UltraSPARC® IV+ Processor

User's Manual Supplement

Version 1.0

October 2005

Table of Contents

List of Tables

List of Figures

Preface

This book contains information about the architecture and programming of the UltraSPARC® IV+ processor, one of Sun Microsystems' family of SPARC® V9 compliant processors. This document is a supplement to the *UltraSPARC III Cu Processor User's Manual* and should be read in conjunction with that document.

This document extends the material in the *UltraSPARC III Cu Processor User's Manual*. Any material that is not referred to in this document remains unchanged for the UltraSPARC IV+ processor. Any material that overrides or extends the material in the *UltraSPARC III Cu Processor User's Manual* should be read from this document.

Target Audience

This user's manual is mainly targeted for programmers who write software for the UltraSPARC IV+ processor. This user's manual supplement contains a depository of information that is useful to operating system programmers, application software programmers, logic designers, and third party vendors, who are trying to understand the architecture and operation of the UltraSPARC IV+ processor. This supplement is both a guide and a reference manual for low-level programming of the processor.

Prerequisites

This user's manual is a companion to the *UltraSPARC III Cu Processor User's Manual*. The reader of this user's manual should be familiar with the contents of the *UltraSPARC III Cu Processor User's Manual*.

Textual Usage

Fonts

Fonts are used as follows:

- *Emphasis* is used for exceptions, traps and errors as well as book titles.
- `Courier` is used for all fields in the registers, register names, instructions, and read-only register fields. "The `rs1` field contains..." is an example of how this font is used.
- UPPERCASE items are acronyms, instruction names, or writable register fields. **Note:** Names of some instructions contain both upper- and lowercase letters.

- Underbar characters join words in register, register field, exception, and trap names. **Note:** Such words can be split across lines at the underbar without an intervening hyphen. “This is true whenever the integer_condition_code field...” is an example of how the underbar characters are used.

Notational Conventions

The following notational conventions are used:

- Square brackets, [ ], indicate a numbered register in a register file. For example, r[0] translates to register 0, indicate a bit number or colon-separated range of bit numbers within a field. “Bits FSR[29:28] and FSR[12] are...” is an example of how the angle brackets are used.
- Curly braces, { }, indicate textual substitution. For example, the string “PRIMARY{_LITTLE}” expands to “ASI_PRIMARY” and “ASI_PRIMARY_LITTLE.”
- If the bar, |, is used with the curly braces, it represents multiple substitutions. For example, the string “ASI_DMMU_TSB_{8 KB|64 KB|DIRECT}_PTR_REG” expands to “ASI_DMMU_TSB_8 KB_PTR_REG”, “ASI_DMMU_TSB_64 KB_PTR_REG”, and “ASI_DMMU_TSB_DIRECT_PTR_REG”.
- The symbol designates concatenation of bit vectors. A comma (**,**) on the left side of an assignment separates quantities that are concatenated for the purpose of assignment. For example, if X, Y, and Z are 1-bit vectors and the 2-bit vector T equals 11_2, then

  $(X, Y, Z) \leftarrow 0 \ \| \ T$

  results in X = 0, Y = 1, and Z = 1.
- “A mod B” means “A modulus B”, where the calculated value is the remainder when A is divided by B.
- “X” and “x” are used to represent states or bits that are either not used or are not relevant (i.e., “don’t care condition”); “X” usually indicates that a state may be either “Yes” or “No” (“True” or “False”), while “x” indicates the bit may be either a 1 or a 0.

Notation for Numbers

Numbers throughout this specification are decimal (base-10) unless otherwise indicated. Numbers in other bases are followed by a numeric subscript indicating their base (for example, 1001_2, FFFF 0000_{16}). In some cases, numbers may be preceded by “0x” to indicate hexadecimal (base-16) notation (for example, 0xFFFF.0000). Long binary and hexadecimal numbers within the text have spaces or periods inserted every four characters to improve readability.

The notation 7h’1F indicates a hexadecimal number of $1F_{16}$ with 7 binary bits of width.

