digital™

DECChip™ 21064–AA RISC Microprocessor Preliminary Data Sheet

Digital Equipment Corporation
Maynard, Massachusetts

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

Chapter 3 Privileged Architecture Library Code

Chapter 4 External Interface

Chapter 5 DC Characteristics

Chapter 6 AC Characteristics

Chapter 7 Package Information

Chapter 8 Pinout

Figures

Tables

Preface

This specification covers the following topics:

- Introduct to terminology and conventions
- An overview of the micro-architecture
- 21064 implimentation of the Privileged Architecture Library Code
- Chip external interface
- Electrical characteristics
- Pinout and Packaging

Chapter 1

Introduction

1.1 Scope

This document describes the 21064-AA RISC CPU microprocessor. The 21064-AA is the first of a family of microprocessors that implement the Digital Equipment Corporation Alpha architecture. This document describes features specific to the 21064 implementation of the architecture. It does not describe the complete details of the implementation nor the Alpha architecture.

1.2 21064 Features Overview

The 21064 microprocessor is a CMOS-4 (.75 micron) super-scalar super-pipelined implementation of the Alpha architecture. It will become the basis of the first family of Alpha products. The 21064 is designed to meet the requirements of a wide variety of systems, ranging from uniprocessor workstations to midrange multiprocessors. To achieve this goal, the CPU enforces as little policy as possible, e.g. it does not enforce a particular cache coherence scheme. The 21064 attempts to spread fairly the design compromises over the range of user requirements. The design balances the cost goals of the low-end workstation with performance goals of the mid-range multiprocessors.

Features Overview:

- 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. 21064 implements 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. The translation buffer can be used with alternative PALcode to implement a different page table structure.
  - On-chip 12-entry I-stream TB with 8 entries for 8K-byte pages and 4 entries for 4M byte pages.

- — 32-entry D-stream TB with each entry able to map 8K, 64K, 512K, or 4M byte pages.

- World class performance. At its nominal frequency the 21064 achieves a 6.6ns cycle time. Cycle times of 5ns will also be possible.
- Low average cycles per instructions (CPI). The 21064 CPU can issue two Alpha instructions in a single cycle, thereby minimizing the average CPI. Branch history tables are also used to minimize the branch latency, further reducing the average CPI.
- On-chip high-throughput floating point unit, capable of executing both DEC and IEEE floating point data types.
- On-chip 8K-byte data cache and an 8K-byte physical instruction cache with ASN support.
- On-chip write buffer with four 32-byte entries.
- On-chip performance counters to measure and analyze CPU and system performance.
- Bus interface unit, which contains logic to directly access external cache RAMs without CPU module action. The size and access time of the external cache is programmable.
- 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 21064. and a pair of system clocks for use by the CPU module.
- The 21064 is packaged in a 431 pin (24 x 24, 100 mil pin pitch) PGA package. The heat sink is separable and application specific.

1.3 Terminology and Conventions

1.3.1 Definitions

This specification is the description of the 21064-AA RISC microprocessor. The full designation will be used where appropriate. 21064, CPU and chip will also be used when referring to the 21064-AA.

1.3.2 Numbering

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

1.3.3 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 a state a state from which there is no transition to a normal state in which the machine executes instructions and not be halted. Only privileged software (that is, software running in kernel mode) is allowed to trigger UNDEFINED operations.

1.3.4 Ranges And Extents

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

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

1.3.5 Must be Zero (MBZ)

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

1.3.6 Should be Zero (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.

1.3.7 Read As Zero (RAZ)

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

1.3.8 Ignore (IGN)

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

1.3.9 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 power-up. If the initialized value is not present, the field is not initialized at power-up.

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 three 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	Denotes a register bit(s) that is read as a 0.
IGN	Denotes a register bit(s) that is ignored on write and UNPREDICTABLE when read if not otherwise specified.

Chapter 2

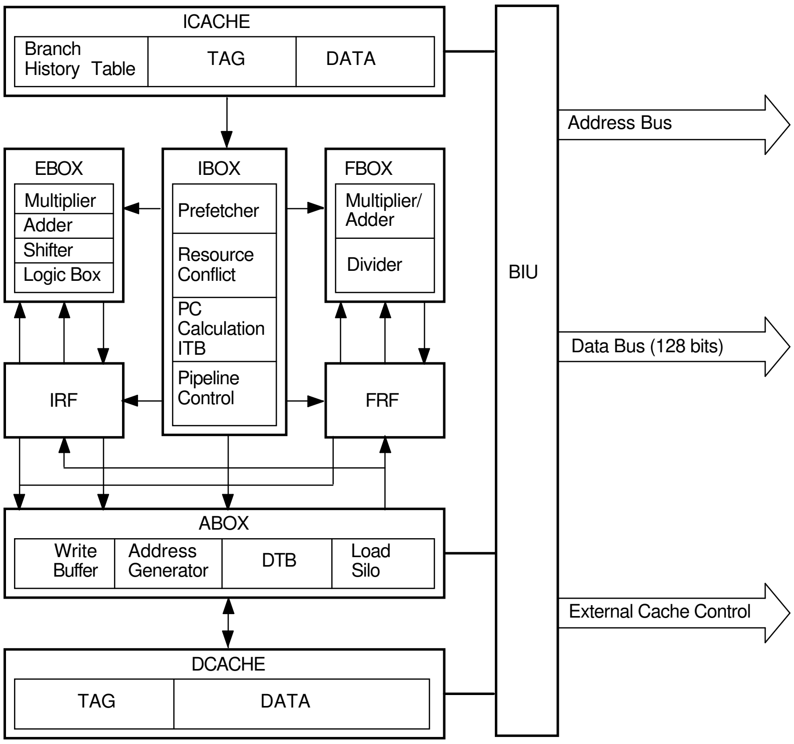
Microarchitecture

2.1 Introduction

This chapter gives a view of the 21064 micro-architecture for programmers and system designers. It is not intended to be a detailed hardware description of the chip. This document first describes the hardware with only minimal forward references to the pipeline and then presents the pipeline. The 21064 CPU can issue two instructions in a single cycle - the scheduling and dual issue rules are defined at the end of the chapter.

It is important to realize that the Alpha architecture is the combination of the 21064 microarchitecture and PALcode. Many hardware design decisions were based on specific PAL functionality. These PAL assumptions and restrictions are detailed in the next chapter. It is important to keep in mind that if a certain piece of hardware appears to be "architecturally incomplete", the missing functionality is implemented in PALcode.

Figure 2–1: Block Diagram



2.2 Overview

The 21064 microprocessor consists of three independent execution units: integer execution unit (Ebox), floating point unit (Fbox), and the address generation, memory management, write buffer and bus interface unit (Abox). Each unit can accept at most one instruction per cycle, however if code is properly scheduled, the 21064 can issue two instructions to two independent units in a single cycle. A fourth unit, the Ibox, is the central control unit. It issues instructions, maintains the pipeline and performs all of the PC calculations. The 21064 also has on-chip instruction and data caches (Icache and Dcache).

2.3 Instruction Box - Ibox

The primary function of the Ibox is to issue instructions to the Ebox, Abox and Fbox. In order to provide those instructions, the Ibox also contains the prefetcher, PC pipeline, ITB, abort logic, register conflict or dirty logic, and exception logic. The Ibox decodes two instructions in parallel and checks that the required resources are available for both instructions. If resources are available and dual issue is possible then both instructions may be issued. The section on Dual Issue Rules details which instructions can be dual issued. If the resources are available for only the first instruction or the instructions cannot be dual issued then the Ibox issues only the first instruction. The Ibox does NOT issue instructions out of order, even if the resources are available for the second instruction and not for the first instruction. The Ibox does not issue instructions until the resources for the first instruction become available. If only the first of a pair of instructions issues, the Ibox does not advance another instruction to attempt to dual issue again. Dual issue is only attempted on aligned quadword pairs.

2.3.1 Branch Prediction Logic

The Ibox contains the branch prediction logic. The 21064 offers a choice of branch prediction strategies selectable through the ICCSR IPR. The Icache records the outcome of branch instructions in a single history bit provided for each instruction location in the cache. This information can be used as the prediction for the next execution of the branch instruction. The prediction for the first execution of a branch instruction is based on the sign of the displacement field within the branch instruction itself. If the sign bit is negative, conditional branches are predicted to be taken. If the sign is positive, conditional branches are predicted to be not taken. Alternatively, if the history table is disabled, branches can be predicted based on the sign of the displacement field at all times.

The 21064 provides a four entry subroutine return stack which is controlled by the hint bits in the BSR, HW_REI, and jump to subroutine instructions (JMP, JSR, RET, or JSR_ COROUTINE). It also provide a means of disabling all branch prediction hardware.

2.3.2 Instruction Translation Buffer - ITB

The Ibox contains an 8-entry, fully associative translation buffer which caches recently used instruction-stream page table entries for 8Kbyte pages, and a four entry fully associative translation buffer which supports the largest granularity hint option (512*8Kbyte pages) as described in the Alpha Architecture Handbook. Both translation buffers use a not-last-used replacement algorithm. They are hereafter referred to as the small-page and large-page ITBs, respectively.

In addition, an extension is provided that is referred to as the super page, which can be enabled by the MAP bit in the ICCSR IPR. Super page mappings provide one-to-one virtual PC<33:13> to physical PC<33:13> translation when virtual address bits <42:41> = 2. This function essentially maps the entire physical address space multiple times over to one quadrant of the virtual address space defined by <42:41> = 2. When translating through the super page, the PTE[ASM] bit used in the Icache is always set. Access to the super page mapping is only allowed while executing in kernel mode.

The ITBs are filled and maintained by PALcode. The operation system via PALcode is responsible for insuring that virtual addresses can only be mapped through a single, large page, small page or super page ITB entry at the same time.

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

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

2.3.3 Interrupt Logic

The 21064 chip supports three sources of interrupts; hardware, software and asynchronous system trap (AST). There are six 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 internal IPR (ASTRR). All interrupts are independently maskable via on-chip enable registers to support a software controlled mechanism for prioritization. In addition, AST interrupts are qualified by the current processor mode and the current state of SIER[2].

By providing distinct enable bits for each independent interrupt source, a software controlled interrupt priority scheme can be implemented with maximum flexibility. For example, a six level interrupt priority scheme can be supported for the six hardware interrupt request pins by defining a distinct state of the corresponding hardware interrupt enable register for each IPL. The current interrupt priority is determined by the state of the interrupt enable register. The lowest interrupt priority level is produced by enabling all 6 interrupts, e.g bits 6-1. The next is produced by enabling bits 6-2 and so on to the highest interrupt priority level which is produced by enabling only bit 6 and disabling bits 5 through 1. When all interrupt enable bits are cleared, the processor can not be interrupted from the hardware interrupt request register. Each state, 6-1,6-2,6-3,6-4,6-5,6 represents an individual interrupt priority level (IPL). If these states are the only states allowed in the interrupt enable register, a six level hardware interrupt priority scheme can be controlled entirely by software.

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

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

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

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 and all outstanding load instructions are completed. The restart address is saved in the Exception Address IPR (EXC_ADDR) and the processor enters PALmode. The cause of the interrupt may be determined by examining the state of any of the interrupt request registers.

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

2.3.4 Performance Counters

The 21064 chip contains a performance recording feature. The implementation of this feature provides a mechanism to count various hardware events and cause an interrupt upon counter overflow. Interrupts are triggered six cycles after the event, and therefore, the exception PC may not reflect the exact instruction causing counter overflow. Two counters are provided to allow accurate comparison of two variables under a potentially non-repeatable experimental condition. Counter inputs include issues, non-issues, total cycles, pipe dry, pipe freeze, mispredicts and cache misses as well as counts for various instruction classifications. In addition, one chip pin input to each counter is provided to measure external events at a rate determined by the selected system clock speed.

2.4 Exectution Box - Ebox

The Ebox contains the 64-bit integer execution datapath: adder, logic box, barrel shifter, byte zapper, bypassers and integer multiplier. The integer multiplier retires four bits per cycle. The Ebox also contains the 32-entry by 64-bit integer register file. This register file has four read ports and two write ports which allow the sourcing (sinking) of operands (results) to both the integer execution datapath and the Abox.

2.5 Address Box - Abox

The Abox contains six major sections: address translation datapath, load silo, write buffer, Dcache interface, IPRs and the external Bus Interface Unit (BIU). The address translation datapath has a displacement adder which generates the effective virtual address for load and store instructions, and a pair of translation buffers which generate the corresponding physical address.

2.5.1 Data Translation Buffer - DTB

The 21064 contains a 32-entry, fully associative translation buffer which caches recently used data-stream page table entries.and supports all four variants of the granularity hint option as described in the Alpha Architecture Handbook.

The 21064 provides an extension referred to as the super page, which can be enabled via ABOX_CTL<5:4>. Super page mappings provide virtual to physical address translation for two regions of the virtual address space. The first enables super page mapping when virtual address bits <42:41> = 2. In this mode, the entire physical address space is mapped multiple times over to one quadrant of the virtual address space defined by VA<42:41> = 2. The second super page mode maps a 30-bit region of the total physical address space defined by PA<33:30> = 0 into a single corresponding region of virtual space defined by VA<42:30> = 1FFE Hex. Super page translation is only allowed in kernel mode.

The operating system, via PALcode, is responsible for insuring that translation buffer entries, including super page regions, do not map overlapping virtual address regions at the same time.

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

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

The DTBs are filled and maintained by PALcode. The chapter on PALcode details the DTB miss flow. Note that the DTBs can be filled in kernel mode by setting the HWE bit in the ICCSR IPR.

2.5.2 Bus Interface Unit - BIU

The BIU controls the interface to the 21064 pin bus. It responds to three classes of CPU generated requests: Dcache fills, Icache fills and write buffer-sourced commands. The BIU resolves simultaneous internal requests using a fixed priority scheme in which Dcache fill requests are given highest priority, followed by Icache fill requests. Write buffer requests have the lowest priority. The external interface chapter of this specification describes the 21064 pin bus. The BIU contains logic to directly access an external cache to service internal cache fill requests and writes from the write buffer. The BIU services reads and writes which do not hit in the external cache with help from external logic.

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

2.5.3 Load Silos

The Abox contains a fully folded memory reference pipeline which may accept a new load or store instruction every cycle until a Dcache fill is required. Since the Dcache lines are only allocated on load misses, the Abox may accept a new instruction every cycle until a load miss occurs. When a load miss occurs the Ibox stops issuing all instructions that use the load port of the register file or are otherwise handled by the Abox. This includes LDx, STx, HW_MTPR, HW_MFPR, FETCH, FETCH_M, RPCC, RS, RC, MB, and all memory format branch instructions, JMP, JSR, JSR_COROUTINE, and RET.

A JSR with a destination of R31 may be issued.

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

- Loads which hit the Dcache are allowed to complete - hit under miss.

- Load misses are placed in a silo and replayed in order after the first load miss completes.
- Store instructions are presented to the Dcache at their normal time with respect to the pipeline. They are siloed and presented to the write buffer in order with respect to load misses.

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

2.5.4 Write Buffer

The Abox contains a write buffer for two purposes:

1. To minimize the number of CPU stall cycles by providing a high bandwidth (but finite) resource for receiving store data. This is required since the 21064 can generate store data at the peak rate of one quadword every CPU cycle which is greater than the rate at which the external cache subsystem can accept the data.
2. To attempt to aggregate store data into aligned 32-byte cache blocks for the purpose of maximizing the rate at which data may be written from the 21064 into the external cache.

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

Each write buffer entry contains a CAM for holding physical address bits <33:5>, four quadwords of data, eight longword mask bits which indicate which of the associated eight longwords in the entry contain valid data, and miscellaneous control bits.

To facilitate the discussion, the following two states are defined: invalid and valid. A write buffer entry is invalid if it does not contain one of the above-listed write buffer-directed commands. A write buffer entry is valid if it contains one of the above-listed write buffer-directed commands.

The write buffer contains two pointers: the head pointer and the tail pointer. The head pointer points to the valid write buffer entry which has been valid the longest period of time. The tail pointer points to the invalid write buffer entry slot which will next be validated. If the write buffer is completely full (empty) the head and tail pointers point to the same valid (invalid) entry.

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. If the address is in the same aligned 32-byte block as an address in a valid write buffer entry which also contains a store then the new store data is merged into that entry, and the entry's longword mask bits are updated. If no matching address is found in the write buffer, then the store data is written into the entry designated by the tail pointer, the entry is validated, and the tail pointer is incremented to the next entry. Note this scheme does not maintain write-ordering.

The write buffer contains four entries and employs a complicated flow control mechanism which allows its entries to be efficiently used. The Ibox issues store instructions irrespective of whether the write buffer is full. If a store instruction enters pipe stage [6] of the Abox and the write buffer is full, the Ibox is forced to stop issuing both loads and stores by the same mechanism which is used for handling load misses. In effect, the store instruction gets treated as if it were a load miss. Any valid instructions in pipe stages [4] or [5] get handled exactly as if they had followed a load miss - loads which hit the Dcache are allowed to complete, stores are presented to the Dcache, placed into the Abox silo and and presented to the write buffer in order with respect to other siloed instructions. The Abox silo control logic insures that no stores are lost when the write buffer is full by retrying silo'ed stores until they are accepted by the write buffer.

The write buffer attempts to send its head entry off-chip by requesting the BIU when one of the following conditions are met:

1. The write buffer contains at least two valid entries.
2. The write buffer contains one valid entry and at least 256 CPU cycles have elapsed since the execution of the last write buffer-directed instruction.
3. The write buffer contains an MB instruction.
4. The write buffer contains a STQ/C or STL/C instruction.
5. A load miss is pending which requires the write buffer to be flushed before an external read is launched to service the load miss.

When the write buffer is requesting the BIU no stores are allowed to merge into the write buffer's head entry.

2.6 Fbox

The 21064 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 except round to +/- infinity which can be provided in PALcode. DEC floating point datatypes F and G are fully supported with limited support for D floating format. The Fbox contains a 32-entry by 64-bit floating point register file and a user accessible control register, FPCR. The FPCR contains round mode controls, trap enables, and exception flag information. The Fbox can accept an instruction every cycle, with the exception of floating point divide instructions. The latency for data dependent, non divide instructions is six cycles. Bypass mechanisms 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 2.9.

For divide instructions, the Fbox does not compute the inexact flag. Consequently, the INE exception flag in the FPCR register is never updated for any DIV instructions. This is a known incompatibility.

2.7 Cache Organization

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

2.7.1 Data Cache

The 21064 Dcache contains 8K-bytes. It is a write-through, direct mapped, read-allocate physical cache and has 32-byte blocks. System components may keep the Dcache coherent with memory by using the invalidate bus described in the pin bus section of this specification.

2.7.2 Instruction Cache

The 21064 Icache is an 8Kbyte physical direct-mapped cache. Icache blocks, or lines, contain 32-bytes of instruction stream data with associated tag as well as a six-bit ASN field, a one-bit ASM field and an eight-bit branch history field per block. It does not contain hardware for maintaining coherency with memory and is unaffected by the invalidate bus.

The chip also contains a single-entry Icache stream buffer which together with its supporting logic reduces the performance penalty due to Icache misses incurred during in-line instruction processing. The stream buffer physically consists of latches for one Icache block's data and tag bits which are adjacent to the fill-side of the cache array, and a comparator, 13-bit incrementer and associated datapath hardware and control in the Abox.

When an Icache miss occurs, the Ibox sends an Icache fill request to the Abox, which simultaneously requests the BIU and checks the stream buffer for the requested block. If the block is present in the stream buffer the Abox aborts the original Icache fill request, writes the requested block into the Icache and launches a prefetch request to the BIU for the next consecutive Icache block. The Ibox does not interact with the stream buffer; from the Ibox's perspective Icache misses which hit the stream buffer are the same as any other Icache miss except that the Icache fill finishes sooner.

When an Icache miss also misses the stream buffer the Abox launches a request for the required fill block and subsequently launches a prefetch request for the next consecutive fill block, thus getting the stream buffer started down the next I-stream path. Stream buffer prefetch requests never cross physical page boundaries, but instead wrap around to first block of the current page.

2.8 Pipeline Organization

The 21064 has a seven stage pipeline for integer operate and memory reference instructions. Floating point operate instructions progress through a ten stage pipeline. The Ibox maintains state for all pipeline stages to track outstanding register writes, and determine Icache hit /miss. The pipeline diagrams below show the Ebox, Ibox, Abox and Fbox pipelines. The first four cycles are executed in the Ibox and the last stages are box specific. There are bypassers in all of the boxes that allow the results of one instruction to be used as operands of a following instruction without having to be written to the register file. The following section describes the pipeline scheduling rules.

Figure 2–2: Integer Operate Pipeline

```
    IF       SWAP       I0       I1       A1        A2       WR
|--------|--------|--------|--------|--------|--------|--------|
    [0]      [1]        [2]      [3]      [4]       [5]      [6]
```

- IF - Instruction Fetch.
- SWAP - Swap Dual Issue Instruction /Branch Prediction.
- I0 - Decode.
- I1 - Register file(s) access / Issue check.
- A1 - Computation cycle 1 / Ibox computes new PC.
- A2 - Computation cycle 2 / ITB look-up
- WR - Integer register file write / Icache Hit/Miss

Figure 2–3: Memory Reference Pipeline

```
    IF       SWAP     I0         I1       AC       TB        HM
|--------|--------|--------|--------|--------|--------|--------|
    [0]      [1]      [2]        [3]      [4]      [5]       [6]
```

- AC - Abox calculates the effective D-stream address.
- TB - DTB look-up.
- HM - Dcache Hit/Miss and load data register file write

Figure 2–4: Floating Point Operate Pipeline

```
   IF    SWAP    I0     I1     F1     F2     F3     F4     F5    FWR
|------|------|------|------|------|------|------|------|------|------|
   [0]    [1]    [2]    [3]    [4]    [5]    [6]    [7]    [8]   [9]
```

- F1-F5 - Floating point calculate pipeline

- FWR - Floating point register file write

The 21064 integer pipeline divides instruction processing into four static and three dynamic stages of execution. The 21064 floating point pipeline maintains the first four static stages and adds six dynamic stages of execution. The first four stages consist of the instruction fetch, swap, decode and issue 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. Pipeline stalls are also referred to as pipeline freezes. A pipeline freeze may occur while zero instructions issue, or while one instruction of a pair issues and the second is held at the issue stage. A pipeline freeze implies that a valid instruction or instructions is (are) presented to be issued but can not proceed.

Upon satisfying all issue requirements, instructions are allowed to continue through any pipeline toward completion. After issuing, instructions cannot be held in a given pipe stage. It is up to the issue stage to insure 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 Architecture Handbook.

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 stopping one instruction of a dual issued pair in the case of an abort condition on the first instruction of the pair. The non exception case, however, does not need to drain the pipeline of all outstanding instructions ahead of the aborting instruction. The pipeline can be immediately restarted at a redirected address. Examples of non exception abort conditions are branch mispredictions, subroutine call/return mispredictions and instruction cache misses. Data cache misses do not produce abort conditions but can cause pipeline freezes.

In the event of an exception, the processor aborts all instructions issued after the excepting instruction as described above. Due to the nature of some error conditions, this may occur as late as the write cycle. Next, the address of the excepting instruction is latched in the EXC_ ADDR IPR. When the pipeline is fully drained, the processor begins instruction execution at the address given by the PALcode dispatch. The pipeline is drained when all outstanding writes to both the integer and floating point register file have completed and 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.

It should be noted that there are two basic reasons for non-issue conditions. The first is a pipeline freeze wherein a valid instruction or pair of instructions are prepared to issue but cannot due to a resource 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 exist due to abort conditions as 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 hardware through bubble squashing, but can also be effectively minimized through careful coding practices. Bubble squashing involves the ability of the first four pipeline stages to advance whenever a bubble is detected in the pipeline stage immediately ahead of it while the pipeline is otherwise frozen.

2.9 Scheduling and Issuing Rules

2.9.1 Instruction Class Definition

It is important to note that the following scheduling and dual issue rules are only performance related. There are no functional dependencies related to scheduling or dual issuing. The scheduling and issuing rules are defined in terms of instruction classes. The table below specifies all of the instruction classes and the box which executes the particular class.

Table 2–1: Producer-Consumer Classes

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

2.9.2 Producer-Consumer Latency Matrix

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

The scheduling rules are described as a producer-consumer matrix. Each row and column in the matrix is a class of Alpha instructions. A ’1’ in the Producer-Consumer Latency Matrix indicates one cycle of latency. A one cycle latency means that if instruction B uses the results of instruction A, then instruction B may be issued ONE cycle after instruction A is issued.

