B an invalidate of block X. This means that CPU B will have to abort the pending WRITE-BLOCK to block X and start over by sending a READ-MISS-MOD to block X. Meanwhile, CPU A converted block X to PRIVATE-CLEAN and responds to CPU B's READ-MISS-MOD by making block X shared in its cache. This begins the process over again. Notice that CPU A completes new writes again and again while CPU B is never

able to complete even one write.

**** Digital Confidential ****

The EV5 Change

The proposed change in the pin-bus command encodings is:

The encoding for WRITE BLOCK and WRITE BLOCK LOCK from 14 and 15, respectively, to 6 and 7, respectively. (This undoes an earlier change which I think was proposed by the Turbo-Laser team. The reason should be clear soon.)

Two new commands are added: READ-MISS-STC0 and READ-MISS-STC1. The encodings for these are 14 and 15 (decimal). These commands are used exclusively for fills due to store conditional execution. The following table shows the four relevant commands and their binary encodings.

CMD<3:0>	Command	Optional	Comments
1010	READ MISS MOD0	No	Request for data, modify in
1011	READ MISS MOD1	No	Request for data, modify in
.			
.			
1110	READ MISS STC0	No	Request for data, STx_C spe
1111	READ MISS STC1	No	Request for data, STx_C spe

The choice of encodings allows the Cbox to logic-or CMD<2> with the status indicating STx_C, making this change feasible.

EV5 will use READ MISSn for read misses without modify intent, READ MISS MODn for read misses to complete an ordinary write, and READ MISS STCn for read misses to complete a store conditional. EV5's behavior is otherwise unchanged from before (i.e., READ MISS STCn should lead to a FILLn and so on).

The Proposed External Logic

To solve both the write timeliness problem and the problem of EV5 hanging, I propose Turbo-Laser implement the following mechanism. It uses LOCKOUT assertion on the Turbo-Laser bus to guarantee a write completes. This proposed mechanism detects when a write is not likely to complete without LOCKOUT assertion and asserts LOCKOUT. Then it correctly detects completion of the write and deasserts LOCKOUT. There is no non-error case in which LOCKOUT is asserted for a write that has been aborted by EV5, so a short timeout on LOCKOUT is not necessary. I'll discuss errors later.

For completeness, I assume LOCKOUT is a wired-or signal on the Turbo-Laser bus. When it is asserted, each CPU module that is not asserting it must not attempt to arb for the bus. As a result the one or more CPUs asserting LOCKOUT get preferential treatment on the bus until they stop asserting lockout. If the CPUs use this to complete pre-existing write operations and not for writes which begin after LOCKOUT assertion, then LOCKOUT deassertion is guaranteed in a bounded amount of time.

Many of the ideas included in this proposal were given to me by Dennis Foley.

Note that this mechanism is required to solve both the write timeliness problem and the write livelock problem. A simpler solution exists in which timeliness is not solved but EV5 will never hang due to livelock. I'll discuss this later.

The external interface implements two address CAMs with associated logic (CAM0 and CAM1). The CAMs load from and compare to address bits 12:6 of the address sent by EV5 with memory access commands. Each CAM has an associated valid bit, a fill number bit, a fill pending bit, and an associated three-bit counter. The CAMs are loaded and validated as a side effect of some READ-MOD-MISSn commands, and are invalidated as a side effect of other READ-MISS-MOD commands or WRITE-BLOCK commands. Here are the details:

1. At reset, invalidate the CAMs and clear their counters.

2. If both CAMs are invalid, load CAM0 with each READ-MISS-MOD0 and CAM1 with each READ-MISS-MOD1, but only if those commands are CACKed. Set the fill pending bit and record the fill number.

3. If exactly one CAM is valid, load the other on every READ-MISS-MODn that doesn't hit in the valid CAM, recording the fill number and setting the fill pending bit as above.

4. If both CAMs are valid, every READ-MISS-MODn should hit one or the other. Otherwise it is an error.

5. Validate a CAM on the corresponding fill (fill pending set and fill number matches), if and only if the fill is SHARED. Clear the fill pending bit on every corresponding fill.