Informational Notes

This guide provides several different types of information in notes, as follows:

Note – This highlights a useful note regarding important and informative processor architecture or functional operation. This may be used for purposes not covered in one of the other notes.

1

Architectural Overview

This chapter provides an overview of the UltraSPARC IV+ processor, focusing on its differences from the UltraSPARC® III and UltraSPARC® IV processors.

The UltraSPARC IV+ processor is a high-performance processor targeted at the enterprise server market and is intended as an upgrade product for the Sun™ Fire family of servers. Compared with the UltraSPARC III/IV processors, the UltraSPARC IV+ processor has an equivalent footprint, fits within a compatible thermal and power budget, and is designed to be incorporated into interchangeable motherboards based on the Sun™ Fireplane Interconnect.

Chapter Topics

1.1 Introduction

Although the UltraSPARC IV+ processor is primarily targeted at addressing the demands of commercial computing, it offers substantial benefits across a wide spectrum of workloads, including technical computing. Within the constraints imposed by building from the common pipeline and the common bus protocol that defines membership in the UltraSPARC III/IV processor family, every effort has been made to optimize the design of the UltraSPARC IV+ processor. Key features of the processor include:

- Chip Multithreading with two upgraded UltraSPARC III processor cores per chip
- Implemented three levels of cache hierarchy
- Enhanced memory controller and system interface unit
- Enhanced error detection and correction

1.2 Chip Multithreading (CMT)

Many of the workloads for the midrange and enterprise server markets exhibit a high degree of thread-level parallelism (TLP). These workloads consist of many independent tasks or *threads* that can be partitioned to run on separate logical processors[1]. These threads scale well in performance as the number of logical processors available is increased (up to the limit of available threads). This characteristic behavior of enterprise workloads will be exploited by future UltraSPARC processors specifically designed to take advantage of TLP in code.

One of the most effective ways to exploit TLP at the processor level is through Chip Multithreading technology. CMT processors incorporate multiple logical processors onto a single chip. Sun Microsystems, a longtime leader in providing support for threads at the operating system level and in creating symmetric multiprocessor (SMP) systems for running threaded workloads, is now taking a leading role in developing CMT processor technology and driving support for threads down to the basic hardware level. Sun's MAJC™ 5200 processor (shipped in 2000) was one of the first commercial CMT products. The UltraSPARC IV+ processor will be the second CMT processor in the UltraSPARC III/IV family (after the UltraSPARC IV processor).

Relative to the UltraSPARC III processor generation, the UltraSPARC IV processor generation achieves a large leap in thread-level parallelism by integrating two UltraSPARC III processor cores onto a single chip. Using UltraSPARC IV processors, products built for the UltraSPARC III family of processors effectively deliver twice as many logical processors in the same system. The dual-core UltraSPARC IV processors are designed to be compatible with the single-core UltraSPARC III processors in terms of both spatial and thermal footprint. It is therefore possible to upgrade a Sun Fire system based on UltraSPARC III processors to UltraSPARC IV processors. This upgrade results in a significant increase in throughput per cubic foot, per watt, and per dollar for that system.

Relative to the initial UltraSPARC IV processor, built in 130 nm process technology, the UltraSPARC IV+ processor takes advantage of the greatly expanded transistor budget possible with 90 nm technology to provide a thorough upgrade of the initial UltraSPARC IV processor dual-core design.

1.3 Enhanced Core Design

Compared with its predecessors, the UltraSPARC IV+ processor's core has been optimized in a number of important ways. The enhancements include improvements in the following processor resources:

- Instruction fetch
- Execution units
- Write cache
- Data prefetching
- Memory management units

1.Defined in Chapter 2.

1.3.1 Instruction Fetch

The UltraSPARC IV+ processor's instruction cache (I-cache) has been doubled in capacity, from 32 KB to 64 KB, and its line size also has been doubled from 32 bytes to 64 bytes. The larger capacity significantly improves the hit rate for programs whose instruction stream exhibits good temporal locality, while the longer line length helps programs whose instruction stream exhibits good spatial locality.