The first thing to do when determining latency for a given instruction sequence is to identify the classes of all the instructions. The example below has the classes listed in the comment field.

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

The SRA instruction consumes the result (R3) produced by the ADDQ instruction. The latency associated with an iadd-shift producer-consumer pair as specified by the matrix is one. That means that if the ADDQ was issued in cycle ’n’ the SRA could be issued in cycle ’n+1’. The SUBQ instruction consumes the result (R5) produced by the SRA instruction. The latency associated with a shift-iadd producer-consumer pair as specified by the matrix is two. That means that if the SRA was issued in cycle ’n’ the SUBQ could be issued in cycle ’n+2’. The Ibox injects one nop cycle in the pipeline for this case.

The final case has the STQ instruction consuming the result (R7) produced by the SUBQ instruction. The latency associated with an iadd-st producer-consumer pair where the result of the iadd is the store data is zero. This means that the SUBQ and STQ instruction pair can be dual-issued.

Figure 2–5: Producer-Consumer Latency Matrix

```
                    Producer Class
```

Figure 2–5 (continued on next page)

Figure 2–5 (Cont.): Producer-Consumer Latency Matrix

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

Notes:

1. For loads, Dcache hit is assumed. The latency for a Dcache miss and an external cache hit is dependent on the system configuration. The latency is determined as the register file write time less 1 cycle.
2. For some producer classes, two latencies, X/Y, are given with the ST consumer class. X represents the latency for base address of store and Y represents the latency for store data. FDIV results cannot be used as the base address for store operations.
3. For IMUL followed by IMUL, there are two latencies given. The first represents the latency with data dependency, i.e. the second IMUL uses the result from the first. The second is the multiply latency without data dependencies.
4. For FDIV followed by FDIV, there are two latencies given. The first represents the latency with data dependency, i.e. the second FDIV uses the result from the first. The second is the division latency without data dependencies.

2.9.3 Producer-Producer Latency

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

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

2.9.4 Instruction Issue Rules

The following is a list of conditions that prevent instruction issue:

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.
2. No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR or BSR,BR,JSR(with destination other than R31) can be issued after a MB instruction until the MB has been acknowledged on the external pin bus.
3. No IMUL instructions can be issued if the integer multiplier is busy.
4. No SHIFT, IADDLOG, ICMP or ICMOV instruction can be issued exactly three cycles before an integer multiplication completes.
5. No integer or floating point conditional branch instruction can be issued in the cycle immediately following a JSR,JMP,RET,JSR_COROUTINE or HW_REI instruction.
6. No TRAPB instruction can be issued as the second instruction of a dual issue pair.
7. No LD instructions can be issued in the two cycles immediately following a STC.
8. No LD, ST, FETCH, MB, RPCC, RS, RC, TRAPB, HW_MXPR or BSR,BR,JSR(with destination other than R31) instruction can be issued when the Abox is busy due to a load miss or write buffer overflow. For more information see section 2.5.3.
9. No FDIV instruction can be issued if the floating pointer divider is busy.
10. No floating point operate instruction can be issued exactly five or exactly six cycles before the floating point divide completes.

2.9.5 Dual Issue Rules

The table below lists the classes of instruction pairs that can be issued in a single cycle. An instruction from a class in the first column below may be issued in the same cycle as an instruction from a class in the second column, in the absence of data dependencies and if the two instructions occupy the same aligned quadword in memory.

Table 2–2: Dual Issue Rules

Instruction 1	Instruction 2
LD integer	LD floating pt
LD floating pt	LD integer
ST floating pt	ST integer
FBR	IBR
IADDLOG	FPOP
SHIFT	FDIV
ICMP	JSR
ICMOV	BSR
IMUL	BR
	HW_x
	CALL_PAL

- No more than one of LD, ST, HW_MxPR, FETCH, RPCC, RS, RC, MB, TRAPB, BSR, BR or JSR can be issued in the same cycle.
- No more than one of JSR, IBR, BSR, HW_REI, BR or FBR can be issued in the same cycle.

Chapter 3

Privileged Architecture Library Code

3.1 Introduction

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

These functions range from the binary encoding of the instructions and data, to the exception mechanisms and synchronization primitives. Some of these functions can be cost effectively implemented in hardware. However several of these functions are impractical to implement directly in hardware. They include low-level hardware support functions such as translation buffer fill routines, interrupt acknowledge, and exception dispatch. Also included is support for privileged and atomic operations that require long instruction sequences such as Return from Exception or Interrupt (REI).

In a CISC architecture, these functions are generally provided by the use of microcode. In the 21064 RISC microprocessor, there is no microcode. Instead there is an architected interface to functions that will be consistent with other members of the Alpha family of machines. The Privileged Architecture Library Code (PALcode) is used to implement these functions without resorting to a microcoded machine.

3.2 PAL Environment

PALcode runs in an environment with privileges enabled, instruction stream mapping disabled, and interrupts disabled. The enabling of privileges allows all functions of the machine to be controlled. Disabling of instruction stream mapping allows PALcode to be used to support the memory management functions (e.g., translation buffer fill routines can not be run via mapped memory).

PALcode can perform both virtual and physical data stream references. The disabling of interrupts allows the system to provide multi-instruction sequences as atomic operations. The PALcode environment in the 21064 also includes 32 PAL temp registers which are accessible only by special PAL instructions.

3.3 Special PAL Instructions

PALcode uses the Alpha instruction set for most of its operations. The 21064 maps the architecturally reserved PALcode opcodes (PALRES0 - PALRES4) to a special load and store (HW_LD, HW_ST), a move to and move from processor register (HW_MTPR, HW_MFPR), and a return from PALmode exception (HW_REI). These instructions produce a Reserved Opcode fault if executed while not in the PALcode environment unless the HWE bit of the ICCSR IPR is set, in which case these instructions can be executed in kernel mode.

Register checking and bypassing logic is provided for PALcode instructions as it is for non-PALcode instructions when using general purpose registers. Explicit software timing is required for accessing the hardware specific IPRs and the PAL_TEMPs. These constraints are described in the PALmode restriction and IPR sections.

3.3.1 HW_MFPR and HW_MTPR

The internal processor register specified by the PAL, ABX, IBX, and index field is written /read with the data from the specified integer register. Processor registers may have side effects that happen as the result of writing/reading them. Coding restrictions are associated with accessing various registers. Separate bits are used to access Abox IPRs, Ibox IPRs, and PAL_TEMPs, therefore it is possible for an MTPR instructions to write multiple registers in parallel if they both have the same index.

The HW_MFPR and HW_MTPR instructions have the following format:

Figure 3–1: HW_MxPR Instruction Format

```
 3         2 2       2 2       1 1           0 0 0 0 0           0
 1         6 5       1 0       6 5           8 7 6 5 4           0
.----------+---------+---------+-----------+-+-+-+-+-----------.
|          |         |         |           |P|A|I|             |
|  OPCODE  |   RA    |   RB    |    IGN    |A|B|B|    INDEX    |
|          |         |         |           |L|X|X|             |
`----------+---------+---------+-----------+-+-+-+-+-----------'
```

Table 3–1: HW_MFPR and HW_MTPR Format Description

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

Table 3–1 (Cont.): HW_MFPR and HW_MTPR Format Description

Field	Description
INDEX	Specifies hardware specific register as shown in Table 3–2

The following table indicates how the PAL, ABX, IBX, and INDEX fields are set to access the internal processor registers. Setting the PAL, ABX, and IBX fields to zero generates a NOP.

Table 3–2: IPR Access

Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
TB_TAG	x	x	1	0	W	PAL mode only
ITB_PTE	x	x	1	1	R/W	PAL mode only
ICCSR	x	x	1	2	R/W	
ITB_PTE_TEMP	x	x	1	3	R	PAL mode only
EXC_ADDR	x	x	1	4	R/W	
SL_RCV	x	x	1	5	R	
ITBZAP	x	x	1	6	W	PAL mode only
ITBASM	x	x	1	7	W	PAL mode only
ITBIS	x	x	1	8	W	PAL mode only
PS	x	x	1	9	R/W	
EXC_SUM	x	x	1	10	R/W	
PAL_BASE	x	x	1	11	R/W	
HIRR	x	x	1	12	R	
SIRR	x	x	1	13	R/W	
ASTRR	x	x	1	14	R/W	
HIER	x	x	1	16	R/W	
SIER	x	x	1	17	R/W	
ASTER	x	x	1	18	R/W	
SL_CLR	x	x	1	19	W	
SL_XMIT	x	x	1	22	W	
TB_CTL	x	1	x	0	W	
DTB_PTE	x	1	x	2	R/W	
DTB_PTE_TEMP	x	1	x	3	R	

Table 3–2 (Cont.): IPR Access

Mnemonic	PAL	ABX	IBX	INDEX	Access	Comments
MMCSR	x	1	x	4	R	
VA	x	1	x	5	R	
DTBZAP	x	1	x	6	W	
DTASM	x	1	x	7	W	
DTBIS	x	1	x	8	W	
BIU_ADDR	x	1	x	9	R	
BIU_STAT	x	1	x	10	R	
DC_ADDR	x	1	x	11	R	
DC_STAT	x	1	x	12	R	
FILL_ADDR	x	1	x	13	R	
ABOX_CTL	x	1	x	14	W	
ALT_MODE	x	1	x	15	W	
CC	x	1	x	16	W	
CC_CTL	x	1	x	17	W	
BIU_CTL	x	1	x	18	W	
FILL_SYNDROME	x	1	x	19	R	
BC_TAG	x	1	x	20	R	
FLUSH_IC	x	1	x	21	W	
FLUSH_IC_ASM	x	1	x	23	W	
PAL_TEMP[31..0]	1	x	x	31-00	R/W	

3.3.2 HW_LD and HW_ST

PALcode uses the HW_LD and HW_ST instructions to access memory outside of the realm of normal Alpha memory management. The HW_LD and HW_ST instructions have the following format:

Figure 3–2: HW_LD/HW_ST Instruction Format

```
 3        2 2      2 2      1 1 1 1 1 1                    0
 1        6 5      1 0      6 5 4 3 2 1                    0
.----------+--------+--------+-+-+-+-+---------------------.
|          |        |        |P|A|R|Q|                     |
|  OPCODE  |   RA   |   RB   |H|L|W|W|        DISP         |
|          |        |        |Y|T|C| |                     |
`----------+--------+--------+-+-+-+-+---------------------'
```

The effective address of these instructions is calculated as follows:

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

Table 3–3: HW_LD and HW_ST Format Description

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

3.3.3 HW_REI

The HW_REI instruction uses the address in the Ibox EXC_ADDR IPR to determine the new virtual program counter (VPC). Bit zero of the EXC_ADDR indicates the state of the PALmode bit on the completion of the HW_REI. If EXC_ADDR bit[0] is set then the processor remains in PALmode. This allows PALcode to transition from PALmode to non-PALmode. The HW_REI instruction can also be used to jump from PALmode to PALmode. This allows PAL instruction flows to take advantage of the D-stream mapping hardware in the 21064, including traps. The HW_REI instruction has the following format:

Figure 3–3: HW_REI Instruction Format

```
 3        2 2      2 2      1 1 1 1                       0
 1        6 5      1 0      6 5 4 3                       0
.----------+--------+--------+-+-+-------------------------.
|          |        |        | | |                         |
|  OPCODE  |   RA   |   RB   |1|0|          IGN            |
|          |        |        | | |                         |
`----------+--------+--------+-+-+-------------------------'
```

Note that bits[15..14] contain the branch prediction hint bits. The 21064 pushes the contents of the EXC_ADDR register on the JSR prediction stack. Bit[15] must be set to pop the stack to avoid misalignment. The next address and PALmode bit are calculated as follows:

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

Table 3–4: The HW_REI Format Description

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

3.4 PAL Entry Points

When an exception or interrupt occurs, the 21064 first drains the pipeline, loads the PC into the EXC_ADDR IPR and then dispatches to one of the exception routines. The pipeline is drained when all instructions that update either register file have completed, and all instructions that do not update the register files are guaranteed to complete without an exception in the absence of a machine check. In addition, all pending Dcache fill operations must have completed before dispatch to one of the exception routines. If multiple exceptions occur, the 21064 dispatches to the highest priority PAL entry point. The table below prioritizes entry points from highest to lowest priority, i.e. the first row in the table (reset) has the highest priority.

The table below defines only the entry point offset, bits [13..0]. The high-order bits of the new PC (bits [33..14]) come from the PAL_BASE IPR.

Note that PALcode at PAL entry points of higher priority than DTBMISS must unlock possible MMCSR IPR and VA IPR locks.

Table 3–5: PAL Entry Points

Entry Name	Time	Offset(Hex)	Cause
RESET	anytime	0000	

Table 3–5 (Cont.): PAL Entry Points

Entry Name	Time	Offset(Hex)	Cause
MCHK	pipe_stage[7]	0020	Uncorrected hardware error.
ARITH	anytime	0060	Arithmetic exception.
INTERRUPT	anytime	00E0	Includes corrected hardware error.
D-stream errors	pipe_stage[6]	01E0, 08E0, 09E0, 11E0	See Table 3–6.
ITB_MISS	pipe_stage[5]	03E0	ITB miss.
ITB_ACV	pipe_stage[5]	07E0	I-stream access violation.
CALL_PAL	pipe_stage[5]	2000,40,80,C0 thru 3FC0	128 locations based on instruction[7,5:0]. See description below.
OPCDEC	pipe_stage[5]	13E0	Reserved or privileged opcode.
FEN	pipe_stage[5]	17E0	FP op attempted with :
			FP instructions disabled via ICCSR FPE bit
			FP IEEE round to +/- infinity
			FP IEEE with datatype field other than S,T,QW

PALcode functions are implemented via the CALL_PAL instruction. CALL_PAL instructions cause exceptions in the hardware. As with all exceptions, the EXC_ADDR register is loaded by hardware with a possible return address.

The 21064 loads the EXC_ADDR register with the address of the instruction *following* the CALL_PAL. For this reason, PALcode supporting the desired PAL mode function should not increment the EXC_ADDR register before executing a HW_REI instruction to return to native mode. This feature requires special handling in the arithmetic trap and machine check PALcode flows. See Section 3.8.6 EXC_ADDR for more complete information.

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

The 21064 provides the first 64 privileged and 64 unprivileged CALL_PAL instructions with code regions of 64 bytes. This produces hardware PAL entry points as described below.

Privileged CALL_PAL Instructions [00000000:0000003F]

Offset(Hex) = 2000 + (<5:0> shift left 6)

Unprivileged CALL_PAL Instructions [00000080:000000BF]

Offset(Hex) = 3000 + (<5:0> shift left 6)

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

The PAL entry points assigned to D-stream errors require a bit more explanation. The hardware recognizes four classes of D-stream memory management errors: bad virtual address (improper sign extension), DTB miss, alignment error and everything else (ACV, FOR, FOW). These errors get mapped into four PAL entry points: UNALIGN, DTB_MISS PAL mode, DTB_MISS Native mode and D_FAULT. Table 3–6 lists the priority of these entry points as a group with respect to each of the other entry points. Since a particular D-stream memory reference may generate errors which fall into more than one of the four error classes which the hardware recognizes, we also must define the priority of each of the D-stream PAL entry points with respect to the others in the D-stream PAL entry group. Table 3–6 gives this priority.

Table 3–6: D-stream Error PAL Entry Points

BAD_VA	DTB_ MISS	UNALIGN	PAL	Other	Offset(Hex)
1	x	0	x	x	01E0 D_FAULT
1	x	1	x	x	11E0 UNALIGN
0	1	x	0	x	08E0 DTB_MISS Native
0	1	x	1	x	09E0 DTB_MISS PAL
0	0	1	x	x	11E0 UNALIGN
0	0	0	x	1	01E0 D_FAULT

3.5 PALmode Restrictions

Many of the restrictions involve waiting 'n' cycles before using the results of PAL instructions. Inserting 'n' instructions between the two time-sensitive instructions is the typical method of waiting for 'n' cycles. Because the 21064 can dual issue instructions it is possible to write code that requires 2*n+1 instructions to wait 'n' cycles. Due to the resource requirements of individual instructions, and the 21064 hardware design, multiple copies of the same instruction can not be dual issued. This fact is used in some of the code examples below.

1. As a general rule, HW_MTPR instructions require at least 4 cycles to update the selected IPR. Therefore, at least three cycles of delay must be inserted before using the result of the register update.

   Note that only the write followed by read operation requires this software timing. Multiple reads, multiple writes, or read followed by write will pipeline properly and do not require software timing except for accesses of the TB registers.

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

An example of this is :

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

The HW_REI instruction uses the ITB if the EXC_ADDR register contains a non PAL mode VPC, VPC<0> = 0. By the rule above, this implies that at least 3 cycles of delay must be included after writing the ITB before executing a HW_REI instruction to exit PAL mode.

Exceptions:

- HW_MFPR instructions reading any PAL_TEMP register can never occur exactly two cycles after a HW_MTPR instruction writing any PAL_TEMP register. The simple solution results in code of the form:

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

  The above code guarantees three cycles of delay after the write before the read. It is also possible to make use of the cycle immediately following a HW_MTPR to execute a HW_MFPR instruction from a different PAL_TEMP register. This requires that the second instruction not stall for exactly one cycle. This is much easier to insure. A HW_MFPR instruction may stall for a single cycle as a result of a write after write conflict.

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

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

2. An MTPR operation to the DTBIS register cannot be bypassed into. In other words, all data being moved to the DTBIS register must be sourced directly from the register file. One way to insure this is to provide at least three cycles of delay before using the result of any integer operation (except MUL) as the source of an MTPR DTBIS. Do not use a MUL as the source of DTBIS data. Sample code for this operation is :

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

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

4. The first cycle (the first one or two instructions) at all PALcode entry points may not execute a conditional branch instruction or any other instruction that uses the jsr stack hardware. This includes instructions JSR,JMP,RET,COROUTINE, BSR,HW_REI and all Bxx opcodes except BR, which is allowed.

5. The following table indicates the number of cycles required after a HW_MTPR instruction before a subsequent HW_REI instruction for the specified IPRs. These cycles can be insured by inserting one HW_MFPR R31,0 instruction or other appropriate instruction(s) for each cycle of delay required after the HW_MTPR.

Table 3–7: HW_MTPR/HWMTPR Restrictions

IPR	Cycles between HW_MTPR and HW_REI
DTBIS,ASM,ZAP	0
ITBIS,ASM,ZAP	2
xIER	3
xIRR	3
PS	4
ICCSR<FPE>	3
ICCSR<ASN>	9 ; Cycles before HW_REI
FLUSH_IC[ASM]	9 ; target can be fetched.

6. When loading the CC register, bits <3:0> must be loaded with zero. Loading non-zero values in these bits may cause the count to be inaccurate.

7. An MTPR DTBIS cannot be combined with an MTPR ITBIS instruction. The hardware will not clear the ITB if both the Ibox and Abox IPRs are simultaneously selected. Instead, two instructions are needed to clear each TB individually. Code example:

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

8. Three cycles of delay are required between:

```
MTxIER and MFxIRR
MTxIRR and MFxIRR
MTPS   and LD or ST, all
MTPS   and MFxIRR
```

9. A HW_MxPR ITB_TAG, ITB_PTE, ITB_PTE_TEMP cannot follow a HW_REI that remains in PAL mode. (Address bit<0> of the EXC_ADDR is set) This rule implies that it is not a good idea to ever allow exceptions while updating the ITB. If an exception interrupts flow of the ITB fill routine and attempts to REI back, and the return address begins with a HW_MxPR instruction to an ITB register, and the REI is predicted correctly to avoid any delay between the two instructions, then the ITB register will not be written. Code example:

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

10. The ITB_TAG,ITB_PTE and ITB_PTE_TEMP registers can only be accessed in PAL mode. If the instructions HW_MTPR or HW_MFPR to/from the above registers are attempted while not in PAL mode by setting the HWE (hardware enable) bit of the ICCSR, the instructions will be ignored.

11. Machine check exceptions taken while in PAL mode may load the EXC_ADDR register with a restart address one instruction earlier than the proper restart address. Some HW_MxPR instructions may have already completed execution even though the restart address indicates the HW_MxPR as the return instruction. Re-execution of some HW_MxPR instructions can alter machine state. (e.g. TB pointers, EXC_ADDR register mask)

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

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

12. When writing the PAL_BASE register, exceptions may not occur. An exception occurring simultaneously with a write to the PAL BASE may leave the register in a metastable state. All asynchronous exceptions but reset can be avoided under the following conditions:

```
PAL mode .................... blocks all interrupts
machine checks disabled ..... blocks I/O error exceptions
  (via ABOX_CTL reg or MB isolation)
Not under trap shadow ....... avoids arithmetic traps
```

The trap shadow is defined as less than:

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

13. The sequence HW_MTPR PTE, HW_MTPR TAG is NOT allowed. At least two null cycles must occur between HW_MTPR PTE and HW_MTPR TAG.
14. The AMCHK exception service routine must check the EXC_SUM register for simultaneous arithmetic errors. Arithmetic traps will not trigger exceptions a second time after returning from exception service for the machine check.
15. Three cycles of delay must be inserted between HW_MFPR DTB_PTE and HW_MFPR DTB_PTE_TEMP. Code example:

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

16. Three cycles of delay must be inserted between HW_MFPR IPTE and HW_MFPR ITB_PTE_TEMP. Code example:

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

17. The content of the destination register for HW_MFPR Rx,DTB_PTE or HW_MFPR Rx,ITB_PTE is UNPREDICTABLE.
18. Two HW_MFPR DTB_PTE instructions cannot be issued in consecutive cycles. This implies that more than one instruction may be necessary between the HW_MFPR instructions if dual issue is possible. Similar restrictions apply to the ITB_PTE register.
19. Reading the EXC_SUM and BC_TAG registers require special timing. Refer to Section 3.8.13 and Section 3.10.7 for specific information.
20. DMM errors occurring one cycle before HW_MxPR instructions to the IPTE will NOT stop the TB pointer from incrementing to the next TB entry even though the HW_MxPR instruction will be aborted by the DMM error. This restriction only affects performance and not functionality.
21. PALcode that writes multiple ITB entries must write the entry that maps the address contained in the EXC_ADDR register last.

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

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

3.6 Power Up

The table below lists the state of all the IPRs immediately following reset. The table also specifies which IPRs need to be initialized by power-up PALcode.

Table 3–8: IPR Reset State

IPR	Reset State	Comments
ITB_TAG	undefined	
ITB_PTE	undefined	
ICCSR	cleared except ASN,PC0,PC1	Floating point disabled, single issue mode, Pipe mode enabled, ASN = 0, jsr predictions disabled, branch predictions disabled, branch history table disabled, performance counters reset to zero, Perf Cnt0(16b) : Total Issues/2, Perf Cnt1(12b) : Dcache Misses
ITB_PTE_TEMP	undefined	
EXC_ADDR	undefined	
SL_RCV	undefined	
ITBZAP	n/a	PALcode must do a itbzap on reset.
ITBASM	n/a	
ITBIS	n/a	
PS	undefined	PALcode must set processor status.
EXC_SUM	undefined	Palcode must clear exception summary and exception register write mask by doing 64 reads.
PAL_BASE	cleared	Cleared on reset.
HIRR	n/a	
SIRR	undefined	PALcode must initialize.
ASTRR	undefined	PALcode must initialize.
HIER	undefined	PALcode must initialize.
SIER	undefined	PALcode must initialize.

Table 3–8 (Cont.): IPR Reset State

IPR	Reset State	Comments
ASTER	undefined	PALcode must initialize.
SL_XMIT	undefined	PALcode must initialize. Appears on external pin.
TB_CTL	undefined	Palcode must select between SP/LP dtb prior to any TB fill.
DTB_PTE	undefined	
DTB_PTE_TEMP	undefined	
MMCSR	undefined	Unlocked on reset.
VA	undefined	Unlocked on reset.
DTBZAP	n/a	PALcode must do a dtbzap on reset.
DTBASM	n/a	
DTBIS	n/a	
BIU_ADDR	undefined	Potentially locked.
BIU_STAT	undefined	Potentially locked.
SL_CLR	undefined	PALcode must initialize.
DC_ADDR	undefined	Potentially locked.
DC_STAT	undefined	Potentially locked.
FILL_ADDR	undefined	Potentially locked.
ABOX_CTL	see comments	[11..0] <- ^x0100 Write buffer enabled, machine checks disabled, correctable read interrupts disabled, Icache stream buffer disabled, Dcache disabled, forced hit mode off.
ALT_MODE	undefined	
CC	undefined	Cycle counter is disabled on reset.
CC_CTL	undefined	
BIU_CTL	see comments	Bcache disabled, parity mode undefined, chip enable asserts during RAM write cycles, Bcache forced-hit mode disabled. BC_PA_DIS field cleared. BAD_TCP cleared. BAD_DP undefined. Note: The Bcache parameters BC RAM read speed, BC RAM write speed, BC write enable control, and BC size are all undetermined on reset and must be initialized before enabling the Bcache.
FILL_SYNDROME	undefined	Potentially locked.
BC_TAG	undefined	Potentially locked.