6. If a READ-MISS-MODn is issued which hits in a valid CAM (valid and matches in address bits <12:6>), increment the associated counter (except don't increment the counter if LOCKOUT is asserted on the bus). Record the new fill number

and set the fill pending bit. This all occurs only if the command is CACKed.

7. If a WRITE-BLOCK is issued that hits in a valid CAM, invalidate the CAM and clear the counter, but only if the WRITE-BLOCK is CACKed.

8. If a fill corresponding to a valid CAM occurs (fill pending and fill number matches) and the fill is NOT-SHARED, invalidate the CAM. (Assumes NOT-SHARED => DIRTY in this case.)

9. If a valid CAM's counter reaches 7, assert LOCKOUT. From here on, the counter shouldn't increment, so LOCKOUT should stay asserted until the CAM is invalidated (meaning the corresponding write operation finished).

**** Digital Confidential ****

10. If a new write operation begins while the CPU is asserting LOCKOUT, the counter must not be incremented. Otherwise there is no guarantee this CPU ever stops assertin

LOCKOUT.

Necessary assumptions

This proposal is based on certain assumptions about EV5. These a

1. EV5 may only process two writes that involve the Bcache or external environment. In particular, if EV5 is working on two writes which require READ-MISS-MODn, READ-MISS-STCn, SET-DIRTYs, or WRITE-BLOCK to complete, EV5 will never begin processing a third write that requires any of these things until one of the current writes is completed. Completion means the write hit dirty, not-shared in the Scache or did a WRITE-BLOCK that was CACKed.

2. EV5 never processes more than one write at a time with a particular value of address bits <12:6>. I.e., if one write is being processed and another is begun which matches in address<12:6>, the second will be retried internally and will never lead to READ-MISS-MODn, READ-MISS-STCn, SET-DIRTYs, or WRITE-BLOCK.

3. After a fill for READ-MISS-MODn which fills not-shared, dirty, EV5 will guarantee to complete the corresponding write (given also that fills are always in the same order as the commands were issued by EV5 on the pin-bus). This guarantee specifically covers the case in which a fill is closely followed by an INVALIDATE to the same block.

4. EV5 will complete a STx_C after a fill for READ-MIS_STCn regardless of the fill state (shared or not-shared) if the lock flag is not set. This guarantee specifically covers the case in which a fill is closely followed by an INVALIDATE to the same block. (If the fill is shared, there is no guarantee, but only an invalidate can prevent completion and this will reset the lock flag.).

An assumption about Turbo-Laser/TLEP required for this proposal is that fills to EV5 for READ-MISS-MODn will either be shared, not-dirty or not-shared, dirty. Not-shared, not-dirty fills require rethinking detail item 8 to include a SET DIRTY that completes.

One rule that EV5 requires is that a READ-DIRTY, INVALIDATE, or SET-SHARED for the same block just filled to EV5 must not follow too closely after the fill for a READ-MISS-MODn that filled dirty, not-shared. This is needed to guarantee that EV5 will complete one write. We are currently evaluating a rule for the number of CPU cycles after the last Sysclock DACK cycle before any of these system commands for the same block are allowed to be sent.

Errors

The only error I can anticipate is that a fill error which caused the environment to use CFAIL*~CACK to reset EV5 will cause EV5 to "drop" all writes. This even should invalidate the CAMs and clear their counters.

Other errors can be detected. Perhaps machine check interrupt is the best way to handle these.

A long timeout on LOCKOUT may be useful, but a short timeout may prevent LOCKOUT from accomplishing its purpose.

An Alternative

An alternative implementation with exactly one CAM will (I think) eventually guarantee EV5 finishes all pending writes provided EV5 won't start new writes past a certain point (i.e., the WMB effect). This scheme doesn't solve timeliness in unusual cases (cases with two antagonist CPUs writing two different blocks continuously). PALcode could insert a WMB in interrupt flows, so every timer interrupt or other interrupt could force "stuck" writes to complete. A waiver from the ALPHA Architecture Board would be needed. A simpler, cheaper scheme covering all reasonable circumstances coupled with eventual guarantees via timer interrupt seems quite reasonable to me.