The expanded I-cache is augmented with a much more aggressive instruction prefetch mechanism, based on an 8-entry prefetch buffer (where each entry is a full 64-byte line). The prefetch buffer is accessed in parallel with the I-cache and, in the case of a prefetch buffer hit on an I-cache miss, the prefetched line is filled into the instruction cache. A prefetch of the next sequential line (address *N*+64) into the prefetch buffer is triggered by one of two conditions: a request for address *N* that either misses altogether in the L1 I-cache (including prefetch buffer) or hits in the prefetch buffer. In addition to reducing the overall latency of instruction fetching, the instruction prefetch mechanism provides more robust performance for applications whose instruction working set exceeds the capacity of the instruction cache.

The UltraSPARC IV+ processor benefits not only from more aggressive instruction prefetching, but also from superior mechanisms for predicting both the direction and target of branch instructions. The branch direction predictor has been made configurable, allowing different mechanisms to be used for code of different types. In addition to a standard Gshare predictor of the sort used in UltraSPARC III processor, two separate history registers are available for privileged (supervisor) and user code. Further, either or both of the history registers can be disabled in favor of a PC-indexed branch predictor. While a standard Gshare predictor works well for smaller applications, a PC-indexed predictor often works better for large, irregular applications, like large databases, as well as for privileged code in general.

To improve target prediction for indirect branches (branches whose targets are specified by a register value), the UltraSPARC IV+ processor incorporates a 32-entry branch target buffer (BTB). When an indirect branch is encountered, the BTB is used in conjunction with the return address stack and the branch preparation instruction to predict the target instruction.

1.3.2 Execution Units

The UltraSPARC IV+ processor's integer execution unit has been augmented with a new special unit that uses the load/store pipe to execute the population count (`POPC` instruction) in hardware.

The UltraSPARC IV+ processor's floating-point functional units have been redesigned to handle many more special functions and IEEE exceptions directly in hardware than did previous members of the UltraSPARC III/IV processor family (which trapped to a software handler on certain IEEE exceptions). For example, the UltraSPARC IV+ processor handles in hardware both: (1) integer-to-floating-point conversions, where the relevant bits of the integer are more than the bits of mantissa, and (2) operations in non-standard mode that produce subnormal results. Programs that generate significant numbers of operations affected by this upgrade should experience substantially improved performance on the UltraSPARC IV+ processor.

1.3.3 Write Cache

Like previous family members, each of the UltraSPARC IV+ processor cores has an associated 2 KB write cache. Write-through stores from a core's L1-cache are written into the write cache rather than directly to the L2-cache. If a store misses in the write cache, the missed line is brought into the cache. If the write cache is already full, the oldest line in the cache is evicted to L2 to make room for the new line.

In previous family members, all of which were characterized by an off-chip L2-cache, the write cache critically served to minimize the amount of off-chip traffic to L2 generated by a core's write-through L1-cache. For this reason, a store generated no off-chip traffic until the written line eventually was evicted from the write cache. On the first write to a line, line ownership was transferred by accessing the on-chip L2 tags. To maximize the capacity of the write cache, just those bytes of a line actually updated by the store stream were cached. Only on eviction was it finally necessary to go off chip, performing a background read of the entire original line in L2, merging any unmodified bytes from that line with the modified bytes held in the write cache, and writing the complete updated line back to L2.

In the UltraSPARC IV+ processor, with the L2-cache brought on-chip, the original function of the write cache (reducing off-chip store traffic) has disappeared, making it possible to streamline write cache operations. In effect, in the UltraSPARC IV+ processor, the write cache functions largely as a 32-entry expansion of the 8-entry store queue. On a store transaction, when the store misses in the write cache, a single read transaction is issued to the L2-cache that both reads in the entire 64-byte line and gets ownership of it before overwriting modified bytes with the store. Subsequent writes to that line update only the copy held in the write cache. When the line is eventually evicted, a single write transaction is made to L2 (without need for any associated background read and merge operations). Storing entire lines in the write cache (unmodified in addition to modified bytes) is less conservative of write cache space but, for an on-chip L2, the associated increase in write traffic is a minor penalty compared to the benefits of greatly simplified write cache operation.