Table 3–8 (Cont.): IPR Reset State

IPR	Reset State	Comments
PAL_TEMP[31..0]	undefined	

PALcode should execute four jsr call instructions to initialize the jsr stack. This is necessary to insure deterministic behavior for testers. The following code will initialize the stack once the ICCSR [JSE] bit is set.

```
BSR    r1,stk_1        ; push RET PC

stk_1:
BSR    r2,stk_2        ; push RET PC

stk_2:
BSR    r3,stk_3        ; push RET PC

stk_3:
BSR    r4,stk_4        ; push RET PC

stk_4:
```

3.7 TB Miss Flows

This section describes hardware specific details to aid the PALcode programmer in writing ITB and DTB fill routines. These flows were included to highlight trade-offs and restrictions between PAL and hardware. The PALcode source that is released with the 21064 should be consulted before any new flows are written. A working knowledge of the Alpha memory management architecture is assumed.

3.7.1 ITB Miss

When the Ibox encounters an ITB miss it latches the VPC of the target instruction-stream reference in the EXC_ADDR IPR, flushes the pipeline of any instructions following the instruction which caused the ITB miss, waits for any other instructions which may be in progress to complete, enters PALmode, and jumps to the ITB miss PAL entry point. The recommended PALcode sequence for translating the address and filling the ITB is described below.

1. Create some scratch area in the integer register file by writing the contents of a few integer registers to the PAL_TEMP register file.
2. Read the target virtual address from the EXC_ADDR IPR.
3. Fetch the PTE (this may take multiple reads) using a physical-mode HW_LD instruction. If this PTE's valid bit is clear report TNV or ACV as appropriate.

4. Since the Alpha Architecture Handbook states that translation buffers may not contain invalid PTEs, the PTE's valid bit must be explicitly checked by PALcode. Further, since the ITB's PTE RAM does not hold the FOE bit, the PALcode must also explicitly check this condition. If the PTE's valid bit is set and FOE bit is clear, PALcode may fill an ITB entry.
5. Write the original virtual address to the TB_TAG register using HW_MTPR. This writes the TAG into a temp register and not the actual tag field in the ITB.
6. Write the PTE to the ITB_PTE register using HW_MTPR. This HW_MTPR causes both the TAG and PTE fields in the ITB to be written. Note it is not necessary to delay issuing the HW_MTPR to the ITB_PTE after the MTPR to the ITB_TAG is issued.
7. Restore the contents of any modified integer registers to their original state using the HW_MFPR instruction.
8. Restart the instruction stream using the HW_REI instruction.

3.7.2 DTB Miss

When the Abox encounters a DTB miss it latches the referenced virtual address in the VA IPR and other information about the reference in the MMCSR IPR, and locks these registers against further modifications. The Ibox latches the PC of the instruction which generated the reference in the EXC_ADDR register, drains the machine as described above for ITB misses, and jumps to the DTB miss PALcode entry point. Unlike ITB misses, DTB misses may occur while the CPU is executing in PALmode. The recommended PALcode sequence for translating the address and filling the DTB is described below.

1. Create some scratch area in the integer register file by writing the contents of a few integer registers to the PAL_TEMP register file.
2. Read the requested virtual address from the VA IPR. Although the act of reading this register unlocks the VA and MMCSR registers, the MMCSR register only updates when D-stream memory management errors occur. It therefore will retain information about the instruction which generated this DTB miss. This may be useful later.
3. Fetch the PTE (may require multiple reads). If the valid bit of the PTE is clear, a TNV or ACV must be reported unless the instruction which caused the DTB miss was FETCH or FETCH/M. This can be checked via the opcode field of the MMCSR register. If the value in this field is 18 (hex), then a FETCH or FETCH/M instruction caused this DTB miss, and as mandated by the Alpha Architecture Handbook, the subsequent TNV or ACV should NOT be reported. Therefore PALcode should read the value in EXC_ADDR, increment it by four, write this value back to EXC_ADDR, and do a HW_REI.
4. Write the register which holds the contents of the PTE to the TB_CTL IPR. This has the effect of selecting one of the four possible granularity sizes of the DTB for subsequent DTB fill operations, based on the value contained in the granularity hint field of the PTE.
5. Write the original virtual address to the TB_TAG register. This writes the TAG into a temp register and not the actual tag field in the DTB
6. Write TB Size selection to TB_CTL.

7. Wait at least one cycle.
8. Write the PTE to the DTB_PTE register. This HW_MTPR causes both the TAG and PTE fields in the DTB to be written. Note it is not necessary to delay issuing the HW_MTPR to the DTB_PTE after the MTPR to the DTB_TAG is issued.
9. Restore the contents of any modified integer registers.
10. Restart the instruction stream using the HW_REI instruction.

3.8 Internal Processor Registers - IPRs

3.8.1 Ibox IPRs

3.8.2 TB_TAG

The TB_TAG register is a write-only register which holds the tag for the next TB update operation in either the ITB or DTB. To insure the integrity of the TB, the tag is actually written to a temporary register and not transferred to the ITB or DTB until the ITB_PTE or DTB_PTE register is written. The entry to be written is chosen at the time of the ITB_PTE or DTB_PTE write operation by a not-last-used algorithm implemented in hardware.

Writing the ITB_TAG register is only performed while in PALmode regardless of the state of the HWE bit in the ICCSR IPR.

Figure 3–4: Translation Buffer Tag (TB_TAG) Registor

```
Small Page Format:

 6            4 4     1 1                              0
 3            3 2     3 2                              0
.--------------+--------------------------+----------------.
|              |                          |                |
|      IGN     |        VA[42..13]        |       IGN      |
|              |                          |                |
`--------------+--------------------------+----------------'

GH = 11(bin) Format (DTB only):

 6            4 4              2 2                     0
 3            3 2              2 1                     0
.--------------+---------------+---------------------------.
|              |               |                           |
|      IGN     | VA[42..22]    |            IGN            |
|              |               |                           |
`--------------+---------------+---------------------------'
```

3.8.3 ITB_PTE

The ITB PTE register is a read/write register representing the eight ITB page table entries. The entry to be written is chosen by a not-last-used algorithm implemented in hardware. Writes to the ITB_PTE use the memory format bit positions as described in the Alpha Architecture Handbook with the exception that some fields are ignored.

To insure the integrity of the ITB, the ITB's tag array is updated simultaneously from the internal tag register when the ITB_PTE register is written. Reads of the ITB_PTE require two instructions. First, a read from the ITB_PTE sends the PTE data to the ITB_PTE_TEMP register, then a second instruction reading from the ITB_PTE_TEMP register returns the PTE entry to the register file. Reading or writing the ITB_PTE register increments the TB entry pointer which allows reading the entire set of eight ITB PTE entries.

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

Figure 3–5: ITB_PTE Register

```
Write Format:

 6       5 5                 3 3          1 1 1 0 0 0 0 0 0 0 0 0 0
 3       3 2                 2 1          2 1 0 9 8 7 6 5 4 3 2 1 0
.---------+-----------------+------------+-+-+-+-+-----+-+-------.
|         |                 |            |U|S|E|K|     |A|       |
|   IGN   |  PFN[33..13]    |    IGN     |R|R|R|R| IGN |S|  IGN  |
|         |                 |            |E|E|E|E|     |M|       |
`---------+-----------------+------------+-+-+-+-+-----+-+-------'

Read Format:

 6                   3 3 3          1 1 1 1 0 0 0 0 0 0 0 0 0 0
 3                   5 4 3          3 2 1 0 9 8 7 6 5 4 3 2 1 0
.--------------------+-+--------------+-+-+-+-+-----------------.
|                    |A|              |U|S|E|K|                 |
|        RAZ         |S| PFN[33..13]  |R|R|R|R|       RAZ       |
|                    |M|              |E|E|E|E|                 |
`--------------------+-+--------------+-+-+-+-+-----------------'
```

3.8.4 Instruction Cache Control/Status Register - ICCSR

The ICCSR register contains various Ibox hardware enables. The only architecturally defined bit in this register is the FPE, floating point enable, which enables floating point instruction execution. When clear, all floating point instructions generate FEN exceptions. This register is cleared by hardware at reset. The HWE bit allows the special PAL instructions to execute in kernel mode. This bit is intended for diagnostics or operating system alternative PAL routines only. It does not allow access to the ITB registers while not running in PALmode. Therefore, some PALcode flows may require the PALmode environment to execute properly (e.g. ITB fill).

Figure 3–6: ICCSR Register

Write Format:

```
 6   5 5      4 4   4 4 4 4 3 3 3 3 3 3 3 3 3   1 1 1 0 0 0 0 0 0 0 0 0
 3   3 2      7 6   3 2 1 0 9 8 7 6 5 4 3 2 1   2 1 0 9 8 7 5 4 3 2 1 0
.-----+--------+-----+-+-+-+-+-+-+-+-+-----+-----+-------+---+-+-+---+-.
|     |        | IC  |F|M|H|D|B|J|B|P|  PC |     | PC    |   |R|P|   |P|
| IGN |ASN[5:0]|[5:2]|P|A|W|I|H|S|P|I|MUX1 | IGN | MUX0  |IGN|E|C|IGN|C|
|     |        |     |E|P|E| |E|E|E|P|[2:0]|     | [3:0] |   |S|0|   |1|
|     |        |     | | | | | | | |E|     |     |       |   | | |   | |
`-----+--------+-----+-+-+-+-+-+-+-+-+-----+-----+-------+---+-+-+---+-'
```

Read Format:

```
 6   3 3 3      2 2   2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
 3   5 4 3      8 7   4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
.-----+-+--------+-----+-+-+-+-+-+-+-+-+-----+-------+-----------+-+-+-.
|     |R|        |     |F|M|H|D|B|J|B|P| PC  | PC    |           |P|P|R|
| RAZ |E|ASN[5:0]| RES |P|A|W|I|H|S|P|I|MUX1 | MUX0  |    RAZ    |C|C|A|
|     |S|        |     |E|P|E| |E|E|E|P|[2:0]| [3:0] |           |1|0|Z|
|     | |        |     | | | | | | | |E|     |       |           | | | |
`-----+-+--------+-----+-+-+-+-+-+-+-+-+-----+-------+-----------+-+-+-'
```

RES = Reserved

Table 3–9: ICCSR

Field	Type	Description
FPE	RW,0	If set, floating point instructions can be issued. If clear,floating point instructions cause FEN exceptions.
HWE	RW,0	If set, allows the five PALRES instructions to be issued in kernel mode. Use of the HW_MTPR instruction to update the EXC_ADDR IPR while in native mode is restricted to values with bit<0> equal to 0. The combination of native mode and EXC_ADDR<0> equal to one causes UNDEFINED behavior.)
MAP	RW,0	If set, allows super page I-stream memory mapping of VPC<33:13> directly to Physical PC<33:13> essentially bypassing ITB for VPC addresses containing VPC<42:41>= 2. Super page mapping is allowed in Kernel mode only. The ASM bit is always set.
DI	RW,0	If set, enables dual issue.
BHE	RW,0	Used in conjunction with BPE. See table Table 3–9 for programming information.
JSE	RW,0	If set, enables the JSR stack to push return addresses.
BPE	RW,0	Used in conjunction with BHE. See table Table 3–9 for programming information.

Table 3–9 (Cont.): ICCSR

Field	Type	Description
PIPE	RW,0	If clear, causes all hardware interlocked instructions to drain the machine and waits for the write buffer to empty before issuing the next instruction. Examples of instructions that do not cause the pipe to drain include HW_MTPR, HW_REI, conditional branches, and instructions that have a destination register of R31.
PCMUX1	RW,0	See table Table 3–11 for programming information.
PCMUX0	RW,0	See table Table 3–10 for programming information.
PC1	RW	If clear, enables performance counter 1 interrupt request after 2**12 events counted. If set, enables performance counter 1 interrupt request after 2**8 events counted.
PC0	RW	If clear, enables performance counter 0 interrupt request after 2**16 events counted. If set, enables performance counter 0 interrupt request after 2**12 events counted.
ASN	RW	The Address Space Number field is used in conjunction with the Icache to further qualify cache entries and avoid some cache flushes. The ASN is written to the Icache during fill operations and compared with the I-stream data on fetch operations. Mismatches invalidate the fetch without affecting the Icache.
RESERVED	RW	The IC state bits are unused by hardware.

Table 3–10: BHE, BPE Branch Prediction Selection

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

3.8.4.1 Performance Counters

The performance counters are reset to zero upon powerup, but are otherwise never cleared. They are intended as a means of counting events over a long period of time relative to the event frequency and therefore provide no means of extracting intermediate counter values. Since the counters continuously accumulate selected events despite interrupts being enabled, the first interrupt after selecting a new counter input has an error bound as large as the selected overflow range. In addition, some inputs may overcount events occurring simultaneously with D-stream errors which abort the actual event very late in the pipeline. For example, when counting load instructions, attempts to execute a load resulting in a DTB miss exception will increment the performance counter after the first aborted execution attempt and again after the TB fill routine when the load instruction reissues and completes.

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

The six cycles before an interrupt is triggered implies that a maximum of 12 instructions may have completed before the start of the interrupt service routine. In most cases, by examining the possible intervening instructions and the issue rules presented in section 2.9, it is possible to further isolate trigger events. Two cases always provide a more accurate exception PC. When counting Icache misses, no intervening instructions can complete and the exception PC contains the address of the last Icache miss. Branch mispredictions allow a maximum of only 2 instructions to complete before start of the interrupt service routine.

Table 3–11: Performance Counter 0 Input Selection

MUX0[3:0]	Input	Comment
000X	Total Issues / 2	Counts total issues divided by 2, e.g dual issue increments count by 1
001X	Pipeline Dry	Counts cycles where nothing issued due to lack of valid I-stream data. Causes include Icache fill, misprediction, branch delay slots and pipeline drain for exception
010X	Load Instructions	Counts all Load instructions
011X	Pipeline Frozen	Counts cycles where nothing issued due to resource conflict. Refer to section 2.9 for information regarding scheduling and issue rules.
100X	Branch Instructions	Counts all Branch instructions, conditional, unconditional, any JSR, HW_REI
1010	PALmode	Counts cycles while executing in PAL mode
1011	Total cycles	Counts total cycles
110X	Total Non-issues / 2	Counts total non_issues divided by 2, e.g no issue increments count by 1
111X	PERF_CNT_H<0>	Counts external event supplied by pin at selected system clock cycle interval

Table 3–12: Performance Counter 1 Input Selection

MUX1[2:0]	Input	Comment
000	Dcache miss	Counts total Dcache misses

Table 3–12 (Cont.): Performance Counter 1 Input Selection

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

3.8.5 ITB_PTE_TEMP

The ITB_PTE_TEMP register is a read-only holding register for ITB_PTE read data. Reads of the ITB_PTE require two instructions to return the data to the register file. The first reads the ITB_PTE register to the ITB_PTE_TEMP register. The second returns the ITB_PTE_TEMP register to the integer register file. The ITB_PTE_TEMP register is updated on all ITB accesses, both read and write. A read of the ITB_PTE to the ITB_PTE_TEMP should be followed closely by a read of the ITB_PTE_TEMP to the register file.

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

Figure 3–7: ITB_PTE_TEMP Register

```
 6                   3 3 3              1 1 1 1 0 0        0
 3                   5 4 3              3 2 1 0 9 8        0
.---------------------+-+----------------+-+-+-+-+---------.
|                     |A|                |U|S|E|K|         |
|         RAZ         |S|   PFN[33..13]  |R|R|R|R|   RAZ   |
|                     |M|                |E|E|E|E|         |
`---------------------+-+----------------+-+-+-+-+---------'
```

3.8.6 Exceptions Address Register (EXC_ADDR)

The EXC_ADDR register is a read/write register used to restart the machine after exceptions or interrupts. The EXC_ADDR register can be read and written by software via the HW_MTPR instruction as well as being written directly by hardware. The HW_REI instruction executes a jump to the address contained in the EXC_ADDR register. The EXC_ADDR register is written by hardware after an exception to provide a return address for PALcode.

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

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

This feature, however, requires careful treatment in PALcode. Arithmetic traps and machine check exceptions can preempt CALL_PAL exceptions resulting in an incorrect value being saved in the EXC_ADDR register. In the cases of an arithmetic trap or machine check exception, and only in those cases, EXC_ADDR<1> takes on special meaning. PALcode servicing these two exceptions should interpret a zero in EXC_ADDR<1> as indicating that the PC in EXC_ADDR<63:2> is too large by a value of 4bytes and subtract 4 before executing a HW_REI from this address. PALcode should interpret a one in EXC_ADDR<1> as indicating that the PC in EXC_ADDR<63:2> is correct and clear the value of EXC_ADDR<1>. All other PALcode entry points except reset can expect EXC_ADDR<1> to be zero.

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

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

Figure 3–8: Exception Address Rigister (EXC_ADDR)

```
 6                                                     0 0 0
 3                                                     2 1 0
.-----------------------------------------------------+-+-.
|                                                     |I|P|
|                  PC[63..2]                          |G|A|
|                                                     |N|L|
`-----------------------------------------------------+-+-'
```

3.8.7 SL_CLR

This write-only register clears the serial line interrupt request, the performance counter interrupt requests and the CRD interrupt request. The indicated bit must be written with a zero to clear the selected interrupt source.

Figure 3–9: Clear Serial Line Interrupt Register (SL_CLR)

```
 6               3 3 3         1 1 1 1 1 1 1 0 0 7 0 0 0 0 0 0 0
 3               3 2 1         6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
.-----------------+-+------------+-+------------+-+----------+-+---.
|                 |S|            |P|            |P|          |C|   |
|       IGN       |L|    IGN     |C|    IGN     |C|   IGN    |R|IGN|
|                 |C|            |1|            |0|          |D|   |
`-----------------+-+------------+-+------------+-+----------+-+---'
```

Table 3–13: SL_CLR

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

3.8.8 SL_RCV

The serial line receive register contains a single read-only bit used with the interrupt control registers and the sRomD_h and sRomClk_h pins to provide an on-chip serial line function. The RCV bit is functionally connected to the sRomD_h pin after the Icache is loaded from the external serial ROM. Reading the RCV bit can be used to receive external data one bit at a time under a software timing loop. A serial line interrupt is requested on detection of any transition on the receive line which sets the SL_REQ bit in the HIRR. Using a software timing loop, the RCV bit can be read to receive data one bit at a time. The serial line interrupt can be disabled by clearing the HIER register SL_ENA bit.

Figure 3–10: Serial Line Receive Register (SL_RCV)

```
 6                                              0 0 0   0
 3                                              4 3 2   0
.-----------------------------------------------+-+-----.
|                                               |R|     |
|                     RAZ                       |C| RAZ |
|                                               |V|     |
`-----------------------------------------------+-+-----'
```

3.8.9 ITBZAP

A write of any value to this IPR invalidates all eight ITB entries. It also resets the NLU pointer to its initial state. The ITBZAP register should only be written in PAL mode.

3.8.10 ITBASM

A write of any value to this IPR invalidates all ITB entries in which the ASM bit is equal to zero. The ITBASM register should only be written in PAL mode.

3.8.11 ITBIS

A write of any value to this IPR invalidates all eight ITB entries. It also resets the NLU pointer to its initial state. The ITBIS register should only be written in PAL mode.

3.8.12 PS

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

Figure 3–11: Processor Status Register (PS)

```
Write Format:
 6                                               0 0 0 0   0
 3                                               5 4 3 2   0
.-----------------------------------------------+-+-+-----.
|                                               |C|C|     |
|                      IGN                      |M|M| IGN |
|                                               |1|0|     |
`-----------------------------------------------+-+-+-----'
Read Format:
 6                      3 3 3                        0 0 0
 3                      5 4 3                        2 1 0
.-------------------------+-+-------------------------+-+-.
|                         |C|                         |C|R|
|            RAZ          |M|          RAZ            |M|A|
|                         |1|                         |0|Z|
`-------------------------+-+-------------------------+-+-'
```

3.8.13 EXC_SUM

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

In addition, the register containing the result of that operation is recorded in the exception register write mask IPR, as a single bit in a 64-bit field specifying registers F31-F0 and I31-I0. This IPR is visible only through the EXC_SUM register. The EXC_SUM register provides a one-bit window to the exception register write mask. Each read to the EXC_SUM shifts one bit in order F31-F0 then I31-I0. The read also clears the corresponding bit. Therefore, the EXC_SUM must be read 64 times to extract the complete mask and clear the entire register.

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

The write mask register bit clears three cycles after a read. Therefore, code intended to read the register must allow at least three cycles between reads to allow the clear and shift operation to complete in order to insure reading successive bits.

Figure 3–12: Exception Summary Register (EXC_SUM)

```
 6                        3 3 3         0 0 0 0 0 0 0 0 0 0
 3                        4 3 2         9 8 7 6 5 4 3 2 1 0
.---------------------------+-+-----------+-+-+-+-+-+-+-+---.
|                           |M|           |I|I|U|F|D|I|S| R |
|            RAZ            |S|    RAZ    |O|N|N|O|Z|N|W| A |
|                           |K|           |V|E|F|V|E|V|C| Z |
`---------------------------+-+-----------+-+-+-+-+-+-+-+---'
```

Table 3–14: EXC_SUM

Field	Type	Description
SWC	WA	Indicates Software Completion possible. The bit is set after a floating point instruction containing the /S modifier completes with an arithmetic trap and all previous floating point instructions that trapped since the last MTPR EXC_SUM also contained the /S modifier. The SWC bit is cleared whenever a floating point instruction without the /S modifier completes with an arithmetic trap. The bit remains cleared regardless of additional arithmetic traps until the register is written via an MTPR instruction. The bit is always cleared upon any MTPR write to the EXC_SUM register.
INV	WA	Indicates Invalid Operation.
DZE	WA	Indicates Divide by Zero.
FOV	WA	Indicates Floating Point Overflow.
UNF	WA	Indicates Floating Point Underflow.
INE	WA	Indicates Floating Inexact Error.
IOV	WA	Indicates Fbox Convert to Integer Overflow or Integer Arithmetic Overflow.
MSK	RC	Exception Register Write Mask IPR Window.

3.8.14 PAL Base Address Register (PAL_BASE)

The PAL base register is a read/write register containing the base address for PALcode. This register is cleared by hardware at reset.

Figure 3–13: PAL Base Register (PAL_BASE)

```
 6                         3 3                        1 1    0
 3                         4 3                        4 3    0
.--------------------------+-------------------------+-----.
|                          |                         | IGN |
|       IGN/RAZ            |     PAL_BASE[33..14]    |  /  |
|                          |                         | RAZ |
`--------------------------+-------------------------+-----'
```

3.8.15 HIRR

The Hardware Interrupt Request Register is a read-only register providing a record of all currently outstanding interrupt requests and summary bits at the time of the read. For each bit of the HIRR [5:0] there is a corresponding bit of the HIER (Hardware Interrupt Enable Register) that must be set to request an interrupt. In addition to returning the status of the hardware interrupt requests, a read of the HIRR returns the state of the software interrupt and AST requests. Note that a read of the HIRR may return a value of zero if the hardware interrupt was released before the read (passive release). The register guarantees that the HWR bit reflects the status as shown by the HIRR bits. All interrupt requests are blocked while executing in PALmode.

See section Section 2.3.3 for description of interrupt logic.

Figure 3–14: Hardware Interrupt Request Register (HIRR)

```
Read Format:

 6         3 3     2 2      1 1 1    0 0 0 0    0 0 0 0 0 0
 3         3 2     9 8      4 3 2    0 9 8 7    5 4 3 2 1 0
.-----------+-------+--------+-+------+-+-+------+-+-+-+-+-.
|           |U S E K|        |S|      |P|P|      |C|A|S|H|R|
|    RAZ    | ASTRR |  SIRR  |L| HIRR |C|C| HIRR |R|T|W|W|A|
|           | [3..0]| [15..1]|R|[2..0]|0|1|[5..3]|R|R|R|R|Z|
`-----------+-------+--------+-+------+-+-+------+-+-+-+-+-'
```

Table 3–15: HIRR

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

3.8.16 SIRR

The Software Interrupt Request Register is a read/write register used to control software interrupt requests. For each bit of the SIRR there is a corresponding bit of the SIER (Software Interrupt Enable Register) that must be set to request an interrupt. Reads of the SIRR return the complete set of interrupt request registers and summary bits, see the HIRR Table 3–15 for details. All interrupt requests are blocked while executing in PALmode.