The write cache in the UltraSPARC IV+ processor is fully set-associative and uses a FIFO allocation policy. This makes the cache much more robust for applications that have multiple store streams and enables the write cache to better exploit temporal locality in the store stream.

1.3.4 Data Prefetching

Because all members of the UltraSPARC III/IV processor family block access to the L1-cache on a miss, data prefetching is an important feature for exploiting memory-level parallelism. In the UltraSPARC IV+ processor, the prefetch mechanisms have been improved in a number of ways. The most important optimizations are focused on making the behavior of software prefetches more predictable, thus making it easier for the compiler or a programmer to use these instructions. Prefetching efficiency also has been improved by optimizing a number of steps in the prefetch process, thereby reducing the latency of prefetch operations.

To make prefetch behavior more predictable, the UltraSPARC IV+ processor supports a new class of prefetch operation: strong prefetches. Strong prefetches are similar to regular software prefetches but will succeed under a wider range of conditions. First, if a strong prefetch has a translation lookaside buffer (TLB) miss, instead of simply dropping the prefetch, the processor will take a trap to fill the TLB and then re-issue the prefetch. Second, if the prefetch queue is full, instead of potentially dropping the prefetch, the processor will wait until one of the outstanding prefetches completes and then place the prefetch in queue. Strong prefetch allows software to use prefetch for critically needed items with a high degree of confidence that the item requested will, in fact, be loaded in advance of use.

1.3.5 Memory Management Units (MMU)

While the general MMU organization remains the same in the UltraSPARC IV+ processor cores as in the cores of previous family members, both the number of TLB entries in the instruction MMU and the page sizes supported by the MMUs have been increased.

Instruction MMU. The instruction MMU of the UltraSPARC IV+ processor core incorporates two TLBs: a small, fully associative, 16-entry TLB and a large 2-way set-associative TLB. In earlier family members, the large I-TLB had a total of 128 entries. In the UltraSPARC IV+ processor, this total has been increased to 512 entries. The UltraSPARC IV+ processor can work with either a base page size of 8 KB or 64 KB.

Data MMU. The data MMU in the UltraSPARC IV+ processor still supports the same three TLBs as earlier family members: a small, fully-associative, 16-entry TLB and two large, 2-way set-associative, 512-entry TLBs. However, in the UltraSPARC IV+ processor, one large D-TLB continues to support page sizes of 8 KB, 64 KB, 512 KB, and 4 MB, while the second of the two large D-TLBs has been modified to support page sizes of 8KB, 64 KB, 32 MB and 256 MB.

The new large page sizes allow the UltraSPARC IV+ processor to support applications that need to map extremely large data sets. A TLB can only access/fill pages of one size at a time, but the two large TLBs are each programmable and may be independently set to support either the same page size or different page sizes. Thus, for systems with very large memories, one TLB can be set to handle “default” pages of either 8 or 64 KB, while the other TLB handles large pages of 32 or 256 MB. Whereas for systems with smaller memories, both TLBs can be set to handle the “default” page size, doubling the number of entries available for mapping smaller pages.

1.4 Cache Hierarchy

The cache hierarchy supported by the UltraSPARC IV+ processor has been completely revised. The cache hierarchy has been expanded from two to three levels (L1, L2, L3).

1.4.1 L1 Cache

The L1 instruction cache (I-cache) was doubled in size, from 32 KB to 64 KB, in the UltraSPARC IV+ processor. The expanded I-cache has a 64-byte line, divided into two 32-byte subblocks with separate valid bits. Also, the 2 KB write cache has been made fully set-associative. The other two L1-caches (64 KB data and 2 KB prefetch) are unchanged. As in the UltraSPARC IV processor, all four L1-caches are duplicated with each core. To maintain data coherency, the L1 data cache is write-through.

1.4.2 L2 Cache

The UltraSPARC IV+ processor’s L2 cache has been reduced in size from 16 MB to 2 MB, but brought on-chip. The L2 cache is also shared, rather than split, between the two cores. The structure of the L2-cache has been revised to 4-way set-associative with a 64-byte line. The L2-cache operates at half the processor frequency, sustaining one read or write request every 2 cycles. With the exception of the prefetch cache, all L1-caches are included in the L2-cache. To minimize off-chip traffic, the L2-cache is copy-back.

1.4.3 L3 Cache

The first two levels of on-chip caches are backed by a large, off-chip L3-cache of 32MB, also shared by the two cores. Like the revised L2-cache, the new L3-cache is 4-way set-associative with a 64-byte line. To maximize access bandwidth and speed, the tags for the L3-cache are kept on-chip. To minimize traffic to main memory, the L3-cache is copy-back.

The L3-cache is a “victim” cache:

- When data or instructions are loaded from memory, they are written into the L2 and L1 on-chip caches, but not into the L3 off-chip cache.
- A line is written into L3 only when it is evicted from the L2-cache. Both clean and dirty (modified) lines are treated the same.

The L2 and L3-caches are exclusive:

- A line cannot be in both caches at the same time.
- A line evicted from L2 and written into L3 is no longer in L2.
- On a “hit” in L3, the line is copied back to the L2 and L1 levels of the cache hierarchy and marked as invalid in the L3-cache.
- Because L2 is not included in L3, in effect, the L2 and L3-caches provide a total of 34MB of secondary cache storage between them.
- Because the L2 and L3-caches are exclusive, either can be the source of data for cache-to-cache transactions.

Shared L2 and L3-caches offer significant benefits. When two running threads operate on common data, shared caches work to minimize access times and maximize hit rates for both. Even when two simultaneously running threads do not share data, shared caches enable more flexible allocation of space to each, according to need, and usually provide superior performance.

However, to handle the occasional case where two running threads are unable to cooperate to their mutual advantage – but instead antagonistically contend for cache space with the result that the performance of each is impaired – the UltraSPARC IV+ processor also provides a mechanism for pseudo-splitting its shared caches. In this mode, each thread is allocated its own half of the L2 and L3-cache resources, thereby avoiding any interference from one thread in the cache operations of the other thread. When operating in split cache mode, while each thread can still read all four-way sets of the L2 and L3-caches, the processor can only write into two of the four-way sets in L2 or L3.

1.5 System Interface and Memory Controller Enhancements

In addition to its revised and more aggressive cache hierarchy, the UltraSPARC IV+ processor has additional improvements to reduce the average latency of off-chip memory and I/O transactions.

1.5.1 Reduced Latency of Cache-to-Cache Transfers

The latency of cache-to-cache transfers has been reduced by overlapping some of the copyback latency with the snoop latency. This is important in symmetric multiprocessor (SMP) configurations because the large L2 (with dirty entries) and L3-caches will cause a significant percentage of requests to be satisfied by modified data currently held in other chips' caches.

1.5.2 Higher Sustainable Bandwidth of Foreign Writes

By a more optimal assignment of transaction identification, the overall sustainable bandwidth for foreign transactions in general, and write streams in particular, has been increased.

1.5.3 Larger Coherent Pending Queue

The size of the Coherent Pending Queue (CPQ), which holds outstanding coherent Sun Fireplane Interconnect transactions, has been increased from 24 entries in earlier family members to 32 entries in the UltraSPARC IV+ processor, reducing the need for frequent Sun Fireplane Interconnect flow control operations.

1.5.4 Supports Larger Main Memories

The memory controller in the UltraSPARC IV+ processor has been redesigned to support higher density DRAM, up to a maximum configuration of 32GB of memory per processor (applicable to both CK DIMM and NG DIMM). This compares with a maximum configuration of 16GB of memory per processor for earlier family members. In SMP configurations, the per processor memory capacity of the UltraSPARC III/IV processor family is additive. An SMP system based on four of the UltraSPARC IV+ processors could support a maximum of 128GB of memory, which is twice the 64GB possible with earlier family members.

1.6 Enhanced Error Detection and Correction

In addition to the many new features intended to improve performance, the reliability, availability, and serviceability (RAS) of the UltraSPARC IV+ processor have been significantly enhanced. Although the UltraSPARC IV+ processor design is much more complicated than the UltraSPARC III processor design, requiring a larger die and over ten times as many transistors to implement, it actually offers a lower fault rate than any previous family member.