Figure 3–15: Software Interrupt Request Register (SIRR)

```
Write Format:

 6           4 4           3 3                            0
 3           8 7           3 2                            0
.-------------+-------------+------------------------------.
|             |             |                              |
|     IGN     | SIRR[15..1] |              IGN             |
|             |             |                              |
`-------------+-------------+------------------------------'

Read Format:

 6         3 3     2 2      1 1 1    0 0 0 0    0 0 0 0 0 0
 3         3 2     9 8      4 3 2    0 9 8 7    5 4 3 2 1 0
.-----------+-------+---------+-+-------+-+-+-------+-+-+-+-+-.
|           |U S E K|         |S|       |P|P|       |C|A|S|H|R|
|     RAZ   | ASTRR |  SIRR   |L|  HIRR |C|C|  HIRR |R|T|W|W|A|
|           | [3..0]| [15..1] |R|[2..0] |0|1|[5..3] |R|R|R|R|Z|
`-----------+-------+---------+-+-------+-+-+-------+-+-+-+-+-'
```

3.8.17 ASTRR

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

All interrupt requests are blocked while executing in PALmode. Reads of the ASTRR return the complete set of interrupt request registers and summary bits, see the HIRR Table 3–15 for details.

Figure 3–16: Asynchronous Trap Request Register (ASTRR)

```
Write Format:

 6       5 5 5 4 4 4                                       0
 3       2 1 0 9 8 7                                       0
.---------+-+-+-+-+----------------------------------------.
|         |U|S|E|K|                                        |
|   IGN   |A|A|A|A|                  IGN                   |
|         |R|R|R|R|                                        |
`---------+-+-+-+-+----------------------------------------'

Read Format:

 6         3 3     2 2      1 1 1    0 0 0 0    0 0 0 0 0 0
 3         3 2     9 8      4 3 2    0 9 8 7    5 4 3 2 1 0
.-----------+-------+---------+-+-------+-+-+------+-+-+-+-+-.
|           |U S E K|         |S|       |P|P|      |C|A|S|H|R|
|    RAZ    | ASTRR |  SIRR   |L| HIRR  |C|C| HIRR |R|T|W|W|A|
|           | [3..0]| [15..1]|R|[2..0]|0|1|[5..3]|R|R|R|R|Z|
`-----------+-------+---------+-+-------+-+-+------+-+-+-+-+-'
```

3.8.18 Hardware Interrupt Enable - HIER

The Hardware Interrupt Enable Register is a read/write register. It is used to enable corresponding bits of the HIRR requesting interrupt. The PC0, PC1, SLE and CRE bits of this register enable the performance counters, serial line and correctable read interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits, as with the reads of the interrupt request IPRs, reads of the HIER return the complete set of interrupt enable registers, see the HIRR Table 3–15 for details.

Figure 3–17: Hardware Interrupt Enable Register (HIER)

```
Write Format:

 6                        3 3 3   1 1 1         0 0 0     0   0
 3                        3 2 1   6 5 4         9 8 7     2   0
.---------------------------+-+-----+-+------------+-+-----+-+---.
|                           |S|     |P|            |P|     |C|   |
|            IGN            |L| IGN |C| HIER[5..0] |C| IGN |R|   |
|                           |E|     |1|            |0|     |E|   |
`---------------------------+-+-----+-+------------+-+-----+-+---'

Read Format:

 6        3 3 3 3 2 2          1 1 1    1 0 0 0    0 0 0     0
 3        3 2 1 0 9 8          4 3 2    0 9 8 7    5 4 3     0
.----------+-+-+-+-+-------------+-+------+-+-+------+-+-------.
|          |U|S|E|K|             |S|      |P|P|      |C|       |
|    RAZ   |A|A|A|A| SIER[15..1] |L| HIER |C|C| HIER |R|  RAZ  |
|          |E|E|E|E|             |E|[2..0]|0|1|[5..3]|E|       |
`----------+-+-+-+-+-------------+-+------+-+-+------+-+-------'
```

Table 3–16: HIER

Field	Type	Description
CRR	RO	CRD correctable read error interrupt enable.
HIER[5..0]	RO	Corresponds to pins Irq_h[5..0].
SIER[15..1]	RO	Corresponds to software interrupt requests 15 thru 1.
ASTRR[3..0]	RO	Corresponds to AST enable three thru zero (USEK).
PC1	RO	Performance counter 1 interrupt enable.
PC0	RO	Performance counter 0 interrupt enable.
SLE	RO	Serial line interrupt enable. (See also SL_RCV, SL_XMIT, and SL_CLR).
CRE	RO	CRD correctable read error interrupt enable. This interrupt request is cleared via the SL_CLR register.

3.8.19 SIER

The Software Interrupt Enable Register is a read/write register. It is used to enable corresponding bits of the SIRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits, as with the reads of the interrupt request IPRs, reads of the SIER return the complete set of interrupt enable registers, see the HIRR Table 3–15 for details.

Figure 3–18: Software Interrupt Enable Register (SIER)

```
Write Format:

 6           4 4           3 3                           0
 3           8 7           3 2                           0
.-------------+-------------+-----------------------------.
|             |             |                             |
|     IGN     | SIER[15..1] |            IGN              |
|             |             |                             |
`-------------+-------------+-----------------------------'

Read Format:

 6        3 3 3 3 2 2          1 1 1    1 0 0 0    0 0 0     0
 3        3 2 1 0 9 8          4 3 2    0 9 8 7    5 4 3     0
.----------+-+-+-+-+-------------+-+------+-+-+------+-+-------.
|          |U|S|E|K|             |S|      |P|P|      |C|       |
|    RAZ   |A|A|A|A| SIER[15..1] |L| HIER |C|C| HIER |R|  RAZ  |
|          |E|E|E|E|             |E|[2..0]|0|1|[5..3]|E|       |
`----------+-+-+-+-+-------------+-+------+-+-+------+-+-------'
```

3.8.20 ASTER

The AST Interrupt Enable Register is a read/write register. It is used to enable corresponding bits of the ASTRR requesting interrupts. There is a one-to-one correspondence between the interrupt requests and enable bits, as with the reads of the interrupt request IPRs, reads of the ASTER return the complete set of interrupt enable registers, see the HIRR Table 3–15 for details.

Figure 3–19: AST Interrupt Enable Register (ASTER)

```
Write Format:

 6       5 5 5 4 4 4                                     0
 3       2 1 0 9 8 7                                     0
.---------+-+-+-+-+---------------------------------------.
|         |U|S|E|K|                                       |
|   IGN   |A|A|A|A|                  IGN                  |
|         |E|E|E|E|                                       |
`---------+-+-+-+-+---------------------------------------'

Read Format:

 6        3 3 3 3 2 2          1 1 1    1 0 0 0    0 0 0     0
 3        3 2 1 0 9 8          4 3 2    0 9 8 7    5 4 3     0
.----------+-+-+-+-+-------------+-+------+-+-+------+-+-------.
|          |U|S|E|K|             |S|      |P|P|      |C|       |
|    RAZ   |A|A|A|A| SIER[15..1] |L| HIER |C|C| HIER |R|  RAZ  |
|          |E|E|E|E|             |E|[2..0]|0|1|[5..3]|E|       |
`----------+-+-+-+-+-------------+-+------+-+-+------+-+-------'
```

3.8.21 SL_XMIT

The Serial Line Transmit register contains a single write-only bit used with the interrupt control registers and the sRomD_h and sRomClk_h pins to provide an on-chip serial line function. The TMT bit is functionally connected to the sRomClk_h pin after the Icache is loaded from the external serial ROM. Writing the TMT bit can be used to transmit data off chip one bit at a time under a software timing loop.

Figure 3–20: Serial Line Transmit Register (SL_XMIT)

```
 6                                              0 0 0     0
 3                                              5 4 3     0
.----------------------------------------------+-+-------.
|                                              |T|       |
|                      IGN                     |M|  IGN  |
|                                              |T|       |
`----------------------------------------------+-+-------'
```

3.9 Abox IPRs

3.9.1 TB_CTL

The granularity hint (GH) field selects between the 21064 TB page mapping sizes. There are two sizes in the ITB and all four sizes in the DTB. When only two sizes are provided, the large-page-select (GH=11(bin)) field selects the largest mapping size (512 * 8Kbytes) and all other values select the smallest (8Kbyte) size. The GH field affects both reads and writes to the ITB and DTB.

Figure 3–21: TB_CTL Register

```
 6                                           0 00 0       0
 3                                           7 65 4       0
.-------------------------------------------+--+---------.
|                                           |  |         |
|                      IGN                  |GH|   IGN   |
|                                           |  |         |
`-------------------------------------------+--+---------'
```

3.9.2 DTB_PTE

The DTB PTE register is a read/write register representing the 32-entry small-page and 4-entry large-page DTB page table entries. The entry to be written is chosen by a not-last-used (NLU) algorithm implemented in hardware and the value in the TB_CTL register. Writes to the DTB_PTE use the memory format bit positions as described in the Alpha Architecture Handbook with the exception that some fields are ignored. In particular the valid bit is not represented in hardware.

To insure the integrity of the DTBs, the DTB's tag array is updated simultaneously from the internal tag register when the DTB_PTE register is written. 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, then a second instruction reading from the DTB_PTE_TEMP register returns the PTE entry to the register file. Reading or writing the DTB_PTE register increments the TB entry pointer of the DTB indicated by the TB_CTL IPR which allows reading the entire set of DTB PTE entries.

Figure 3–22: DTB_PTE Register

```
Small Page Format:

 6 5 5             3 3       1 1 1 1 1 1 1 0 0 00 0 0 0 0 0
 3 3 2             2 1       6 5 4 3 2 1 0 9 8 75 4 3 2 1 0
.---+---------------+---------+-+-+-+-+-+-+-+-+---+-+-+-+-+-.
|   |               |         |U|S|E|K|U|S|E|K|I  |A|I|F|F|I|
|IGN|  PFN[33..13]  |   IGN   |W|W|W|W|R|R|R|R|G  |S|G|O|O|G|
|   |               |         |E|E|E|E|E|E|E|E|N  |M|N|W|R|N|
`---+---------------+---------+-+-+-+-+-+-+-+-+---+-+-+-+-+-'

Large Page Format:

 6 5 5             4 4       1 1 1 1 1 1 1 0 0 00 0 0 0 0 0
 3 3 2             1 0       6 5 4 3 2 1 0 9 8 75 4 3 2 1 0
.---+---------------+---------+-+-+-+-+-+-+-+-+---+-+-+-+-+-.
|   |               |         |U|S|E|K|U|S|E|K|I  |A|I|F|F|I|
|IGN|  PFN[33..22]  |   IGN   |W|W|W|W|R|R|R|R|G  |S|G|O|O|G|
|   |               |         |E|E|E|E|E|E|E|E|N  |M|N|W|R|N|
`---+---------------+---------+-+-+-+-+-+-+-+-+---+-+-+-+-+-'
```

3.9.3 DTB_PTE_TEMP

The DTB_PTE_TEMP register is a read-only holding register for DTB_PTE read data. Reads of the DTB_PTE require two instructions to return the data to the register file. The first reads the DTB_PTE register to the DTB_PTE_TEMP register. The second returns the DTB_PTE_TEMP register to the integer register file.

Figure 3–23: DTB_PTE_TEMP Register

```
Small Page Format:

 6           3 3 3                1 1 1 1 0 0 0 0 0 0 0 00
 3           5 4 3                3 2 1 0 9 8 7 6 5 4 3 20
.--------------+-+-----------------+-+-+-+-+-+-+-+-+-+-+--.
|              |A|                 |U|S|E|K|U|S|E|K|F|F| R|
|       RAZ    |S|   PFN[33..13]   |R|R|R|R|W|W|W|W|O|O| A|
|              |M|                 |E|E|E|E|E|E|E|E|W|R| Z|
`--------------+-+-----------------+-+-+-+-+-+-+-+-+-+-+--'

Large Page Format:

 6           3 3 3           2 2  1 1 1 1 0 0 0 0 0 0 0 00
 3           5 4 3           2 2  3 2 1 0 9 8 7 6 5 4 3 20
.--------------+-+-------------+----+-+-+-+-+-+-+-+-+-+-+--.
|              |A|             | I  |U|S|E|K|U|S|E|K|F|F| R|
|       RAZ    |S| PFN[33..22] | G  |R|R|R|R|W|W|W|W|O|O| A|
|              |M|             | N  |E|E|E|E|E|E|E|E|W|R| Z|
`--------------+-+-------------+----+-+-+-+-+-+-+-+-+-+-+--'
```

3.9.4 MM_CSR

When D-stream faults occur the information about the fault is latched and saved in the MM_CSR register. The VA and MM_CSR registers are locked against further updates until software reads the VA register. Palcode must explicitly unlock this register whenever its entry point was higher in priority than a DTB miss. MM_CSR bits are only modified by hardware when the register is not locked and a memory management error or a DTB miss occurs. The MM_CSR is unlocked after reset.

Figure 3–24: MM_CSR Register

```
 6                            1 1        0 0     0 0 0 0 0
 3                            5 4        9 8     4 3 2 1 0
.--------------------------------+----------+------+-+-+-+-.
|                                |          |      |F|F|A|W|
|               RAZ              |  OPCODE  |  RA  |O|O|C|R|
|                                |          |      |W|R|V| |
`--------------------------------+----------+------+-+-+-+-'
```

Table 3–17: MM_CSR

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

3.9.5 Virtual Address Register (VA)

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

3.9.6 DTBZAP

A write of any value to this IPR invalidates all 32 small-page and four large-page DTB entries. It also resets the NLU pointer to its initial state.

3.9.7 DTBASM

A write of any value to this IPR invalidates all 32 small-page and 4 large-page DTB entries in which the ASM bit is equal to zero.

3.9.8 DTBIS

If the virtual address in the RB field is mapped in either the small-page or large-page DTB then those entries are invalidated.

3.9.9 FLUSH_IC

A write of any value to this pseudo-IPR flushes the entire instruction cache.

3.9.10 FLUSH_IC_ASM

A write of any value to this pseudo-IPR invalidates all Icache blocks in which the ASM bit is clear.

3.9.11 Abox Control Register (ABOX_CTL)

Figure 3–25: Abox Control Register (ABOX_CTL)

```
 63           12 11 10  9  8  7  6  5  4  3  2  1  0
.--------------+--+--+--+--+--+--+--+--+--+--+--+--.
|              |  |  |           |  |  |  |  |  |  |
|      IGN     |WO|WO|   IGN     |  |  |  |WO|WO|WO|WO|
|              |  |  |           |  |  |  |  |  |  |
`--------------+--+--+--+--+--+--+--+--+--+--+--+--'
                 |  |           |  |  |  |  |  |  |
                 |  |           |  |  |  |  |  |  |
                 |  |           |  |  |  |  |  |  `-> WB_DIS
                 |  |           |  |  |  |  |  `----> MCHK_EN
                 |  |           |  |  |  |  `-------> CRD_EN
                 |  |           |  |  |  `----------> IC_SBUF_EN
                 |  |           |  |  `-------------> SPE_1
                 |  |           |  `----------------> SPE_2
                 |  |           `-------------------> EMD_EN
                 |  `-------------------------------> DC_ENA
                 `----------------------------------> DC_FHIT
```

Table 3–18: Abox Control Register

Field	Type	Description
WB_DIS	WO,0	Write Buffer unload Disable. When set, this bit prevents the write buffer from sending write data to the BIU. It should be set for diagnostics only.
MCHK_EN	WO,0	Machine Check Enable. When this bit is set the Abox generates a machine check when errors which are not correctable by the hardware are encountered. When this bit is cleared, uncorrectable errors do not cause a machine check, but the BIU_STAT, DC_STAT, BIU_ADDR, FILL_ADDR and DC_ADDR registers are updated and locked when the errors occur.
CRD_EN	WO,0	Corrected read data interrupt enable. When this bit is set the Abox generates an interrupt request whenever a pin bus transaction is terminated with a cAck_h code of SOFT_ERROR.
IC_SBUF_EN	WO,0	Icache stream buffer enable. When set, this bit enables operation of a single entry Icache stream buffer.

Table 3–18 (Cont.): Abox Control Register

Field	Type	Description
SPE_1	WO,0	This bit, when set, enables one-to-one super page mapping of D-stream virtual addresses with VA<33:13> directly to physical addresses PA<33:13>, if virtual address bits VA<42:41> = 2. Virtual address bits VA(<)40:34> are ignored in this translation. Access is only allowed in kernel mode.
SPE_2	WO,0	This bit, when set, enables one-to-one super page mapping of D-stream virtual addresses with VA<42:30> = 1FFE(Hex) to physical addresses with PA<33:30> = 0(Hex). Access is only allowed in kernel mode.
EMD_EN	WO,0	Limited hardware support is provided for big endian data formats via bit <6><LITERAL> of the ABOX_CTL register. This bit, when set, inverts physical address bit <LITERAL><2> for all D-stream references. It is intended that chip endian mode be selected during initialization PALcode only.
DC_EN	WO,0	Dcache enable. When clear, this bit disables and flushes the Dcache. When set, this bit enables the Dcache.
DC_FHIT	WO,0	Dcache force hit. When set, this bit forces all D-stream references to hit in the Dcache. This bit takes precedence over DC_EN, i.e. when DC_FHIT is set and DC_EN is clear all D-stream references hit in the Dcache.

3.9.12 ALT_MODE

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

Figure 3–26: Alternate Processor Mode Register (ALT_MODE)

```
 6                                             0 00 0       0
 3                                             5 43 2       0
.---------------------------------------------+--+--------.
|                                             |A |        |
|                     IGN                     |M |  IGN   |
|                                             |  |        |
`---------------------------------------------+--+--------'
```

Table 3–19: ALT Mode

ALT_ MODE[4..3]	Mode
0 0	Kernel
0 1	Executive
1 0	Supervisor
1 1	User

3.9.13 Cycle Counter (CC)

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

3.9.14 Cycle Counter Control Register (CC_CTL)

HW_MTPR Rn,CC_CTL writes CC[31..0] with the value held in Rn[31..0], and CC[63..32] are not changed. CC[3..0] must be written with zero. If Rn[32] is set then the counter is enabled, otherwise the counter is disabled. CC_CTL is a write-only IPR.

3.9.15 Bus Interface Unit Control Register (BIU_CTL)

Figure 3–27: Bus Interface Unit Control Register (BIU_CTL)

```
 6 3 3 3 3 3 3 2 2              1 1 1
 3 7 6 5 2 1 0 8 7              3 2 1      8 7       4 3 2 1 0
.---+-+---+-+-+---+----------------+-+---------+---------+-+-+-+-.
|   | |   | | |   |                |I|         |         | | | | |
|IGN| |   | | |   |BC_WE_CTL[15..1]|G|BC_WR_SPD|BC_RD_SPD| | | | |
|   | |   | | |   |                |N|         |         | | | | |
`---+-+---+-+-+---+----------------+-+---------+---------+-+-+-+-'
     |   |  |  |                                          | | | |
     |   |  |  |                                          | | | `--> BC_ENA
     |   |  |  |                                          | | `----> ECC
     |   |  |  |                                          | `------> OE
     |   |  |  |                                          `--------> BC_FHIT
     |   |  |  `-----------------------------------------------------> BC_SIZE
     |   |  `--------------------------------------------------------> BAD_TCP
     |   `-----------------------------------------------------------> BC_PA_DIS
     `---------------------------------------------------------------> BAD_DP
```

Table 3–20: BIU Control Register

Field	Type	Description
BC_EN	WO,0	External cache enable. When clear, this bit disables the external cache. When the external cache is disabled the BIU does not probe the external cache tag store for read and write references; it launches a request on cReq_h immediately.
ECC	WO	When this bit is set the 21064 generates/expects ECC on the check_h pins. When this bit is clear the chip generates/expects parity on four of the check_h pins.
OE	WO,0	When this bit is set the 21064 does not assert its chip enable pins during RAM write cycles, thus enabling these pins to be connected to the output enable pins of the cache RAMs.
BC_FHIT	WO,0	External cache force hit. When this bit is set and BC_EN is also set, all pin bus READ_BLOCK and WRITE_BLOCK transactions are forced to hit in the external cache. Tag and tag control parity are ignored when the BIU operates in this mode. BC_EN takes precedence over BC_FHIT. When BC_EN is clear and BC_FHIT is set no tag probes occur and external requests are directed to the cReq_h pins. Note that the BC_PA_DIS field takes precedence over the BC_FHIT bit.
BC_RD_SPD	WO	External cache read speed. This field indicates to the BIU the read access time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less the read access time of the external cache RAMs. Access times for reads must be in the range 16..3 CPU cycles, which means the values for the BC_RD_SPD field are in the range of 15..2. BC_RD_SPD are not initialized on reset and must be explicitly written before enabling the external cache.
BC_WR_SPD	WO	External cache write speed. This field indicates to the BIU the write cycle time of the RAMs used to implement the off-chip external cache, measured in CPU cycles. It should be written with a value equal to one less the write cycle time of the external cache RAMs. Access times for writes must be in the range 16..2 CPU cycles, which means the values for the BC_RD_SPD field are in the range of 15..1. BC_WR_SPD are not initialized on reset and must be explicitly written before enabling the external cache.

Table 3–20 (Cont.): BIU Control Register

Field	Type	Description
BC_WE_CTL	WO	External cache write enable control. This field is used to control the timing of the write enable and chip enable pins during writes into the data and tag control RAMs. It consists of 15 bits, where each bit determines the value placed on the write enable and chip enable pins during a given CPU cycle of the RAM write access. When a given bit of BC_WE_CTL is set, the write enable and chip enable pins are asserted during the corresponding CPU cycle of the RAM access. BC_WE_CTL[0] (bit 13 in BIU_CTL) corresponds to the second cycle of the write access, BC_WE_CTL[1] (bit 14 in BIU_CTL) to the third CPU cycle, and so on. The write enable pins will never be asserted in the first CPU cycle of a RAM write access. Unused bits in the BC_WE_CTL field must be written with zeros. BC_WE_CTL is not initialized on reset and must be explicitly written before enabling the external cache.
BC_SIZE	WO	This field is used to indicate the size of the external cache. BC_SIZE is not initialized on reset and must be explicitly written before enabling the external cache. See Table 3–19 for the encodings.
BAD_TCP	WO,0	When set, BAD_TCP causes the 21064 to write bad parity into the tag control RAM whenever it does a fast external RAM write.
BC_PA_DIS	WO	This 4-bit field may be used to prevent the CPU chip from using the external cache to service reads and writes based upon the quadrant of physical address space which they reference. The correspondence between this bit field and the physical address space is shown in Table 3–20. When a read or write reference is presented to the BIU the values of BC_PA_ DIS, BC_ENA and physical address bits [33:32] together determine whether to attempt to use the external cache to satisfy the reference. If the external cache is not to be used for a given reference the BIU does not probe the tag store, and makes the appropriate system request immediately. The value of BC_PA_DIS has NO impact on which portions of the physical address space may be cached in the primary caches. System components control this via the RDACK field of the pin bus. BC_PA_DIS are not initialized by reset.
BAD_DP	WO	When set, BAD_DP causes the 21064 to invert the value placed on bits [0],[7],[14] and [21] of the check_h[27..0] field during off-chip writes. This produces bad parity when the 21064 is in parity mode, and bad check bit codes when it is in ECC mode.

Table 3–21: BC_SIZE

BC_SIZE	Size
0 0 0	128 Kbytes
0 0 1	256 Kbytes
0 1 0	512 Kbytes
0 1 1	1 Mbytes
1 0 0	2 Mbytes
1 0 1	4 Mbytes
1 1 0	8 Mbytes

Table 3–22: BC_PA_DIS

BIU_CTL bits	Physical Address
[32]	PA[33..32] = 0
[33]	PA[33..32] = 1
[34]	PA[33..32] = 2
[35]	PA[33..32] = 3

3.10 PAL_TEMPs

The CPU chip contains 32 registers which are accessible via HW_MxPR instructions. These registers provide temporary storage for PALcode.

3.10.1 DC_STAT

The DC_STAT is a read-only IPR.

Overview:

When an external ECC or parity error is recognized during a primary cache fill operation, the DC_STAT register is locked against further updates. In the event that the cache fill was due to D-stream activity the contents of this register may be used by PAL code in conjunction with information latched elsewhere (see Section 3.12) to recover from some single-bit ECC errors. DC_STAT is unlocked when DC_ADDR is read.