All the large memory arrays in the UltraSPARC IV+ processor design have error detection and recovery logic associated with them. Like earlier family members, the UltraSPARC IV+ processor protects its "clean" write-through L1-caches with simple parity checking. If a data error is detected, the faulty line is invalidated and quickly reloaded with a good copy of the data from the L2-cache. In addition, the UltraSPARC IV+ processor improves the bit layout of the L1 instruction and data caches, specifically to reduce the probability of a single cosmic ray striking two bits in the same parity group, thereby causing an undetectable double-bit error. Also, parity protection has been added to smaller (previously unguarded) "clean" data structures, including the prefetch cache and the large 512-entry TLBs in both the I-MMU and D-MMU.

The on-chip L2-cache (both tag and data) as well as the on-chip L3-cache tags are protected by full error correcting code (ECC) that supports single-bit error correction and double-bit error detection.

While all members of the UltraSPARC III/IV processor family provide ECC for the external data buses, the UltraSPARC IV+ processor goes a step further by providing protection for the external address buses that connect the processor to its external cache and main memory.

2

Chip Multithreading (CMT)

The UltraSPARC IV+ processor supports Sun's new software interface and registers to support logical processor identification, reset, diagnostics, and error reporting. These CMT registers can be classified as *private* or *shared*.

Chapter Topics

2.1 Introduction

This chapter corresponds to Sun's common interface between hardware and software and addresses issues common to CMT processors.

The UltraSPARC IV+ processor uses Sun's standard CMT programming model, an interface specifying the basic functionality needed in the operating system, diagnostics, and recovery code to control and configure a processor comprised of multiple logical processors. Among other requirements, the CMT programming model defines how logical processors are identified, how errors and resets are steered to logical processors, and how logical processors can be disabled or suspended.

Logical processors are identified by two globally unique IDs: one used to reference the processor's registers and a second used to reference interrupts. The former ID is used to disable or suspend logical processors, while the latter is used for steering a thread's errors, resets and traps. These IDs enable CMT processors to behave much like traditional symmetrical multiprocessor systems.

When an error can be identified as being associated with a logical processor, the error will be reported to that logical processor. For errors that cannot be associated with any specific thread or logical processor, the CMT model defines an ASI register used by the operating system to steer the errors to a designated logical processor. Resets generated by an externally initiated reset (XIR) signal can be steered to an arbitrary subset of the logical processors by either the operating system or an external service processor, by setting the appropriate bit mask in an ASI register.

Logical processors can be enabled/disabled either by software or by an external service processor through an ASI register. This action only takes effect after a system reset. Logical processors can be suspended at any time by software or by an external service processor through an ASI register. Logical processors can suspend either themselves or other logical processors. When a logical

processor is suspended, it stops fetching instructions, completes any instructions it already has in its pipe, and then becomes idle. A suspended logical processor still fully participates in cache coherency transactions and remains coherent. When it is started again, it continues execution from the point of suspension. The ability to suspend logical processors is very important for diagnostic and recovery code and is used during the boot process to facilitate initial bring-up.

2.1.1 CMT Definition

A CMT processor is defined by its external visible nature and not its internal organization. The following section provides background terminology followed by a description of the CMT definition.

Background Terminology

Thread

The basic unit of program execution; a stream of computer instructions that constitutes the control flow of a process.

Logical Processor (LP)

The abstraction of a processor's architecture that maintains the state and management of an executing thread.

Core

A hardware unit that instantiates one or more logical processors. In addition to the basic execution pipeline(s) and associated registers, a core often includes L1 caches.

Processor

A single piece of silicon ("chip") that interprets and executes operating system functions and other software tasks. A processor is implemented by one or more cores.

Chip Multithreading (CMT)

A processor capable of executing 2 or more software threads simultaneously without resorting to a software context switch. Chip Multithreading may be achieved through the use of a single core able to execute multiple threads in parallel, or multiple cores each able to execute one or more parallel threads.