Figure 3–28: Data Cache Status Register (DC_STAT)

```
 63           15 14 13 12 11 10   9 8         4   3 2      0
.--------------+--+--+--+--+--+--+----------+--+--------.
|              |  |  |  |  |  |  |          |  |        |
|     RAZ      |RO|RO|RO|RO|RO|RO|    RO    |RO| R A Z  |
|              |  |  |  |  |  |  |          |  |        |
`--------------+--+--+--+--+--+--+----------+--+--------'
                 |  |  |  |  |  |     |       |
                 |  |  |  |  |  |     |       `-----------> DC_HIT
                 |  |  |  |  |  |     `-------------------> RA
                 |  |  |  |  |  `-------------------------> INT
                 |  |  |  |  `----------------------------> LW
                 |  |  |  `-------------------------------> VAX_FP
                 |  |  `----------------------------------> LOCK
                 |  `-------------------------------------> STORE
                 `----------------------------------------> SEO
```

Table 3–23: Dcache Status Register

Field	Type	Description
DC_HIT	RO	This bit indicates whether the last load or store instruction processed by the Abox hit, (DC_HIT set) or missed, (DC_HIT clear) the Dcache. Loads that miss the Dcache may be completed without requiring external reads. e.g. pending fill or pending store hits.
SEO	RO	Second Error Occurred. Set when an error which would normally lock the DC_STAT register occurs while the DC_STAT register is already locked.

The following bits are only meaningful if the FILL_ECC or FILL_DPERR bit in the BIU_STAT register is set.

Table 3–24: Dcache STAT Error Modifiers

Field	Type	Description
RA	RO	The Ra field of the instruction which resulted in the error.
INT	RO	When set, indicates an integer load or store.
LW	RO	When set, indicates that the data length of the load or store was longword.
VAX_FP	RO	When INT is clear, this bit is set to indicate that a VAX floating point format load or store caused the error.
LOCK	RO	This bit is set to indicate that the error stemmed from a LDLL, LDQL, STLC, or STQC instruction.
STORE	RO	This bit is set to indicate that the error stemmed from a store instruction.

3.10.2 DC_ADDR

DC_ADDR is a pseudo-register used for unlocking DC_STAT. DC_STAT and DC_ADDR are unlocked when DC_ADDR is read.

3.10.3 BIU_STAT

BIU_STAT is a read-only IPR.

When one of BIU_HERR, BIU_SERR, BC_TPERR or BC_TCPERR is set, BIU_STAT[6..0] are locked against further updates, and the address associated with the error is latched and locked in the BIU_ADDR register. BIU_STAT[6..0] and BIU_ADDR are also spuriously locked when FILL_ECC or FILL_DPERR is set. BIU_STAT[7..0] and BIU_ADDR are unlocked when the BIU_ADDR register is read.

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

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

Figure 3–29: Bus Interface Unit Status Register (BIU_STAT)

```
 63       14 13 12 11 10  9  8  7 6   5  4  3  2  1  0
.--------+--+-----+--+--+--+--+--+--------+--+--+--+--.
|        |  |     |  |  | R|  |  |        |  |  |  |  |
|   RAZ  |RO| RO  |RO|RO| A|RO|RO|   RO   |RO|RO|RO|RO|
|        |  |     |  |  | Z|  |  |        |  |  |  |  |
`--------+--+-----+--+--+--+--+--+--------+--+--+--+--'
           |   |    |  |     |  |     |     |  |  |  |
           |   |    |  |     |  |     |     |  |  |  `-> BIU_HERR
           |   |    |  |     |  |     |     |  |  `----> BIU_SERR
           |   |    |  |     |  |     |     |  `-------> BC_TPERR
           |   |    |  |     |  |     |     `----------> BC_TCPERR
           |   |    |  |     |  |     `----------------> BIU_CMD
           |   |    |  |     |  `----------------------> BIU_SEO
           |   |    |  |     `-------------------------> FILL_ECC
           |   |    |  `-------------------------------> FILL_DPERR
           |   |    `----------------------------------> FILL_IRD
           |   `---------------------------------------> FILL_QW
           `-------------------------------------------> FILL_SEO
```

Table 3–25: BIU STAT

Field	Type	Description
BIU_HERR	RO	This bit, when set, indicates that an external cycle was terminated with the cAck_h pins indicating HARD_ERROR.

Table 3–25 (Cont.): BIU STAT

Field	Type	Description
BIU_SERR	RO	This bit, when set, indicates that an external cycle was terminated with the cAck_h pins indicating SOFT_ERROR.
BC_TPERR	RO	This bit, when set, indicates that a external cache tag probe encountered bad parity in the tag address RAM.
BC_TCPERR	RO	This bit, when set, indicates that a external cache tag probe encountered bad parity in the tag control RAM.
BIU_CMD	RO	This field latches the cycle type on the cReq_h pins when a BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR error occurs.
BIU_SEO	RO	This bit, when set, indicates that an external cycle was terminated with the cAck_h pins indicating HARD_ERROR or that a an external cache tag probe encountered bad parity in the tag address RAM or the tag control RAM while one of BIU_HERR, BIU_SERR, BC_TPERR, or BC_TCPERR was already set.
FILL_ECC	RO	ECC error. This bit, when set, indicates that primary cache fill data received from outside the CPU chip contained an ECC error.
FILL_DPERR	RO	Fill Parity Error. This bit when set, indicates that the BIU received data with a parity error from outside the CPU chip while performing either a Dcache or Icache fill. FILL_DPERR is only meaningful when the CPU chip is in parity mode, as opposed to ECC mode.
FILL_IRD	RO	This bit is only meaningful when either FILL_ECC or FILL_DPERR is set. FILL_IRD is set to indicate that the error which caused FILL_ECC or FILL_DPERR to set occurred during an Icache fill and clear to indicate that the error occurred during a Dcache fill.
FILL_QW	RO	This field is only meaningful when either FILL_ECC or FILL_DPERR is set. FILL_QW identifies the quadword within the hexaword primary cache fill block which caused the error. It can be used together with FILL_ADDR[33..5] to get the complete physical address of the bad quadword.
FILL_SEO	RO	This bit, when set, indicates that a primary cache fill operation resulted in either an uncorrectable ECC error or in a parity error while FILL_ECC or FILL_DPERR was already set.

3.10.4 BIU_ADDR

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

BIU_ADDR[33..5] contain the values of adr_h[33..5] associated with the pin bus transaction which resulted in the error indicated in BIU_STAT[7..0].

If the BIU_CMD field of the BIU_STAT register indicates that the transaction which received the error was READ_BLOCK or LDx/L, then BIU_STAT[4..2] are UNPREDICTABLE. If the BIU_CMD field of the BIU_STAT register encodes any pin bus command other than READ_BLOCK or LDx/L, then BIU_ADDR[4..2] will contain zeros. BIU_ADDR[63..34] and BIU_ADDR[1..0] always read as zero.

3.10.5 FILL_ADDR

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

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

If the FILL_IRD bit of the BIU_STAT register is clear, indicating that the error occurred during a D-stream cache fill, then FILL_ADDR[4..2] contain bits [4..2] of the physical address generated by the load instruction which triggered the cache fill. If FILL_IRD is set, then FILL_ADDR[4..2] are UNPREDICTABLE. FILL_ADDR[63..34] and FILL_ADDR[1..0] read as zero.

3.10.6 FILL_SYNDROME

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

If the chip is in ECC mode and an ECC error is recognized during a primary cache fill operation, the syndrome bits associated with the bad quadword are locked in the FILL_SYNDROME register. FILL_SYNDROME[6..0] contain the syndrome associated with the lower longword of the quadword, and FILL_SYNDROME[13..7] contain the syndrome associated with the higher longword of the quadword. A syndrome value of zero means that no errors where found in the associated longword. See Table 3–24 for a list of syndromes associated with correctable single-bit errors. The FILL_SYNDROME register is unlocked when the FILL_ADDR register is read.

If the chip is in parity mode and a parity error is recognized during a primary cache fill operation, the FILL_SYNDROME register indicates which of the longwords in the quadword got bad parity. FILL_SYNDROME[0] is set to indicate that the low longword was corrupted, and FILL_SYNDROME[7] is set to indicate that the high longword was corrupted. FILL_SYNDROME[13..8] and [6..1] are RAZ in parity mode.

Figure 3–30: FILL_SYNDROME Register

```
 6                                  1 1         0 0         0
 3                                  4 3         7 6         0
.-----------------------------------+----------+----------.
|                                   |          |          |
|                 RAZ               | HI[6..0] | LO[6..0] |
|                                   |          |          |
`-----------------------------------+----------+----------'
```

Table 3–26: Syndromes for Single-Bit Errors

Data Bit	Syndrome(Hex)	Check Bit	Syndrome(Hex)
00	4F	00	01
01	4A	01	02
02	52	02	04
03	54	03	08
04	57	04	10
05	58	05	20
06	5B	06	40
07	5D		
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		

Table 3–26 (Cont.): Syndromes for Single-Bit Errors

Data Bit	Syndrome(Hex)	Check Bit	Syndrome(Hex)
22	1A		
23	1C		
24	62		
25	64		
26	67		
27	68		
28	6B		
29	6D		
30	70		
31	75		

3.10.7 BC_TAG

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

Successive HW_MFPRs from the BC_TAG register must be separated by at least one null cycle.

Figure 3–31: Backup Cache Tag Register (BC_TAG)

```
 63          23 22 21                  5  4  3  2  1  0
.-------------+--+--------------------+--+--+--+--+--.
|             |  |                    |  |  |  |  |  |
|     RAZ     |RO|    TAG [33..17]    |RO|RO|RO|RO|RO|
|             |  |                    |  |  |  |  |  |
`-------------+--+--------------------+--+--+--+--+--'
                |                       |  |  |  |  |
                |                       |  |  |  |  `---> HIT
                |                       |  |  |  `------> TAGCTL_P
                |                       |  |  `---------> TAGCTL_D
                |                       |  `------------> TAGCTL_S
                |                       `---------------> TAGCTL_V
                `---------------------------------------> TAG_P
```

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

3.11 ECC Error Correction

When in ECC mode, the 21064 generates longword ECC on writes, and checks ECC on reads. The 21064 does not include hardware to correct single-bit errors, however.

When an ECC error is recognized during a Dcache fill the BIU places the affected fill block into the Dcache unchanged, validates the block and posts a machine check. The load instruction which triggered the Dcache fill is completed by writing the requested longword(s) into the register file. The longword(s) read by the load instruction may or not have been the cause of the error, but a machine check is posted either way. The Ibox will react to the machine check by aborting instruction execution before any instruction issued subsequent to the load could overwrite the register containing the load data, and vectoring to the PAL code machine check handler. Sufficient state is retained in various status registers (see Section 3.12) for PAL code to determine whether the error affects the longword(s) read by the load instruction, and whether the error is correctable. In any event, PAL code must explicitly flush the Dcache. If the longword containing the error was written into the register file, PAL code must either correct it and restart the machine, or report an uncorrectable hardware error to the operating system. Independent of whether the failing longword was read by the load instruction, PAL may scrub memory by explicitly reading the longword with the physical/lock variant of the HW_LD instruction, flipping the necessary bit, and writing the longword with the physical/conditional variant of the HW_ST instruction. Note that when PAL rereads the affected longword the hardware may report no errors, indicating that the longword has been overwritten.

When an ECC error occurs during an Icache fill the BIU places the affected fill block into the Icache unchanged, validates the block and posts a machine check. The Ibox will vector to the PAL code machine check handler before it executes any of the instructions in the bad block. PAL code may then flush the Icache and scrub memory as described above.

As compared with hardware error correction, this approach is vulnerable to single-bit errors which may occur during I-stream reads of the PAL code machine check handler, to single-bit errors which occur in multiple quadwords of a cache fill block, and to single-bit errors which occur as a result of multiple silo'ed load misses.

3.12 Error Flows

The following sections give a summary of the hardware flows for various error conditions.

3.12.1 I-stream ECC error

- data put into Icache unchanged, block gets validated
- machine check
- BIU_STAT: FILL_ECC, FILL_IRD set, FILL_SEO set if multiple errors occurred
- FILL_ADDR[33..5] & BIU_STAT[FILL_QW] give bad QW's address
- FILL_SYNDROME contains syndrome bits associated with failing quadword
- BIU_ADDR, BIU_STAT[6..0] locked - contents are UNPREDICTABLE
- DC_STAT locked - contents are UNPREDICTABLE
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

3.12.2 D-stream ECC error

- data put into Dcache unchanged, block gets validated
- machine check
- BIU_STAT: FILL_ECC set, FILL_IRD clear, FILL_SEO set if multiple errors occurred
- FILL_ADDR[33..5] & BIU_STAT[FILL_QW] give bad QW's address
- FILL_ADDR[4..2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL_SYNDROME contains syndrome bits associated with failing quadword
- BIU_ADDR, BIU_STAT[6..0] locked - contents are UNPREDICTABLE
- DC_STAT: RA identifies register which holds the bad data. LW,LOCK,INT,VAX_FP identify type of load instruction
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

3.12.3 BIU: tag address parity error

- recognized at end of tag probe sequence
- lookup uses predicted parity so transaction misses the external cache
- BC_TAG holds results of external cache tag probe
- machine check
- BIU_STAT: BC_TPERR set
- BIU_ADDR holds address

3.12.4 BIU: tag control parity error

- recognized at end of tag probe sequence
- transaction forced to miss external cache
- BC_TAG holds results of external cache tag probe
- machine check
- BIU_STAT: BC_TCPERR set
- BIU_ADDR holds address

3.12.5 BIU: system external transaction terminated with CACK_SERR

- CRD interrupt.
- BIU_STAT: BIU_SERR set, BIU_CMD holds cReq_h[2..0].
- BIU_ADDR holds address.

3.12.6 BIU: system transaction terminated with CACK_HERR

- machine check
- BIU_STAT: BIU_HERR set, BIU_CMD holds cReq_h[2..0]
- BIU_ADDR holds address

3.12.7 BIU: I-stream parity error (parity mode only)

- data put into Icache unchanged, block gets validated
- machine check
- BIU_STAT: FILL_DPERR set, FILL_IRD set
- FILL_ADDR[33..5] & BIU_STAT[FILL_QW] give bad QW's address
- FILL_SYNDROME identifies failing longword(s)
- BIU_ADDR, BIU_STAT[6..0] locked - contents are UNPREDICTABLE
- DC_STAT locked - contents are UNPREDICTABLE

- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

3.12.8 BIU: D-stream parity error (parity mode only)

- data put into Dcache unchanged, block gets validated
- machine check
- BIU_STAT: FILL_DPERR set, FILL_IRD clear
- FILL_ADDR[33..5] & BIU_STAT[FILL_QW] give bad QW's address
- FILL_ADDR[4..2] contain PA bits [4..2] of location which the failing load instruction attempted to read
- FILL_SYNDROME identifies failing longword(s)
- BIU_ADDR, BIU_STAT[6..0] locked - contents are UNPREDICTABLE
- DC_STAT: RA identifies register which holds the bad data. LW,LOCK,INT,VAX_FP identify type of load instruction
- BC_TAG holds results of external cache tag probe if external cache was enabled for this transaction

Chapter 4

External Interface

4.1 Overview

The 21064 chip connects directly to an external cache built from off-the-shelf static RAMs. Because building high-speed logic is very difficult in low-end systems, the chip controls the RAMs directly. The chip contains a programmable external cache interface, so that each system can make its own external cache speed and configuration tradeoffs.

The clocks used by the external interface are generated by the chip, but the speed of the clocks is programmable, and is determined during chip reset. This allows each system to make its own external interface speed tradeoffs. The 21064 is configured during reset to use either a 64-bit or 128-bit wide external data bus. The bulk of this chapter describes the chip's operation in 128-bit mode, and Section 4.3 of this chapter describes details specific to 64-bit mode operation.

4.2 Signals

The following table lists all of the signals on the chip. In the "type" column, an "I" means a pin is an input, an "O" means the pin is an output, and a "B" means the pin is bidirectional.

Table 4–1: 21064 Signal Pins

Signal Name	Count	Type	Function
clkIn_h, _l	2	I	Clock input
testClkIn_h, _l	2	I	Clock input for testing
cpuClkOut_h	1	O	CPU clock output
sysClkOut1_h, _l	2	O	System clock output, normal
sysClkOut2_h, _l	2	O	System clock output, delayed
dcOk_h	1	I	Power and clocks ok
reset_l	1	I	Reset
icMode_h 1..0	2	I	Icache Test Mode Selection
sRomOE_l	1	O	Serial ROM output enable
sRomD_h	1	I	Serial ROM data/Rx data
sRomClk_h	1	O	Serial ROM clock/Tx data
adr_h 33..5	29	B	Address bus
data_h 127..0	128	B	Data bus
check_h 27..0	28	B	Check bit bus
dOE_l	1	I	Data bus output enable
dWSel_h 1..0	2	I	Data bus write data select
dRAck_h 2..0	3	I	Data bus read data acknowledge
tagCEOE_h	1	O	tagCtl and tagAdr CE/OE
tagCtlWE_h	1	O	tagCtl WE
tagCtlV_h	1	B	Tag valid
tagCtlS_h	1	B	Tag shared
tagCtlD_h	1	B	Tag dirty
tagCtlP_h	1	B	Tag V/S/D parity
tagAdr_h 33..17	17	I	Tag address
tagAdrP_h	1	I	Tag address parity (Cont.)
tagOk_h, _l	2	I	Tag access from CPU is ok

Table 4–1 (Cont.): 21064 Signal Pins

Signal Name	Count	Type	Function
tagEq_l	1	O	Tag compare output
dataCEOE_h 3..0	4	O	data CE/OE, longword
dataWE_h 3..0	4	O	data WE, longword
dataA_h 4..3	2	O	data A[4..3]
holdReq_h	1	I	Hold request
holdAck_h	1	O	Hold acknowledge
cReq_h 2..0	3	O	Cycle request
cWMask_h 7..0	8	O	Cycle write mask
cAck_h 2..0	3	I	Cycle acknowledge
iAdr_h 12..5	8	I	Invalidate address
dInvReq_h	1	I	Invalidate request, Dcache
dMapWE_h	1	O	Backmap WE, Dcache
irq_h 5..0	6	I	Interrupt requests
vRef	1	I	Input reference
eclOut_h	1	I	Output mode selection
perf_cnt_h 1..0	2	I	Performance counter inputs
tristate_l	1	I	Set pins to high impedance state for testing
cont_l	1	I	Continuity for testing

Systems using 21064 in 128-bit mode should ignore dataA_h 3 and tie dWSel_h 0 false. See Section 4.3 for 64-bit mode details.

4.2.1 Clocks

External logic supplies the 21064 with a differential clock at twice the desired internal clock frequency via the clkIn_h and clkIn_l pins. This clock is divided by 2 to generate the internal chip clock.

The internal chip clock is supplied to the external interface via the cpuClkOut_h pin. The false-to-true transition of cpuClkOut_h is the "CPU clock" used in the timing specification for the tagOk_h,_l signals.

The CPU clock is divided by a programmable value between 2 and 8 to generate a system clock, which is supplied to the external interface via the sysClkOut1_h and sysClkOut1_l pins. The system clock is delayed by a programmable number of CPU clock cycles between 0 and 3 to generate a delayed system clock, which is supplied to the external interface via the sysClkOut2_h and sysClkOut2_l pins.

The clock generator runs generating cpuClkOut_h and correctly timed and positioned sysClk-Out1 and sysClkOut2, while the chip is held in reset.

The output of the programmable divider is symmetric if the divisor is even and asymmetric with sysClkOut1_h TRUE for one extra CPU cycle if the divisor is odd.

The false-to-true transition of sysClkOut1_h is the "system clock" used as a timing reference throughout this specification.

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

4.2.2 DC_OK and Reset

The 21064 contains a ring oscillator which is switched into service during power up to provide an internal chip clock. The dcOk_h signal switches clock sources between an on-chip ring oscillator and the external clock oscillator. If dcOk_h is false then the on-chip ring oscillator feeds the clock generator and the chip is held in reset independent of the state of the reset_l signal. If dcOk_h is true, then the external clock oscillator feeds the clock generator. When dcOk_h is true the vRef input must be valid so that inputs can be sensed. The dcOk_h signal is special in that it does not require that vRef be stable to be sensed. It is important to drive dcOk_h false until the voltage on vRef has stabilized. Because chip testers can apply clocks and power to the chip at the same time, the chip tester can always drive dcOk_h true, but the tester must drive reset_l true for a period longer than the minimum hold time of vRef.

When the 21064 is running off the internal ring oscillator the clock outputs follow it, just like they would when real clocks are applied. The frequency of the ring oscillator varies from chip to chip within a range of 10MHZ to 100MHz, which corresponds to an internal CPU clock frequency of between 5 MHz and 50 MHz. Also, when the dcOk_h signal is false, the system clock divisor is forced to eight, and the sysClkOut2_h, _l delay is forced to three.

If the dcOk_h signal is generated by an RC delay, there is no check to verify that the input clocks are really running. If a board is powered up in manufacturing with a missing, defective, or mis-soldered clock oscillator, then the 21064 will enter a possibly destructive high-current state. Furthermore, if a clock oscillator fails in stage 1 burn-in then the 21064 can also enter this state. The frequency and duration of such events need to be understood by the module designer to decide if this is really a problem.

The reset_l signal forces the CPU into a known state - see Table 3–8. The reset_l signal can be asynchronous, and need not be asserted beyond the assertion of dcOk_h to guarantee that the chip is properly reset.

In order to bring the chip out of internal reset at a deterministic time, the reset_l pin can be deasserted synchronously with respect to the system clock. See chapter Chapter 6 for the setup and hold requirements of the reset_l pin when used in this way.

The 21064 CPU chip uses a 3.3V power supply. This 3.3V supply must be stable before any input goes above 4V.

While in reset, sysClkOut and external bus configuration information is read from the irq_h pins. External logic should drive the configuration information onto the irq_h pins any time reset_l is true.

The irq_h 5 bit is used to select 128-bit or 64-bit mode. If irq_h 5 is true then 128-bit mode is selected.

The irq_h 2..0 bits encode the value of the divisor used to generate the system clock from the CPU clock.

Table 4–2: System Clock Divisor

irq_h 2	irq_h 1	irq_h 0	Ratio
F	F	F	2
F	F	T	3
F	T	F	4
F	T	T	5
T	F	F	6
T	F	T	7
T	T	F	8
T	T	T	8

The irq_h 4..3 bits encode the delay, in CPU clock cycles, from sysClkOut1 to sysClkOut2.

Table 4–3: System Clock Delay

irq_h 4	irq_h 3	Delay
irq_h 4	irq_h 3	Delay
F	F	0
F	T	1
T	F	2
T	T	3

When the tristate_l pin is asserted the chip is internally forced into the reset state, without resampling the interrupt pins.

4.2.3 Initialization and Diagnostic Interface

The 21064 implements three Icache initialization modes to support the chip and module level testing. The value placed on icMode_h 1..0 determines which of these modes is used after the 21064 is reset. Unlike the value placed on irq_h 5..0 during reset, the value placed on icMode_h 1..0 must be retained after reset_l is deasserted.

Table 4–4: Icache Test Modes

icMode_h 1	icMode_h 0	Mode
F	F	Serial Rom
F	T	Disabled
T	F	Icache Test - Write
T	T	Icache Test - Read

If the value on icMode_h 1..0 selects Serial ROM Mode, the 21064 will load the contents of its internal Icache from an external serial ROM (such as an AMD Am1736) before executing its first instruction. The serial ROM could contain as many instructions as needed 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. The 21064 is in PALmode following the deassertion of reset_l. This gives the code loaded into the Icache access to all of the visible state within the chip.

Three signals are used to interface to the serial ROM. The sRomOE_l output signal supplies the output enable to the ROM, serving both as an output enable and as a reset (refer to the serial ROM specifications for details). The sRomClk_h output signal supplies the clock to the ROM that causes it to advance to the next bit. The ROM data is read via the sRomD_h input signal.

Once the data in the serial ROM has been loaded into the Icache, the three special signals become simple parallel I/O pins that can be used to drive a diagnostic terminal. When the serial ROM is not being read, the sRomOE_l output signal is false. If this pin is wired to the active high enable of an RS422 receiver driving onto sRomD_h (the 26LS32 will work) and to the active high enable of an RS422 driver driving from sRomClk_h (the 26LS31 will work). The CPU allows sRomD_h to be read and sRomClk_h to be written by PALcode; this is sufficient hardware support to implement a bit-banged serial interface.

Using the icMode_h 1..0 pins, the Icache diagnostic interface may be disabled altogether. In this case, since the Icache valid bits are cleared by reset, the first instruction fetch will miss the Icache.

In addition to Serial ROM mode, the 21064 includes two test modes which together allow chip tester hardware full read and write access to the Icache. Icache Test/Write Mode works exactly like Serial ROM mode except that bits are loaded into the Icache at a higher rate. Icache Test/Read Mode allows the contents of the Icache to be read in a bit-serial manner from the sRomOE_l pin. These two modes are available only to chip test hardware. Systems using the 21064 must tie icMode_h 1 to FALSE.

In the 21064, all Icache bits are loaded from the diagnostic interface, including each blocks' tag, ASN, ASM, valid and branch history bits. The Icache blocks are loaded in sequential order starting with block zero and ending with block 255. The order in which bits within each block are serially loaded is shown below:

Figure 4–1: Serial RAM Load - Block Ordering

```
                                   a
                                   s
 bht   lw7   lw5   lw3   lw1   v   m   asn   tag   lw6   lw4   lw2   lw0
.___. .___. .___. .___. .___. ._. ._. .___. .___. .___. .___. .___. .___.
|   | |   | |   | |   | |   | | | | | |   | |   | |   | |   | |   | |   |
|   | |   | |   | |   | |   | | | | | |   | |   | |   | |   | |   | |   |
|   | |   | |   | |   | |   | | | | | |   | |   | |   | |   | |   | |   |
\___/ \___/ \___/ \___/ \___/ \_/ \_/ \___/ \___/ \___/ \___/ \___/ \___/
```

Bits within each field are arranged such that high-order bits are on the left. The serial chain shifts to the right.

When the Icache is loaded the valid bit in each cache block is set, and the tag is cleared. Conceptually, the data bits from the serial ROM are shifted into a 64-bit wide holding register and then written into the Icache 64 bits at a time. The bits from the serial ROM are shifted into this holding register from the least significant bit to the most significant bit. Quadwords are written into the Icache in increasing order starting with the quadword at byte address zero.

4.2.4 Address Bus

The three state, bidirectional adr_h pins provide a path for addresses to flow between the 21064 and the rest of the system. The adr_h pins are connected to the buffers that drive the address pins of the external cache RAMs and to the transceivers that are located between the 21064 local address bus and the CPU module address bus.

The address bus is normally driven by the processor. The 21064 stops driving the address bus during reset and during external cache hold. In the external cache hold state the address bus acts like an input, and the tagEq_l output is the result of an equality compare between adr_h and tagAdr_h. Only bits that are part of the cache tag, as specified by the BC_SIZE field of the BIU_CTL IPR, participate in the compare. The tagEq_l pin is asserted during external cache hold only if the result of the tag comparison is true, and the parity calculated across the appropriate bits of tagAdr_h matches the value on tagAdrP_h. Even parity is used. tagEq_l is deasserted when the address bus is not in the external cache hold state.

4.2.5 Data Bus

The three state, bidirectional data_h pins provide a path for data to flow between the 21064 and the rest of the system. The data_h pins connect directly to the I/O pins of the external cache data RAMs and to the transceivers that are located between the 21064 local data bus and the CPU module data bus.

The three state, bidirectional check_h pins provide a path for check bits to flow between the CPU and the rest of the system. The check_h pins connect directly to the I/O pins of the external cache data RAMs, to the transceivers that are located between the 21064 local check bus, and the CPU module check bus.

The data bus is driven by the 21064 when it is running a fast write cycle on the external caches or when some type of write cycle has been presented to the external interface and external logic has enabled the data bus drivers (via dOE_l).

If the 21064 is in ECC mode then the check_h pins carry 7 check bits for each longword on the data bus. Bits check_h 6..0 are the check bits for data_h 31..0. Bits check_h 13..7 are the check bits for data_h 63..32. Bits check_h 20..14 are the check bits for data_h 95..64. Bits check_h 17..21 are the check bits for data_h 127..96.

The following ECC code is used. This code is the same one used by the IDT49C460 and AMD29C660 32-bit ECC generator/checker chips.

Figure 4–2: ECC Code

```
        dddddddddddddddddddddddddddddddd
        33222222222211111111110000000000
        10987654321098765432109876543210
c6 XOR  xxxxxxxx                xxxxxxxx
c5 XOR  xxxxxxxx        xxxxxxxx
c4 XOR  xx      xxxxxx  xx      xxxxxx
c3 XNOR   xxx   xxx   xx  xxx   xxx   xx
c2 XNOR x x  xx x  xx  xx x  xx x  xx  x
c1 XOR     x x x x x xxx   x x x x x xxx
c0 XOR  x xx x    x xxx  x  x xxxx x   x
```

By arranging the data and check bits correctly, it is possible to arrange that any number of errors restricted to a 4-bit group can be detected. One such arrangement is as follows:

Figure 4–3: Example of Errors Detected

```
d 00,  d 01,  d 03,  d 25
d 02,  d 04,  d 06,  c 06
d 05,  d 07,  d 12,  c 03
d 08,  d 09,  d 11,  d 14
d 10,  d 13,  d 15,  d 19
d 16,  d 17,  d 22,  d 28
d 18,  d 23,  d 30,  c 05
d 20,  d 27,  c 04,  c 00
d 21,  d 26,  c 02,  c 01
d 24,  d 29,  d 31
```

If the 21064 is in PARITY mode, then 4 of the check_h pins carry EVEN parity for each longword on the data bus and the rest of the bits are unused. Bit check_h 0 is the parity bit for data_h 31..0. Bit check_h 7 is the parity bit for data_h 63..32. Bit check_h 14 is the parity bit for data_h 95..64. Bit check_h 21 is the parity bit for data_h 127..96.

The ECC bit in the BIU_CTL IPR determines if the 21064 is in ECC mode or in PARITY mode.

4.2.6 External Cache Control

The external cache is a direct-mapped, write-back cache. The 21064 always views the external cache as having a tag for each 32-byte block (the same as the on-chip Icache and Dcache).

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

A cycle requires no module level action if it is a non-LDxL read hit to a valid block, or a non-STxC write hit to a valid but not shared block. All other cycles require module level action. All cycles require module level action if the external cache is disabled (the BC_EN bit in the BIU_CTL IPR is cleared) or the physical address of the reference is in a quadrant in memory that is not cached, i.e. the appropriate bit in the BC_PA_DIS field in the BIU_CTL IPR is set for the quadrant of the reference.

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

The external cache RAMs can be partitioned into three sections; the tagAdr RAM, the tagCtl RAM, and the data RAM. Sections do not straddle physical RAM chips.

4.2.6.1 The TagAdr RAM

The tagAdr RAM contains the high order address bits associated with the external cache block, along with a parity bit. The contents of the tagAdr RAM is fed to the on-chip address comparator and parity checker via the tagAdr_h and tagAdrP_h inputs.

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

The number of bits of tagAdr_h that participate in the address compare and the parity check is controlled by the BC_SIZE field in the BIU_CTL IPR. The tagAdr_h signals go all the way down to address bit 17, allowing for a 128 Kbyte cache built out of RAMs that are 8K deep.

The chip enable or output enable for the tagAdr RAM is normally driven by a two input NOR gate (such as the 74AS805B). One input of the NOR gate is driven by tagCEOE_h and the other input is driven by external logic. The 21064 drives tagCEOE_h false during reset, during external cache hold, and during any external cycle. The OE bit in the BIU_CTL IPR determines if tagCEOE_h has chip enable timing or output enable timing.

4.2.6.2 The TagCtl RAM

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

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

The following combinations of the tagCtl RAM bits are allowed. Note that the bias toward conditional write-through coherence is really only in name. The tagCtlS_h bit can be viewed simply as a write protect bit.

Table 4–5: Tag Control Encodings

tagCtlV_h	tagCtlS_h	tagCtlD_h	Meaning
F	X	X	Invalid
T	F	F	Valid, private
T	F	T	Valid, private, dirty
T	T	F	Valid, shared
T	T	T	Valid, shared, dirty

The 21064 can satisfy a read probe if the tagCtl bits indicate the entry is valid (tagCtlV_h = T). The 21064 can satisfy a write probe if the tagCtl bits indicate the entry is valid and not shared (tagCtlV_h = T, tagCtlS_h = F).

The chip enable or output enable for the tagCtl RAM is normally driven by a two input NOR gate (such as the 74AS805B). One input of the NOR gate is driven by tagCEOE_h and the other input is driven by external logic. The 21064 drives tagCEOE_h false during reset, external cache hold, and any external cycle. The OE bit in the BIU_CTL IPR determines if tagCEOE_h has chip enable timing or output enable timing.

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

4.2.6.3 The Data RAM

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

The most significant bits of the data RAM address are driven, via buffers, from the address bus. The least significant bit of the data RAM address is driven by a two input NOR gate (such as the 74AS805B). One of the inputs of the NOR gate is driven by dataA_ h 4 and the other input is driven by external logic. The 21064 drives dataA_h 4 false during reset, external cache hold, and any external cycle.

The chip enables or output enables for the data RAM are driven by a two input NOR gate. One input of the NOR gate is driven by dataCEOE_h 3..0 and the other input is driven by external logic. The 21064 drives dataCEOE_h 3..0 false during reset, external cache hold, and external cycles. The OE bit in the BIU_CTL IPR determines if dataCEOE_h 3..0 has chip enable timing or output enable timing.

The write enables for the data RAM are normally driven by a two input NOR gate (such as the 74AS805B). One input of the NOR gate is driven by dataWE_h 3..0 and the other input is driven by external logic. The 21064 drives dataWE_h 3..0 false during reset, external cache hold, and any external cycle. The BC_WE_CTL field in the BIU_CTL IPR determines the width of the write enable and its position within the write cycle.

4.2.6.4 Backmap

Some systems may wish to maintain a backmap of the contents of the primary data cache to improve the quality of their invalidate filtering. The 21064 must maintain the backmap for external cache read hits, since external cache read hits are controlled totally by the 21064. External logic maintains the backmaps for external cycles (read misses, invalidates, and so on).

The backmap is only consulted by external logic. Its format and existence is of no concern to the 21064. All the 21064 does is generate a backmap write pulse at the right time. Simple systems will not bother to maintain a backmap, will not connect the backmap write pulse to anything, and will generate extra invalidates.

The write enable input of the data cache backmap RAM is driven by a two input NOR gate (such as the 74AS805B). One side of the NOR gate is driven by dMapWE_h and the other input is driven by external logic. The CPU drives a write pulse onto dMapWE_h whenever is fills the on-chip data cache from the external cache.

In 128-bit mode the dMapWE_h 1..0 signals assert one CPU cycle into the second (last) data read cycle and negate one CPU cycle from the end of that cycle. If read cycles are 3 CPU cycles long, then dMapWE_h is one CPU cycle long. See Section 4.3 for 64-bit mode operations.

Note

This is normally caused by the fact that the backmap write overlaps a cycle whose length is specified by BC_RD_SPD. If we used the standard write pulse timing mechanism and BC_WR_SPD were longer than BC_RD_SPD, the address would go away in the middle of the write cycle.

4.2.6.5 External Cache Access

The external caches are normally controlled by the 21064. Two methods exist for gaining access to the external cache RAMs.

4.2.6.5.1 HoldReq and HoldAck

The simple method for external logic to access the external caches is to assert the holdReq_h signal. When holdReq_h is asserted, the 21064 finishes any external cache cycle which may be in progress, three states adr_h, data_h, check_h, tagCtlV_h, tagCtlD_h, tagCtlS_h, and tagCtlP_h drives tagCEOE_h, tagCtlWE_h, dataCEOE_h, data_WE_h, and dataA_h false and asserts holdAck_h. The cReq_h and cWMask_h signals are not modified in any way. When external logic is finished with the external caches it deasserts holdReq_h. When the 21064 detects the deassertion of holdReq_h it deasserts holdAck_h and re-enables its outputs.

The holdReq_h signal is synchronous. External logic must guarantee setup and hold requirements with respect to the system clock. The holdAck_h signal is synchronous to the CPU clock but phase aligned to the system clock, so it can be used as an input to state machines running off the system clock.

The 21064 generates the holdAck_h signal such that it can be tied directly to the enable-inputs of external three state drivers which connect to the bidirectional pin bus signals. The 21064 will turn off its three state drivers on or before the system clock edge, at which time it asserts holdAck_h and will turn on its three state drivers two CPU cycles after the system clock edge at which it deasserts holdAck_h.

The delay from holdReq_h assertion to holdAck_h assertion depends on the programming of the external interface and on exactly how the system clock is aligned with a pending external cache cycle. In the best case the external cache is idle or is just about to start a cycle. In which case, holdAck_h asserts one system clock cycle after the system clock edge, at which time 21064 samples the holdReq_h assertion. In the worst case at the system clock edge at which 21064 samples the holdReq_h assertion happens one CPU clock cycle into an external cache write probe that hits on a non shared line and requires two RAM data cycles to complete. In this case, holdAck_h asserts at the first system clock edge that is at least ((BC_RD_SPD + 1) - 1) + 2*(BC_WR_SPD + 1) + 1 CPU cycles after the system clock edge at which time the 21064 sampled the holdReq_h assertion.

HoldAck_h deasserts in the system clock cycle immediately following the system clock edge at which time the deassertion of holdReq_h is sampled.

A holdReq_h/holdAck_h sequence can happen at any time, even in the middle of an external transaction. In this case all of the acknowledge-like signals (dOE_l dWSel_h, dRAck_h, cAck_h) work normally. Although the 21064 has forced most of its outputs to either three state or false. Doing anything useful with them is difficult.

The assertion of holdReq_h prevents the BIU sequencer from starting new CPU requests. However, if the BIU sequencer has already started an external cache tag probe when holdReq_h is asserted and the result of the tag probe is such that an external transaction is required to complete the CPU's request. The BIU sequencer will initiate the external transaction by driving the cReq_h signals to the appropriate value despite holdReq_h's assertion. The holdAck_h signal will assert at the next system clock edge after the tag probe completes.

The 21064 does not turn on its three state drivers until two CPU cycles after it deasserts holdAck_h. Care must be taken as to when external logic begins processing new external transactions at the tail end of a holdReq_h/holdAck_h sequence.

4.2.6.5.2 TagOk

The fastest way for external logic to gain access to the external caches is to use the tagOk_ h, _l signals. The TagOk_h, _l signals are 21064 bus interface control signals which allow external logic to stall a CPU cycle on the external cache RAMs at the last possible instant. All tradeoffs surrounding these signals have been made in favor of high-performance systems, making them next to impossible to use in low-end systems.

The tagOk_h and tagOk_l signals are synchronous. External logic must guarantee setup and hold requirements with respect to the CPU clock. This implies very fast logic, since the CPU clock may run at 200 MHz for the binned parts.

The only thing that tagOk does is stall a sequencer in the 21064 bus interface unit. The 21064 does not tri-state the busses that run between the CPU and the external cache RAMs. External logic must supply the necessary multiplexing functions in the address and data path.

If the tagOk is true at a CPU clock edge, the external logic is guaranteeing that the tagCtl and tagAdr RAMs were owned by the 21064 in the previous BC_RD_SPD+1 CPU cycles, that the tagCtl RAMs will be owned by the 21064 in the next BC_WR_SPD+1 cycles, that the data RAMs were owned by the 21064 in the previous BC_RD_SPD+1 cycles, and that the data RAMs will be owned by the 21064 in the next BC_RD_SPD+1 CPU cycles or in the next 2*(BC_WR_SPD+1) CPU cycles, whichever is longer.

The bus interface unit samples tagOk in the last two cycles of each tag probe and only proceeds if tagOk was asserted in both of these cycles. Two cycles of tagOk assertion rather then one was chosen to alleviate a tight circuit path inside the chip. This choice in no way impacts the above stated use of tagOk by external logic. If the 21064 samples tagOk as false in either of the last two CPU cycles of a tag probe, then it stalls until it samples tagOk true in consecutive cycles (at which time all of the above assertions are true, which means that any address the 21064 has been holding on the address bus has made it through the external cache RAMs) and then it proceeds normally.

4.2.7 External Cycle Control

On READ_BLOCK and LDxL cycles, the cWMask_h pins have additional information about the cache miss overloaded onto them. The cWMask_h 1:0 and cWMask_h 3 pins contain miss address bits 4:3 and 2 respectively. These additional address bits, which specify the longword that missed are needed to implement longword granularity to I/O devices.

An external cycle begins when the 21064 puts a cycle type onto the cReq_h outputs. Some cycles put an address on the adr_h outputs and additional information (low-order address bits, I/D stream indication, write masks) on the cWMask_h outputs. All of these outputs are synchronous and the 21064 meets setup and hold requirements with respect to the system clock.

The cycle types are as follows.

Table 4–6: Cycle Types

cReq_h 2	cReq_h 1	cReq_h 0	Type
F	F	F	IDLE
F	F	T	BARRIER
F	T	F	FETCH
F	T	T	FETCHM
T	F	F	READ_BLOCK
T	F	T	WRITE_BLOCK
T	T	F	LDxL
T	T	T	STxC

A BARRIER cycle is generated by the MB instruction. Normally all the module does with this cycle is acknowledge it. Modules which have write buffers between the 21064 and the memory system must drain these buffers before the cycle is acknowledged. This guarantees that machine checks caused by transport and/or memory system errors get posted on the correct side of the MB instruction.

The FETCH and FETCHM cycles are generated by the FETCH and FETCHM instructions, respectively. The address bus contains the effective address of the FETCH or FETCHM instruction. These addresses can be used by module level prefetching logic. Simple systems simply acknowledge the cycles.

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

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

The LDxL cycle is generated by the interlocked load instructions. The cycle works just like a READ_BLOCK, although the external cache has not been probed (so the external logic needs to check for hits) and the address has to be latched into a locked address register.

The STxC cycle is generated by the conditional store instructions. The cycle works just like a WRITE_BLOCK, although the external cache has not been probed (so that external logic needs to check for hits) and the cycle can be acknowledged with a failure status.

On WRITE_BLOCK and STxC cycles the cWMask_h pins supply longword write masks to the external logic, indicating which longwords in the 32-byte block are valid. A cWMask_ h bit is true if the longword is valid. WRITE_BLOCK commands can have any combination of mask bits set. STxC cycles can only have combinations that correspond to a single longword or quadword.

On READ_BLOCK and LDxL cycles the cWMask_h pins have additional information about the miss overloaded onto them. The cWMask_h 1..0 pins contain miss address bits 4..3 (indicating the address of the quadword that actually missed), which is needed to implement quadword read granularity to I/O devices. The cWMask_h 2 pin is true if the miss is a D-stream reference and false if the miss is an I-stream reference.

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

The acknowledgment types are as follows.

Table 4–7: Acknowledgment Types

cAck_h 2	cAck_h 1	cAck_h 0	Type
F	F	F	IDLE
F	F	T	HARD_ERROR
F	T	F	SOFT_ERROR
F	T	T	STxC_FAIL
T	F	F	OK

The 21064 behavior in response to cAck_h encodings others than those listed above is UNDEFINED.

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

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

The STxC_FAIL type indicates that a STxC cycle has failed. It is UNDEFINED what happens if this type is used on anything but an STxC cycle.

The OK type indicates success.

The dRAck_h pins inform the 21064 that read data is valid on the data bus, if the data should be cached, and if ECC checking and correction or parity checking should be attempted. The dRAck_h inputs are synchronous. External logic must guarantee setup and hold requirements with respect to the system clock. If dRAck_h is sampled IDLE at a system clock, then the data bus is ignored. If dRAck_h is sampled non IDLE at a system clock, then the data bus is latched at that system clock and external logic must guarantee that the data meets setup and hold with respect to the system clock.

The acknowledgment types are as follows.

Table 4–8: Read Data Acknowledgment Types

dRAck_h 2	dRAck_h1	dRAck_h 0	Type
F	F	F	IDLE
T	F	F	OK_NCACHE_NCHK
T	F	T	OK_NCACHE
T	T	F	OK_NCHK
T	T	T	OK

The 21064 behavior in response to dRAck_h encoding others than those listed above is UNDEFINED.

The first non IDLE sample of dRAck_h tells the 21064 to sample data bytes 15..0 and the second non IDLE sample of dRAck_h tells the 21064 to sample data bytes 31..16. External logic may drive the second dRAck_h and the cAck_h during the same system clock.

READ_BLOCK and LDxL transactions may be terminated with HARD_ERROR status before all expected dRAck_h cycles are received. The contents of the entire cache block (including its tag and valid bit) are UNPREDICTABLE.

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

The 21064 does not change its interpretation of dRAck_h 1..0 based on cAck_h if all expected dRAck's are received. External logic must avoid caching and/or ECC/parity checking data which is known to be garbage.

The 21064 behavior is UNDEFINED if dRAck_h is asserted in a non-read cycle.

The 21064 checks dRAck_h 0 (the bit that determines if the block is ECC/parity checked) during both halves of the 32-byte block. It is legal, but probably not useful, to check only one half of the block. The 21064 checks dRAck_h 1 during the first half of the 32-byte block.

The dOE_l inputs tells the 21064 if it should drive the data bus. It is a synchronous input, so external logic must guarantee setup and hold with respect to the system clock. If dOE_l is sampled true at a system clock, then the 21064 drives the data bus at the system clock if it has a WRITE_BLOCK or STxC request pending (the request may already be on the cReq pins, or it may appear on the cReq pins at the same system clock edge as the data appears). If dOE_l is sampled false at the system clock, then the 21064 tri-states the data bus on the next system clock cycle. The cycle type is factored into the enable so that systems can leave dOE_l asserted unless it is necessary to write a victim.

The dWSel_h inputs tells the 21064 which half of the 32-byte block of write data should be driven onto the data bus (dOE_l permitting). They are synchronous inputs. External logic must guarantee setup and hold with respect to the system clock. If dWSel_h 1 is sampled false at the end of a system clock cycle, then bytes 15..0 are driven onto the data bus in the next system clock cycle. If dWSel_h 1 is sampled true at the end of a system clock cycle, then bytes 31..16 are driven onto the data bus in the next system clock cycle. Once dWSel_h 1 has been sampled true bytes 15..0 are lost. There is no backing up.

4.2.8 Primary Cache Invalidate

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

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

The first situation is the obvious one. Any time an external agent updates a block in memory (for example, an I/O device does a DMA transfer into memory) and the block has been loaded into the external cache, then the external cache block must be either invalidated or updated. If the external cache block has been loaded into the Dcache, then the Dcache block must be invalidated.

The second situation is more subtle. If a system is maintaining the Dcache as a subset of the external cache and an Icache miss results in an external cache block being replaced and that external cache block has been loaded into Dcache, then an invalidate is needed.

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

The 21064 can accept an invalidate at every system clock.

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

4.2.9 Interrupts

External interrupts are fed to the 21064 via the irq_h bus. The 6 interrupts are identical. They can be asynchronous, level sensitive, and individually masked by PALcode.

It is expected that on most systems, a combination of hardware and PALcode will use these 6 inputs as a power fail interrupt, a halt interrupt, and as 4 external interrupts (with the timer interrupt, the interprocessor interrupt, and the corrected read data interrupt wired to their normal IPL). This is not enforced by the 21064. Low-end systems could, for example, use all of them as device interrupts and arrange that its PALcode treated them all as IPL20 interrupts, using fixed vectors. See Section 2.3.3 for more details on interrupts.

To aid pattern-driven chip testers, the irq_h pins may be driven synchronously with respect to the system clock. See chapter Chapter 6 for the setup and hold requirements of the irq_h pins with respect to the system clock for this case.

4.2.10 Electrical Level Configuration

The 21064 can drive and receive either CMOS levels or 100K ECL levels (with assistance from resistors on the module).

The vRef input supplies a reference voltage to the input sense circuits. If external logic ties this to VSS + 1.4V, then all inputs sense TTL levels. If external logic ties this to VDD -1.3V (which can be obtained, for example, from the VBB output of an MC100E111) then all inputs sense ECL 100K levels.

The eclOut_h input selects the output levels. If external logic ties this false then all outputs generate CMOS levels. If external logic ties this true then all outputs are switched into a mode in which external resistors can be used to generate ECL 100K compatible levels.

4.2.11 Performance Monitoring

The perf_cnt_h 1..0 pins provide a means of giving the 21064's internal performance monitoring hardware access to off-chip events. These pins are system clock synchronous inputs which may be selected via the ICCSR IPR to be inputs to the performance counters inside the 21064 chip. If in a given system clock cycle a perf_cnt_h pin is sampled TRUE and the pin is selected as the source of its respective performance counter, then the counter will increment.

4.2.12 tristate

The tristate_l signal, if asserted, causes the 21064 to float all of its output and bidirectional pins with the exception of cpuClkOut_h. When tristate_l is asserted, The 21064 is forced into the reset state, but the irq_h pins are not resampled.

4.2.13 Continuity

The cont_l signal, if asserted, causes the 21064 to connect all of its pins to VSS, with the exception of clkIn_h, clkIn_l, testClkIn_h, testClkIn_l, cpuClkOut_h, sysClkOut1_h, sysClkOut1_l, sysClkOut2_h, sysClkOut2_l, VREF and cont_l.

4.3 64-Bit Mode

The 21064 may be configured at reset to use a 64-bit wide external data bus. In which case, data_ h 127..64 and check_h 27..14 are not used. These pins are internally pulled to VSS.

The dataA_h 3 pin is used as an additional address line for the external cache data RAMs. Like the dataA_h 4 pin, it should drive one input of a two input NOR gate, with the other input being driven by external logic. The 21064 drives dataA_h 3 false during reset, external cache hold, and any external cycle.

The dWSel_h 0 and dWSel_h 1 pins should be used by external logic to select which quadword of a 32-byte block is driven onto data_h 63..0 during each system clock cycle of an external WRITE_BLOCK or STxC transaction. The relationship between dWSel_h 1..0 and the selected bytes of the 32-block block is as follows:

Table 4–9: dWSel_h

dWSel_h 1..0	Selected Bytes
00	07..00
01	15..08
10	23..16
11	31..24

External logic must select quadwords in increasing order within the 32-byte block, but is free to skip over any quadword which does not have corresponding longword mask bits TRUE in cWMask_h 7..0.

Systems should ignore dataCEOE_h 3..2 and dataWE_h 3..2.

External cache read hit transactions are extended to consist of four cache read cycles in 64-bit mode. Each cache read cycle is (BC_RD_SPD + 1) CPU cycles in duration. The first cache read cycle consists of a tag probe and data read, while the subsequent three cache read cycles consist of data reads. The 21064 bus interface optimizes the external cache read hit transaction by wrapping cache read cycles around the quadword, which the 21064 originally requested. The dMapWE_h pin asserts 1 CPU cycle into the second cache read cycle and remains asserted until one CPU cycle before the end of the fourth cache read cycle.

External cache write hit transactions consist of one cache tag probe cycle, which is (BC_RD_SPD + 1) CPU cycles long and followed by one, two, three or four external cache write cycles which are each (BC_WR_SPD + 1) cycles long. The 21064 bus interface uses the minimum number of cache write cycles required to write the necessary longwords within the 32-byte block.

Note that the maximum latency from holdReq_h assertion to holdAck_h assertion in 64-bit mode is longer than in 128-bit mode. Also, the guarantee which external logic must make to the availability of the external cache data RAMs when asserting tagOk is different for 64-bit mode than for 128-bit mode.

For external READ_BLOCK and LDxL transactions the 21064 chip normally expects four distinct dRAck_h acknowledgment cycles. The first non-IDLE dRAck_h sample informs the 21064 to sample data bytes 7..0. The second to sample data bytes 15..8 and so on. Each quadword is parity/ECC checked based on the dRAck_h code supplied with that quadword. The dRAck_h code supplied with the first quadword determines whether the 32-byte block is cached.

4.4 Transactions

4.4.1 Reset

External logic resets the 21064 by asserting reset_l. When the 21064 detects the assertion of reset_l it terminates all external activity and places the output signals on the external interface into the following state. Note that all of the control signals have been placed in the state that allows external access to the external cache.

Table 4–10: Reset State

Pin	State
sRomOE_l	F
sRomClk_h	T
adr_h	Z
data_h	Z
check_h	Z
tagCEOE_h	F
tagCtlWE_h	F
tagCtlV_h	Z
tagCtlS_h	Z
tagCtlD_h	Z
tagCtlP_h	Z
dataCEOE_h	F
dataWE_h	F
dataA_h	F
holdAck_h	F
cReq_h 2:0	FFF

After asserting reset_l for long enough to reset the serial ROM (100 ns), external logic negates reset_l.

When the 21064 detects reset_l negate, it may load bits from an external serial ROM into its internal Icache, based on the value placed on icMode_h 1..0. The timing is shown below (assuming the 21064 only read 3 bits from the serial ROM):

Figure 4–4: 21064 RESET Sequence Timing

```
reset_l         --------|  ...
sRomOE_l                          |-------------------|
sRomClk_h                       ----| |-----| |-----| |--
Sample sRomD_h                        ^       ^       ^
```

Each half-tick of the sRomClk_h signal is 63 CPU cycles long, which guarantees the 200ns clock high and clock low specifications and the 400ns clock to data specification of the serial ROM with 5ns CPU cycles.

It is possible to disable the serial ROM mechanism altogether. See Section 4.2.3.

4.4.2 Fast External Cache Read Hit

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

The following diagram assumes that the external cache is using 4 cycle reads (BC_RD_SPD = 3), 4 cycle writes (BC_WR_SPD = 3), and chip enable control (OE = L).

Figure 4–5: Fast External Cache Read Sequence

```
Internal Clock        |0   |1   |2   |3   |4   |5   |6   |7  |
adr_h                 |------------------------------|
tagCEOE_h             |---------------|
tagCtlWE_h
tagAdr_h                        -ram-|
tagCtl_h                        -ram-|
dMapWE_h                                    |-------|
dataCEOE_h            |------------------------------|
dataWE_h
dataA_h 4                             |---------------|
data_h                         -ram-0-|        -ram-1-|
check_h                        -ram-0-|        -ram-1-|
```

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

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

4.4.3 Fast External Cache Write Hit

A fast external cache write consists of a probe read, followed by 1 or 2 data writes.

The following diagram assumes that the external cache is using 4 cycle reads (BC_RD_SPD = 3), 4 cycle writes (BC_WR_SPD = 3), chip enable control (OE = L), and a 2 cycle write pulse centered in the 4 cycle write (BC_WE_CTL 15..1 = LLLLLLLLLLLLLHH).

Figure 4–6: Fast External Cache Write Sequence

```
Internal Clock|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10   |11  |
adr_h         |------------------------------------------------|
tagCEOE_h     |---------------|   |-------|
tagCtlWE_h                        |-------|
tagAdr_h                 -ram-|
tagCtl_h                 -ram-|   |-cpu-------|
dataCEOE_h                        |-------|       |---------|
dataWE_h                          |-------|       |---------|
dataA_h 4                                     |---------------|
data_h                        |-cpu-0---------|-cpu-1---------|
check_h                       |-cpu-0---------|-cpu-1---------|
```

The 21064 drives the tagCtl_h pins one CPU cycle later than it drives the data_h and check_h pins, relative to the start of the write cycle. Unlike data_h and check_h, the tagCtl_h field must be read during the tag probe which proceeds the write cycle. Since the 21064 can switch its pins to a low impedance state much more quickly than most RAMs can switch their pins to a high impedance state, the 21064 waits one CPU cycle before driving the tagCtl_h pins in order to minimize three state driver overlap.

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

4.4.4 READ_BLOCK Transaction

A READ_BLOCK transaction appears at the external interface on external cache read misses, either because it really was a miss or because the external cache has not been enabled.

Figure 4–7: READ_BLOCK Sequence

```
sysClkOut Cycle |   0   |   1   |   2   |   3   |   4   |   5   |   6   |
sysClkOut1_h    |---|   |---|   |---|   |---|   |---|   |---|   |---|   |
adr_h               |-------------------------------------------|
RAM Ctl         ------------|
data_h                                  |-0-----|       |-1-----|
check_h                                 |-0-----|       |-1-----|
cReq_h                  |---------------------------------------|
cWMask_h                |-------------------------------|
dRAck_h                                 |-------|       |-------|
cAck_h                                                  |-------|
```

0. The cReq_h pins are always idle in the system clock cycle immediately before the beginning of an external transaction. The adr_h pins always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.
1. The READ_BLOCK transaction begins. The 21064 has already placed the address of the block containing the miss on adr_h. The 21064 places the quadword-within-block and the I/D indication on cWMask_h. The 21064 places a READ_BLOCK command code on cReq_h. The 21064 will clear the RAM control pins (dataA_h 4..3, dataCEOE_h 3..0 and tagCEOE_h) no later than one CPU cycle after the system clock edge at which the transaction begins.
2. The external logic obtains the first 16 bytes of data. Although a single stall cycle has been shown, there could be no stall cycles or many stall cycles.
3. The external logic has the first 16 bytes of data. It places it on the data_h and check_h busses. It asserts dRAck_h to tell the 21064 that the data and check bit busses are valid. The 21064 detects dRAck_h at the end of this cycle and reads in the first 16 bytes of data at the same time.
4. The external logic obtains the second 16 bytes of data. Although a single stall cycle has been shown, there could be no stall cycles or many stall cycles.

5. The external logic has the second 16 bytes of data. It places it on the data_h and check_h busses. It asserts dRAck_h to tell the 21064 that the data and check bit busses are valid. The 21064 detects dRAck_h at the end of this cycle and reads in the second 16 bytes of data at the same time. The external logic places an acknowledge code on cAck_h to tell the 21064 that the READ_BLOCK cycle is completed. The 21064 detects the acknowledge at the end of this cycle and can change the address.

6. Everything is idle. The 21064 could start a new external cache cycle at this time.

External logic owns the RAMs by virtue of 21064 having deasserted its RAM control signals at the beginning of the transaction, external logic may cache the data by asserting its write pulses on the external cache during cycles 3 and 5.

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

4.4.5 Write Block

A WRITE_BLOCK transaction appears at the external interface on external cache write misses (either because it really was a miss or because the external cache has not been enabled) or external cache write hits to shared blocks.

Figure 4–8: WRITE_BLOCK Sequence

```
sysClkOut Cycle |    0    |    1    |    2    |    3    |    4    |    5    |    6    |
sysClkOut1_h    |---|     |---|     |---|     |---|     |---|     |---|     |---|     |-
adr_h               |--------------------------------------------|
RAM ctl         ------------|
data_h                                        |-0-----|           |-1-----|
check_h                                       |-0-----|           |-1-----|
cReq_h                    |----------------------------------------|
cWMask_h                  |----------------------------------------|
dOE_l                               |-------|           |-------|
dWSel_h                                                 |-------|
cAck_h                                                            |-------|
```

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

1. The WRITE_BLOCK cycle begins. The 21064 has already placed the address of the block on adr_h. The 21064 places the longword valid masks on cWMask_h and a WRITE_BLOCK command code on cReq_h. The 21064 will clear dataA_h 4..3 and tagCEOE_h no later than one CPU cycle after the system clock edge, at which time the transaction begins. The 21064 clears dataCEOE_H 3..0 at least one CPU cycle before the system clock edge, at which time the transaction begins.

2. The external logic detects the command and asserts dOE_l to tell the 21064 to drive the first 16 bytes of the block onto the data bus. The timing shown for dOE_l is chosen for discussion purposes. External logic can assert dOE_l by default and only deassert it when it needs to read the data RAMs, such as when writing back a victim block.
3. The 21064 drives the first 16 bytes of write data onto the data_h and check_h busses and the external logic writes it into the destination. Although a single stall cycle has been shown, there could be no stall cycles or many stall cycles.
4. The external logic asserts dOE_l and dWSel_h to tell the 21064 to drive the second 16 bytes of data onto the data bus.
5. The 21064 drives the second 16 bytes of write data onto the data_h and check_h busses and the external logic writes it into the destination. Although a single stall cycle has been shown, there could be no stall cycles or many stall cycles. External logic places an acknowledge code on cAck_h to tell the 21064 that the WRITE_BLOCK cycle is completed. The 21064 detects the acknowledge at the end of this cycle and changes the address and command to their next values.
6. Everything is idle. The 21064 can start a new external cache access at this time.

External logic owns the RAMs by virtue of the 21064 having deasserted its RAM control signals at the beginning of the transaction, external logic may cache the data by asserting its write pulses on the external cache during cycles 3 and 5.

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

Although in the above diagram external logic cycles through both 128-bit chunks of potential write data, this need not always be the case. External logic must pull from the 21064 chip only those 128-bit chunks of data which contain valid longwords as specified by the cWMask_h signals. The only requirement is that if both halves are pulled from the 21064, then the lower half must be pulled before the upper half.

4.4.6 LDxL Transaction

An LDxL transaction appears at the external interface when an interlocked load instruction is executed. The external cache is not probed. With the exception of the command code output on the cReq pins, the LDxL transaction is exactly the same as a READ_BLOCK transaction. See section Section 4.4.4.

4.4.7 STxC Transaction

An STxC transaction appears at the external interface when a conditional store instruction is executed. The external cache is not probed.

The STxC transaction is the same as the WRITE_BLOCK transaction, with the following exceptions:

0. The code placed on the cReq pins is different.
1. The cWMask field will never validate more than a single longword or quadword of data.

2. External logic has the option of making the transaction fail by using the cAck code of STxC_FAIL. It may do so without asserting either dOE_l or dWSel_h.

See section Section 4.4.5.

4.4.8 BARRIER Transaction

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

Figure 4–9: BARRIER Request/Acknowledge Sequence

```
sysClkOut Cycle |   0    |   1    |   2    |
sysClkOut_h     |---|    |---|    |---|    |-
cReq_h          |---------------|
cAck_h                   |-------|
```

0. The BARRIER transaction begins. The 21064 places the command code for BARRIER onto the cReq_h outputs.
1. The external logic notices the BARRIER command and places an acknowledge code on the cAck_h inputs.
2. The 21064 detects the acknowledge on cAck_h and removes the command. The external logic removes the acknowledge code from cAck_h. The cycle is finished.

4.4.9 FETCH Transaction

A FETCH transaction appears on the external interface as a result of a FETCH instruction. The transaction supplies an address to the external logic, which can choose to ignore it or use it as a memory-to-cache prefetching hint.

Figure 4–10: FETCH Sequence

```
sysClkOut Cycle |    0    |    1    |    2    |    3    |
sysClkOut_h     |---|     |---|     |---|     |
adr_h               |--------------------|
RAM ctl         ------------|
cReq_h                    |----------------|
cAck_h                              |-------|
```

0. The cReq_h pins are always idle in the system clock cycle immediately before the beginning of an external transaction. The adr_h pins always change to their final value (with respect to a particular external transaction) at least one CPU cycle before the start of the transaction.
1. The FETCH transaction begins. The 21064 has already placed the effective address of the FETCH on the address outputs. The 21064 places the command code for FETCH on the cReq_ h outputs. The 21064 will clear the RAM control pins (dataA_h 4..3, dataCEOE_h 3..0 and tagCEOE_h) no later than one CPU cycle after the system clock edge, at which time the transaction begins.
2. The external logic notices the FETCH command and places an acknowledge code on the cAck_h inputs.
3. The 21064 detects the acknowledge on cAck_h and removes the address and the command. The external logic removes the acknowledge code from cAck_h. The cycle is finished.

4.4.10 FETCHM Transaction

A FETCHM transaction appears on the external interface as a result of a FETCHM instruction. The transaction supplies an address to the external logic, which can choose to ignore it or use it as a memory-to-cache prefetching hint. With the exception of the command code placed on cReq_h, the FETCHM transaction is the same as the FETCH transaction. See section Section 4.4.9.

Chapter 5
DC Characteristics

5.1 Overview

The 21064 is capable of running in a CMOS/TTL environment or an ECL environment. The chip 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 5.2.

5.1.1 Power Supply

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

To prevent damage to the chip, it is important that the 3.3V power supply be stable before any input or bidirectional pins be allowed to rise above 4.0V.

To help in meeting this requirement, the assertion levels of the 21064's input pins have been arranged so that their default state is the electrical low state. This makes them active high, with the exception of tagOk_l and dOE_l, which are true (low) by default.

5.1.2 Reference Supply

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

5.1.3 Input Clocks

The clkIn_h,_l signals are differential signals generated from an ECL oscillator circuit, although non-ECL circuits can also be used. The signals can be AC coupled, with a nominal DC bias of VDD/2 set by a high-impedence (i.e. >1K) resistive network on the chip. The signals 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.

5.1.4 Signal pins

Input pins are ordinary CMOS inputs with standard TTL levels, see Table 5–1. Once power has been applied and vRef has met its hold time, the majority of input pins can be driven by 5.0V (nominal) signals without harming the 21064. There are some signals that are sampled before vRef is stable, and these signals can not be driven above the power supply. These signals are:

- dcOk_h
- tristate_l
- cont_l
- eclOut_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 5–1.

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 the 21064 (it is not necessary to use static RAMS with 3.3V outputs).

Table 5–1: CMOS DC Characteristics - TTL Inputs/Outputs

Symbol	Description	Min	Max	Units	Test Cond.
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
Icin	Clock input Leakage	4	4	mA	-0.5<Vin<3.6V
Iil	Input leakage current	10	10	uA	0<Vin<Vdd V
Iol	Output leakage current	-10	-10	uA	

5.2 ECL 100K Mode

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

5.2.1 Power Supply

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

5.2.2 Reference Supply

In ECL 100K mode the vRef input is connected to a reference supply at VDD minus 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.

5.2.3 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.

5.2.4 Outputs

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

Figure 5–1: ECL Termination

```
    |                        |
    |      .---.             |
CPU |------|R1 |-------------| ECL 100K
    |      `---'     |       |
    |       50 ohms .-.      |
                    |R|
                    |2| 100 ohms
                    `-'
                     |
                    -+-
                    -2.0V
```

5.2.5 Bidirectionals

In ECL 100K mode the bidirectional pins should be converted into unidirectional input and output busses as close to the 21064 as possible. The 21064 chip bidirectional bus is buffered and driven onto the system output bus. The system input bus is driven onto the 21064'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.

5.3 Power Dissipation

A comprehensive power dissipation analysis was performed using a program that caused maximum dynamic power dissipation. It was run on a logic simulator and using analysis tools, power dissipation was analytically predicted. The results from that analysis are shown in Table 5–2.

Table 5–2: 21064 Power Dissipation @Vdd=3.45V

Speed	Min	Typ	Max	Units
5.0ns	24	29.5	36	Watts
6.6ns	19	23	27.5	Watts

The minimum power occurs during reset, the "Typ" column is the worst case average program and the "Max" column is the worst case pathological program. An important observation is the fact that all normal programs (both stand-alone and under a high level operating system) run in a range between "Min" and "Typ". So while the pathological case is theoretically possible, it is extremely unlikely in practice. The following approach is recommended for system designers:

- Design the heat sink and thermal environment to keep the die temperature to 85C for the "Typ" power case. This is certainly the limiting case for average power dissipation to be used for long term reliability assessment. With "Typ" designed for 85C, then in all cases, "Max" will result in die temperatures under the 100C design, test and process qual limits.
- Design the overall enclosure and the power supply to handle the "Max" power case.

It is possible to account for applications where the maximum supply voltage is other than 3.45V and/or the operating frequency is not 150 or 200MHz. The formulae for calculating Idd under various conditions are as follows:

Idd (Min) = 116mA/V + 9.6mA/V*MHz
Idd (Typ) = 116mA/V + 11.7mA/V*MHz
Idd (Max) = 116mA/V + 14.4mA/V*MHz

Chapter 6

AC Characteristics

This chapter contains the AC specification for the 21064-AA. Timing parameters are given for an internal clock speed of 150 Mhz (6.6ns cycle time).

6.1 vRef

vRef is an analog reference voltage used by the input buffers of all signals except clkIn_h,_l, testclkIn_h,_l, tagOk_h,_l, dcOk_h, eclOut_h, tristate_l, and cont_l. Upon power up, reset_l cannot be sampled until vRef is stable. There is a large internal capacitance on vRef and an RC delay between its pin and the input buffers. Therefore, systems must not assert dcOk_h until a suitable interval following the stability of the vRef source. This interval is specified as the greater of 1us and 10nF * Zout, where Zout is the vRef source impedance.

6.2 Input Clocks

The input clocks clkIn_h,_l and testclkIn_h,_l are received differentially, then XORed to provide the time-base when dcOk_h is asserted. The testclkIn_h, _lsignals should only be used by testers unable to drive clkIn_h, _l at full speed. The terminations on these signals are designed to be compatible with system oscillators of arbitrary DC bias. Schematically, they look as follows:

Figure 6–1: Clock Termination

```
.-----.         .-----.
| PIN |----+----| PAD |-----------+------------> (to diff-amp)
`-----'    |    `-----'           |
           |                      |
           |                      | 50ohms       Hi_Z
         ----- Cpkg               `----RRRR----+----RRRR----.
         -----                                 |            |
           |                                 -----          |
           |                                 ----- 40pF     |
           |       Vbias = (Vdd-Vss)/2         |            |
           `-----------------------------------+------------'
```

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 the clock input. The peak-to-peak amplitude of the clock source must be between 0.6V and 3.0V as seen by the 21064. Either a "square-wave" or a sinusoidal source can be used.

Note that full-rail clocks can be driven by testers.

The following table lists the input clock cycle times. Note that these periods equal one-half the corresponding cpu cycle times.

Table 6–1: Input Clock Timing

Name	21064-AA	Unit
clkIn period min	3.3	ns
clkIn period max	tbd	ns
clkIn symmetry	50%+/-10%	percent

6.3 cpuClkOut_h

The cpuClkOut_h signal is expected to be used only by an ECL synchronizer in systems using the tagOk protocol. In order to accommodate ECL levels, the driver consists of only a PMOS pullup device. ECL 100K levels may be constructed with a 50ohm board resistor in series with the driver and a 100ohm board resistor between the load and Vdd minus 2V. CMOS Vdd must equal ECL Vcc in this scheme. Note that the trace must be short to insure good signal integrity if the board impedance is not in the vicinity of 100ohm.

6.4 Test Configuration

All outputs and bidirectional signals including clocks (but excluding cpuClkOut_h) are specified with respect to a standard 40pF load as shown below. All timing is specified with respect to the crossing of standard TTL input levels at 0.8V and 2.0V.

Figure 6–2: Standard Load

```
       |
 21064 |------------.
   PIN |            |
       |           --- 40pf
                   ---
                    |
                   -+-
                   GND
```

6.5 Fast Cycles on External Cache

From a system standpoint, fast cycles on the external cache are completely unclocked. The two cases of read and write cycles require separate treatment.

6.5.1 Fast Read Cycles

External logic must meet the maximum flow-through delay, as defined with respect to the circuits below.

Figure 6–3: Flow-through Delay

```
      | Address                         |   Address    .----------.
21064 |-------------.             21064 |-------------|          |
  PIN | Control     |               PIN |   Control   |          |
      |            --- 40pf             |             | External |
                   ---                                |          |
                    |                                 |  Logic   |
                   -+-                  |    Data     |          |
                   GND            21064 |-------------|          |
                                    PIN |             `----------'
                                        |
```

"Address" refers to adr_h and dataA_h. "Control" refers to dataCEOE_h and tagCEOE_h. "Data" refers to data_h, check_h, tagAdr_h, and tagCtl_h. Assume that address/control is driven from the same internal clock edge in the two cases above. External flow-through delay is defined as the delay between address/control valid to the 40pF standard load in the case on the left and data valid to the 21064 in the case on the right. It can not exceed the fast read cycle time (i.e. BC_RD_SPD+1 cpu cycles) less 5.0ns. The 21064 guarantees that its address drivers are enabled at least one cpu cycle prior to a fast cache access, such that, adr_h never needs to be pulled down from 5V during the cycle.

6.5.2 Fast Write Cycles

External logic must guarantee that fast write cycles complete. Data, address, and control (including dataWE_h and tagCtlWE_h) are driven by the 21064 with identical timing from its internal clock. Actual pulse widths are at least the nominal width less 1.5ns or 2.9ns on lines precharged to 5V (i.e. data lines following a probe read). The timing of dMapWE_h during dcache read hits is specified in the same way.

6.6 External Cycles

All external cycle timing is referenced to the rising edge of sysClkOut1_h. Input setup, hold times, output delay, and enable times are referenced to the point at which sysClkOut1_h crosses 0.8V. (Output enable time is defined as output delay time from a tri- stated state. It can differ from the nominal delay, because it may entail pulling the signal down from a 5V level.) Output hold times are referenced to the point at which sysClkOut1_h crosses 2.0V. They denote the times beyond sysClkOut1_h for which outputs hold their valid values from the previous cycle. Note that these times are negative, meaning that data may lose validity BEFORE sysClkOut1_h becomes valid high. (This is possible because there is no cause-effect relationship between the system clock outputs and data. The system clock outputs are nothing more than data pins which happen to switch in a fixed pattern.) Address enable timing is relevant only for systems using the holdReq protocol with two cpu cycles per system cycle. All bidirectional lines can be considered enable or disabled simultaneously with the rising edge of sysClkOut1_h.

Table 6–2: External Cycles

Name	Min	Max	Units
Enable, sysClkOut1_h to			
adr_h, data_h, check_h		2.9	ns
Output Delay, sysClkOut1_h to			
adr_h, data_h, check_h, cReq_h,		+1.5	ns
cWMask_h, holdAck_h			
Output hold, sysClkOut1_h to			
adr_h, data_h, check_h, cReq_h,	-1.5		ns
cWMask_h, holdAck_h			
Input Setup relative to sysClkOut1_h			
dRAck_h, dWSel_h, dOE_l	9.3		ns
cAck_h	Tcyc/2+6.0		ns
holdReq_h	4.8		ns
dInvReq_h, iAdr_h, , perf_cnt_h	4.5		ns
data_h, check_h	3.5		ns
Input Hold relative to sysClkOut1_h			
cAck_h, dRAck_h, dWSel_h, dOE_l	0		ns
data_h, check_h	0		ns
holdReq_h, dInvReq_h, iAdr_h	0		ns

The cAck_h input setup time is a function of the chip cycle time(Tcyc). At the nominal 6.6ns cycle time, required setup on the cAck_h pin is 9.3ns.

6.7 tagEq

When active during external cache hold, the timing of tagEq_l is specified from the time its inputs become valid at the pins.

Table 6–3: tagEq

Name	Min	Max	Units
Delay, adr_h -> tagEq_l		17.0	ns
Delay, tagAdr_h -> tagEq_l		17.0	ns

6.8 tagOk

The tagOk_h,_l signals are expected to be driven to the 21064 directly from the final stage of an ECL synchronizer clocked by cpuClkOut_h. As in the case of fast external cache cycles, the system must meet a maximum flow-through delay. This delay is defined with respect to the circuits below.

Figure 6–4: Flow-Through Delay

```
           | cpuClkOut_h                     | cpuClkOut_h .----------.
   21064 |----RRRR----+----.        21064 |-------------|          |
     PIN |    50ohms   |    | 10pF    PIN |             |          |
         |             |  -----          |             | External |
                       |  -----                        |          |
    Vdd-2.0V           |    |                          |  Logic   |
        O----RRRR----'     V               | tagOk_h,_l  |          |
             100ohms                21064 |-------------|          |
                                      PIN |             `----------'
                                          |
```

Assume that cpuClkOut_h is driven from the same internal clock edge in the two cases above. External flow-through delay is defined as the delay between cpuClkOut_h valid to the 10pF ECL "standard" load in the case on the left and tagOK_h,_l valid to the 21064 in the case on the right. It can not exceed the nominal cpu cycle time less 3.9ns. Note that board resistors must be part of "external logic" in the circuit on the right. For purposes of this specification, cpuClkOut_h is considered valid when it crosses the ECL threshold "Vbb" (equal to roughly Vcc - 1.3V) and tagOk is considered valid when the differential lines cross each other.

6.9 Tester Considerations

6.9.1 Inputs

The signals reset_l, irq_h, and sRomD_h (in serial port mode) are asynchronous during normal system operation. However, for test purposes they should be driven synchronously with sysClkOut1_h with the timing given below. Note that these parameters are given with respect to the time at which the rising edge of sysClkOut1_h crosses 0.8V.

Table 6–4: Asynchronous Signals on a Tester

Name	Min	Max	Units
Setup, reset_l -> sysClkOut1_h	5.0		ns
Setup, irq_h -> sysClkOut1_h	5.0		ns
Hold, irq_h -> sysClkOut1_h	0		ns
Setup, sRomD_h -> sysClkOut1_h	5.0		ns
Hold, sRomD_h -> sysClkOut1_h	0		ns

6.9.2 Signals Timed from Cpu Clock

Due to the speed of the 21064, it is expected that at-speed testing will be done with tester cycle equal to system cycle (i.e. sysClkOut1_h). However, fast external cache operation and serial ROM operation are timed from internal cpu clock. Therefore, input sampling, output enabling, and switching can occur at different time points within a tester cycle from one cycle to the next. Fortunately, the number of such points is finite, equal to the number of cpu cycles per tester cycle. For any given transaction, each signal will have its standard external cycle timing with respect to the rising edge of sysClkOut1_h OR to a "phantom" edge offset from sysClkOut1_h by exactly an integer number of cpu cycles. (Note that dataA_h, dataCEOE_h, dataWE_h, tagCEOE_h, tagCtlWE_h, and dMapWE_h have the same delay timing as adr_h.) Therefore, outputs may be sampled deterministically with appropriate placement of the tester strobe and inputs may be received deterministically with appropriate placement of the drive edge. Bidirectional signals present a different problem. Because the tester can enable or disable a given driver at just one point within its cycle, it must in the worst case drive an input beyond its 21064 sample point by at least (N-1) cpu cycles, where N is the number of cpu cycles per system cycle. However, in the worst case the 21064 will enable its drivers just one cpu cycle after sampling (for example, tagCtl_h following probe write). Therefore, the number of cpu cycles per system cycle must not exceed two to avoid driver conflict between the 21064 and the tester.

The serial ROM outputs sRomOE_l and sRomClk_h can be strobed with the same timing as the data_h pins when driven by the 21064. The serial ROM input sRomD_h can be switched with the same timing used in serial port mode.

Chapter 7

Package Information

Figure 7–1 shows the package physical dimensions without heat sink. Figure 7–2 shows pin locations.

Figure 7–1: Package Dimensions

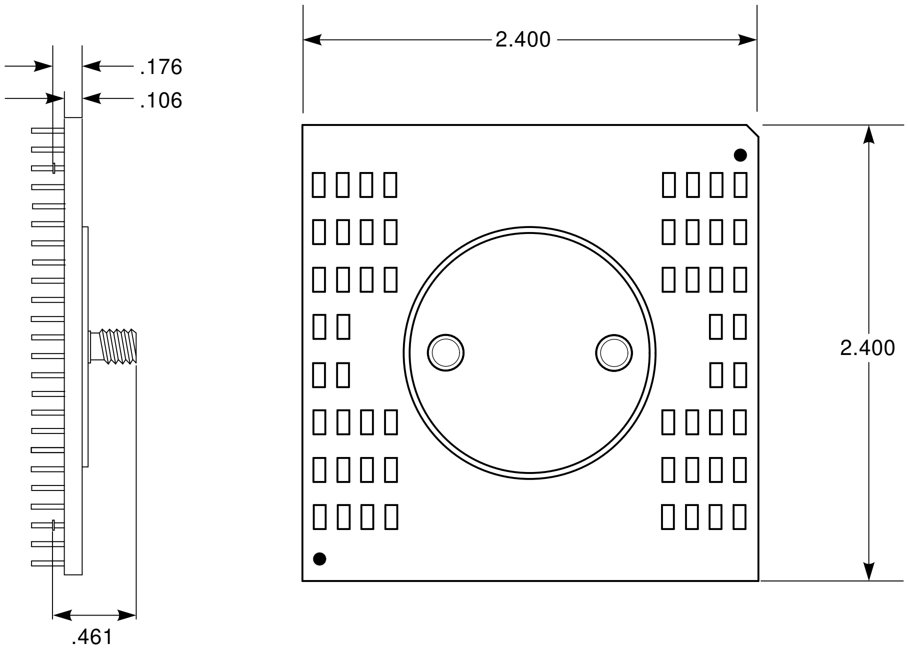
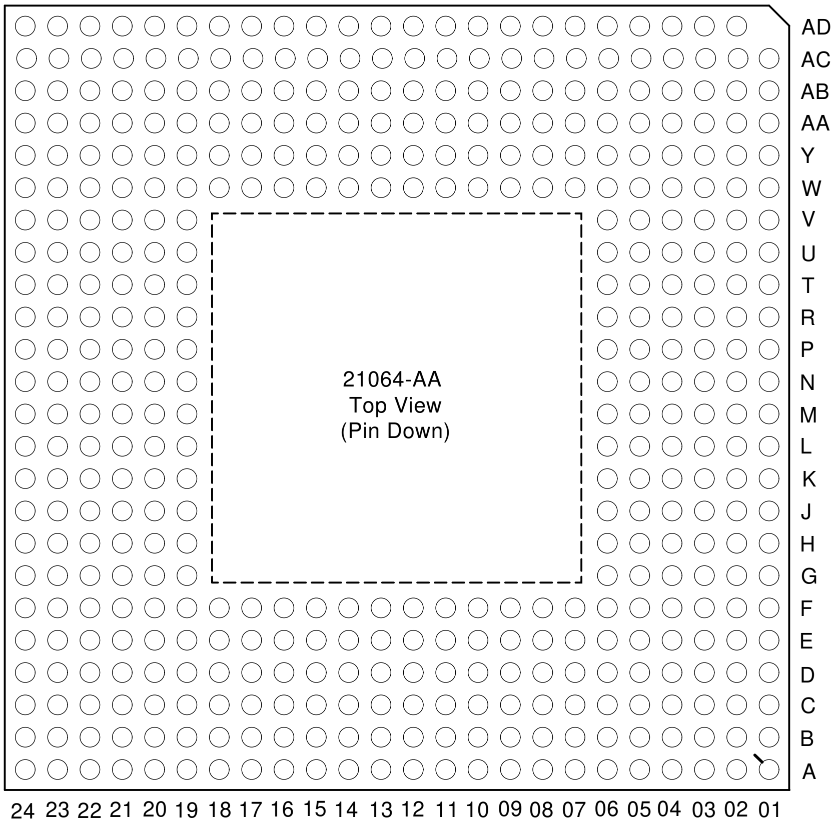


Figure 7–2: PGA Cavity Down View



Chapter 8

Pinout

8.1 Overview

This chapter contains the entire 21064 pinout. The order is by PGA location. Int the "TYPE" column, B = Bidirectional, I = Input, N = Not connected, P = Power or Ground, and O = Output.

8.2 21064 Pinout

Table 8–1: 21064 Pin List

PGA Loc.	PIN No.	Type	Name
A1	001	B	data_h 33
A2	002	B	data_h 97
A3	003	B	data_h 98
A4	004	B	data_h 100
A5	005	B	data_h 38
A6	006	B	check_h 27
A7	007	B	data_h 104
A8	008	B	data_h 42
A9	009	B	data_h 44
A10	010	B	data_h 109
A11	011	B	data_h 47
A12	012	B	data_h 49
A13	013	B	data_h 113

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
A14	014	B	data_h 52
A15	015	B	check_h 12
A16	016	B	data_h 55
A17	017	B	data_h 120
A18	018	B	data_h 122
A19	019	B	check_h 7
A20	020	B	data_h 60
A21	021	B	data_h 61
A22	022	B	data_h 62
A23	023	B	data_h 127
A24	024	B	check_h 9
B1	025	B	check_h 15
B2	026	P	VDD plane
B3	027	B	data_h 35
B4	028	P	VSS plane
B5	029	B	data_h 101
B6	030	P	VDD plane
B7	031	B	data_h 40
B8	032	P	VSS plane
B9	033	B	data_h 107
B10	034	P	VDD plane
B11	035	B	data_h 110
B12	036	P	VSS plane
B13	037	B	data_h 50
B14	038	P	VDD plane
B15	039	B	check_h 26
B16	040	P	VSS plane
B17	041	B	data_h 57
B18	042	P	VDD plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
B19	043	B	check_h 21
B20	044	P	VSS plane
B21	045	B	data_h 125
B22	046	P	VDD plane
B23	047	P	VSS plane
B24	048	B	check_h 8
C1	049	B	check_h 16
C2	050	P	VSS plane
C3	051	B	data_h 96
C4	052	B	data_h 99
C5	053	B	data_h 37
C6	054	B	check_h 13
C7	055	B	data_h 103
C8	056	B	data_h 105
C9	057	B	data_h 43
C10	058	B	data_h 45
C11	059	B	data_h 46
C12	060	B	data_h 112
C13	061	B	data_h 114
C14	062	B	data_h 116
C15	063	B	data_h 54
C16	064	B	data_h 119
C17	065	B	data_h 121
C18	066	B	check_h 11
C19	067	B	data_h 59
C20	068	B	data_h 124
C21	069	B	data_h 126
C22	070	B	check_h 23
C23	071	I	dRAck_h 0

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
C24	072	N	spare 3
D1	073	B	data_h 94
D2	074	B	check_h 2
D3	075	B	check_h 1
D4	076	B	data_h 34
D5	077	B	data_h 36
D6	078	B	data_h 102
D7	079	B	data_h 39
D8	080	B	data_h 41
D9	081	B	data_h 106
D10	082	B	data_h 108
D11	083	B	check_h 24
D12	084	B	data_h 48
D13	085	B	data_h 51
D14	086	B	data_h 53
D15	087	B	data_h 118
D16	088	B	data_h 56
D17	089	B	data_h 58
D18	090	B	check_h 25
D19	091	B	data_h 123
D20	092	B	data_h 63
D21	093	B	check_h 22
D22	094	I	dRAck_h 2
D23	095	P	VDD plane
D24	096	I	dOE_l
E1	097	B	data_h 30
E2	098	P	VDD plane
E3	099	B	data_h 31
E4	100	B	data_h 32

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
E5	101	P	VDD plane
E6	102	P	VSS plane
E7	103	P	VDD plane
E8	104	P	VSS plane
E9	105	P	VDD plane
E10	106	P	VSS plane
E11	107	B	check_h 10
E12	108	B	data_h 111
E13	109	B	data_h 115
E14	110	B	data_h 117
E15	111	P	VDD plane
E16	112	P	VSS plane
E17	113	P	VDD plane
E18	114	P	VSS plane
E19	115	P	VDD plane
E20	116	P	VSS plane
E21	117	I	dRAck_h 1
E22	118	I	dWSel_h 0
E23	119	I	dWSel_h 1
E24	120	I	cAck_h 0
F1	121	B	data_h 92
F2	122	B	data_h 29
F3	123	B	data_h 93
F4	124	B	data_h 95
F5	125	P	VSS plane
F6	126	P	VDD plane
F7	127	P	VSS plane
F8	128	P	VDD plane
F9	129	P	VSS plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
F10	130	P	VDD plane
F11	131	P	VSS plane
F12	132	P	VDD plane
F13	133	P	VSS plane
F14	134	P	VDD plane
F15	135	P	VSS plane
F16	136	P	VDD plane
F17	137	P	VSS plane
F18	138	P	VDD plane
F19	139	P	VSS plane
F20	140	P	VDD plane
F21	141	I	cAck_h 1
F22	142	I	cAck_h 2
F23	143	P	VSS plane
F24	144	I	holdReq_h
G1	145	B	data_h 27
G2	146	P	VSS plane
G3	147	B	data_h 91
G4	148	B	data_h 28
G5	149	P	VDD plane
G6	150	P	VSS plane
G19	151	P	VDD plane
G20	152	P	VSS plane
G21	153	O	holdAck_h
G22	154	O	dataCEOE_h 0
G23	155	O	dataCEOE_h 1
G24	156	O	dataCEOE_h 2
H1	157	B	check_h 4
H2	158	B	check_h 18

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
H3	159	B	check_h 0
H4	160	B	check_h 14
H5	161	P	VSS plane
H6	162	P	VDD plane
H19	163	P	VSS plane
H20	164	P	VDD plane
H21	165	O	dataCEOE_h 3
H22	166	O	tagCtlWE_h
H23	167	P	VDD plane
H24	168	O	cWMask_h 0
J1	169	B	data_h 89
J2	170	P	VDD plane
J3	171	B	data_h 26
J4	172	B	data_h 90
J5	173	P	VDD plane
J6	174	P	VSS plane
J19	175	P	VDD plane
J20	176	P	VSS plane
J21	177	O	cWMask_h 1
J22	178	O	cWMask_h 2
J23	179	O	cWMask_h 3
J24	180	O	cWMask_h 4
K1	181	B	data_h 87
K2	182	B	data_h 24
K3	183	B	data_h 88
K4	184	B	data_h 25
K5	185	P	VSS plane
K6	186	P	VDD plane
K19	187	P	VSS plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
K20	188	P	VDD plane
K21	189	O	cWMask_h 5
K22	190	O	cWMask_h 6
K23	191	P	VSS plane
K24	192	O	cWMask_h 7
L1	193	B	check_h 19
L2	194	P	VSS plane
L3	195	B	data_h 22
L4	196	B	data_h 86
L5	197	B	data_h 23
L6	198	P	VSS plane
L19	199	P	VDD plane
L20	200	O	dataWE_h 0
L21	201	O	dataWE_h 1
L22	202	O	dataWE_h 2
L23	203	O	dataWE_h 3
L24	204	O	dMapWE_h
M1	205	B	data_h 20
M2	206	B	data_h 84
M3	207	B	data_h 21
M4	208	B	data_h 85
M5	209	B	check_h 5
M6	210	P	VDD plane
M19	211	P	VSS plane
M20	212	O	cReq_h 0
M21	213	O	cReq_h 1
M22	214	O	cReq_h 2
M23	215	P	VDD plane
M24	216	N	spare 0

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
N1	217	B	data_h 83
N2	218	P	VDD plane
N3	219	B	data_h 19
N4	220	B	data_h 82
N5	221	B	data_h 18
N6	222	P	VSS plane
N19	223	P	VDD plane
N20	224	I	tagOk_l
N21	225	I	tagOk_h
N22	226	O	dataA_h 4
N23	227	O	dataA_h 3
N24	228	O	tagCEOE_h
P1	229	B	data_h 81
P2	230	B	data_h 17
P3	231	B	data_h 80
P4	232	B	data_h 16
P5	233	B	data_h 79
P6	234	P	VDD plane
P19	235	P	VSS plane
P20	236	B	tagCtlS_h
P21	237	B	tagCtlD_h
P22	238	B	tagCtlP_h
P23	239	P	VSS plane
P24	240	O	tagEq_l
R1	241	B	data_h 15
R2	242	P	VSS plane
R3	243	B	data_h 78
R4	244	B	data_h 14
R5	245	P	VDD plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
R6	246	P	VSS plane
R19	247	P	VDD plane
R20	248	P	VSS plane
R21	249	I	tagadr_h 19
R22	250	I	tagadr_h 18
R23	251	I	tagadr_h 17
R24	252	B	tagCtlV_h
T1	253	B	check_h 17
T2	254	B	check_h 3
T3	255	B	data_h 77
T4	256	B	data_h 13
T5	257	P	VSS plane
T6	258	P	VDD plane
T19	259	P	VSS plane
T20	260	P	VDD plane
T21	261	I	tagadr_h 22
T22	262	I	tagadr_h 21
T23	263	P	VDD plane
T24	264	I	tagadr_h 20
U1	265	B	data_h 76
U2	266	P	VDD plane
U3	267	B	data_h 12
U4	268	B	data_h 75
U5	269	P	VDD plane
U6	270	P	VSS plane
U19	271	P	VDD plane
U20	272	P	VSS plane
U21	273	I	tagadr_h 26
U22	274	I	tagadr_h 25

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
U23	275	I	tagadr_h 24
U24	276	I	tagadr_h 23
V1	277	B	data_h 11
V2	278	B	data_h 74
V3	279	B	data_h 10
V4	280	B	data_h 73
V5	281	P	VSS plane
V6	282	P	VDD plane
V19	283	P	VSS plane
V20	284	P	VDD plane
V21	285	I	tagadr_h 29
V22	286	I	tagadr_h 28
V23	287	P	VSS plane
V24	288	I	tagadr_h 27
W1	289	B	data_h 9
W2	290	P	VSS plane
W3	291	B	data_h 72
W4	292	B	check_h 6
W5	293	P	VDD plane
W6	294	P	VSS plane
W7	295	P	VDD plane
W8	296	P	VSS plane
W9	297	I	testClkIn_h
W10	298	I	testClkIn_l
W11	299	P	VDD plane
W12	300	I	clkIn_h
W13	301	I	clkIn_l
W14	302	P	VSS plane
W15	303	P	VDD plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
W16	304	P	VSS plane
W17	305	P	VDD plane
W18	306	P	VSS plane
W19	307	P	VDD plane
W20	308	P	VSS plane
W21	309	I	tagadrP_h
W22	310	I	tagadr_h 32
W23	311	I	tagadr_h 31
W24	312	I	tagadr_h 30
Y1	313	B	data_h 8
Y2	314	B	data_h 71
Y3	315	B	data_h 7
Y4	316	B	data_h 68
Y5	317	P	VSS plane
Y6	318	P	VDD plane
Y7	319	P	VSS plane
Y8	320	P	VDD plane
Y9	321	P	VSS plane
Y10	322	P	VDD plane
Y11	323	P	VSS plane
Y12	324	P	VDD plane
Y13	325	P	VSS plane
Y14	326	P	VDD plane
Y15	327	P	VSS plane
Y16	328	P	VDD plane
Y17	329	P	VSS plane
Y18	330	P	VDD plane
Y19	331	P	VSS plane
Y20	332	P	VDD plane

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
Y21	333	B	adr_h 8
Y22	334	B	adr_h 5
Y23	335	P	VDD plane
Y24	336	I	tagadr_h 33
AA1	337	B	check_h 20
AA2	338	P	VDD plane
AA3	339	B	data_h 5
AA4	340	B	data_h 66
AA5	341	B	data_h 0
AA6	342	I	iAdr_h 6
AA7	343	I	iAdr_h 10
AA8	344	I	vRef
AA9	345	O	sysClkOut2_h
AA10	346	O	sysClkOut2_l
AA11	347	N	spare 6
AA12	348	O	sysClkOut1_h
AA13	349	O	sysClkOut1_l
AA14	350	I	cont_l
AA15	351	I	irq_h 5
AA16	352	N	spare 8
AA17	353	B	adr_h 31
AA18	354	B	adr_h 27
AA19	355	B	adr_h 24
AA20	356	B	adr_h 17
AA21	357	B	adr_h 15
AA22	358	B	adr_h 11
AA23	359	B	adr_h 7
AA24	360	B	adr_h 6
AB1	361	B	data_h 70

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
AB2	362	B	data_h 69
AB3	363	B	data_h 67
AB4	364	B	data_h 2
AB5	365	B	data_h 64
AB6	366	I	iAdr_h 7
AB7	367	I	iAdr_h 12
AB8	368	I	reset_l
AB9	369	I	sRomD_h
AB10	370	O	sRomOE_l
AB11	371	O	cpuClkOut_h
AB12	372	I	dcOk_h
AB13	373	I	triState_l
AB14	374	I	icMode_h 0
AB15	375	I	irq_h 4
AB16	376	I	perf_cnt_h 0
AB17	377	B	adr_h 32
AB18	378	B	adr_h 28
AB19	379	B	adr_h 25
AB20	380	B	adr_h 21
AB21	381	B	adr_h 18
AB22	382	B	adr_h 14
AB23	383	P	VSS plane
AB24	384	B	adr_h 9
AC1	385	B	data_h 6
AC2	386	P	VSS plane
AC3	387	P	VDD plane
AC4	388	B	data_h 65
AC5	389	P	VSS plane
AC6	390	I	iAdr_h 8

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
AC7	391	P	VDD plane
AC8	392	I	iAdr_h 11
AC9	393	P	VSS plane
AC10	394	O	sRomClk_h
AC11	395	P	VDD plane
AC12	396	N	spare 5
AC13	397	P	VSS plane
AC14	398	I	irq_h 2
AC15	399	P	VDD plane
AC16	400	I	perf_cnt_h 1
AC17	401	P	VSS plane
AC18	402	B	adr_h 29
AC19	403	P	VDD plane
AC20	404	B	adr_h 22
AC21	405	P	VSS plane
AC22	406	B	adr_h 16
AC23	407	P	VDD plane
AC24	408	B	adr_h 10
AD2	409	B	data_h 4
AD3	410	B	data_h 3
AD4	411	B	data_h 1
AD5	412	I	iAdr_h 5
AD6	413	I	iAdr_h 9
AD7	414	N	spare 1
AD8	415	I	eclOut_h
AD9	416	I	dInvReq_h
AD10	417	N	spare 2
AD11	418	N	spare 4
AD12	419	I	icMode_h 1

Table 8–1 (Cont.): 21064 Pin List

PGA Loc.	PIN No.	Type	Name
AD13	420	I	irq_h 0
AD14	421	I	irq_h 1
AD15	422	I	irq_h 3
AD16	423	N	spare 7
AD17	424	B	adr_h 33
AD18	425	B	adr_h 30
AD19	426	B	adr_h 26
AD20	427	B	adr_h 23
AD21	428	B	adr_h 20
AD22	429	B	adr_h 19
AD23	430	B	adr_h 13
AD24	431	B	adr_h 12