2.1.2 General CMT Behavior

In general, each logical processor of a CMT processor behaves functionally, from the viewpoint of software visibility, as if it was an independent unit. This is an important aspect of CMT because user code running on a logical processor need not know whether or not that logical processor is

just part of a single multithreaded CMT processor, or an individual single-threaded processor that has aggregated together with other separate single-threaded processors to form a Symmetric MultiProcessing (SMP) system. In either case, the operating system exploits logical processors to simultaneously schedule multiple threads of execution. Various low-level software programs, however – boot, error, diagnostic and other codes – must be aware of logical processors as elements of the same CMT processor chip. This chapter describes the interface between low-level software and the multiple logical processors that cohabit a CMT processor.

Logical processors obey the same memory model semantics as if they were independent processors. All multiprocessing libraries, thread libraries and code will be able to operate without modification on a CMT processor comprising ***N*** logical processors, in exactly the same way they operate on an SMP system composed of ***N*** independent processors.

Note – All previous documentation including the *UltraSPARC III Cu Processor User's Manual* and the SPARC V9 use the term *processor*. When these earlier documents are read in conjunction with this supplement, replace the term *processor* with *logical processor* to read them in context of the UltraSPARC IV+ processor.

2.2 Accessing CMT Registers

A key part of the CMT Programming Model is a set of specific, privileged registers. This section covers how these registers are organized and accessed. These registers can be read and written by any logical processor, when running in privileged mode, by using special Address Space Identifiers (ASIs) in specific load and store instructions.

ASIs are a feature of the SPARC instruction set, providing a convenient and flexible mechanism for mapping additional architectural states. SPARC architecture processors can access these additional states through special load and store instructions, that take an ASI value together with an address (virtual address). Certain ASI values cause an access to the corresponding address in physical memory, but with behavior different from the default semantics of normal load and store operations. Other ASI values are used to access special locations, reserved for storing configuration, diagnostic, or other vital information. The CMT Programming Model defines a number of new ASIs, used specifically for accessing the CMT-specific registers.

2.2.1 Types of CMT Registers

The two main classes of CMT-specific registers are: private registers and shared registers.

- Private registers: a private copy of the register is associated with each logical processor.
- Shared registers: a single copy of each register is shared by all the logical processors.

Both private and shared registers can be accessed as ASI-mapped registers by privileged software running on one of the logical processors. A logical processor can access only its own private registers, since it has no way to address the private registers of any other logical processor. The specific semantics for accessing the CMT registers through the ASI interface are described in *Accessing CMT Registers Through ASI Interface* on page 12.

2.2.2 Accessing CMT Registers Through ASI Interface

Each CMT-specific register is accessible through an ASI address – a combination of an address space identifier value and virtual address. All CMT registers are mapped into ASI values that are only accessible in privileged mode. The specific ASI number and virtual address of each CMT register is covered later in this document.

Each logical processor can access (only) its own private registers. Accesses by logical processors to their own associated private registers follow the standard semantics for accessing ASI mapped internal registers.

Each logical processor can access all the shared registers. An update to a shared register from one logical processor will be visible to all other logical processors. The ordering of accesses to shared registers from different logical processors is not defined, but there are a number of hardware rules that are enforced:

- The hardware guarantees that accesses to a shared register from the same logical processor follow sequential semantics.
- The hardware also guarantees that if multiple logical processors attempt to write the same shared register at the same time, after the updates, the register contains the value from just one of those writes. That is, stores to shared CMT registers must be performed atomically on all bits of the register.

All the CMT registers are 64-bit registers, although some of the bits of individual registers can be reserved or defined to a fixed value. *Reserved* register fields always should be written by software with values of those fields previously read from that register or with zeroes; they should read as zero in hardware. Software intended to run on future versions of CMTs should not assume that these fields will read as 0 or any other particular value. This software convention makes future expansion of the interface easier.

Only the Load extended from alternate space (`LDXA`) or Load double floating-point register from alternate space (`LDDFA`) instructions can be used to read CMT registers. Only the Store extended into alternate space (`STXA`) and the Store double floating-point register to alternate space (`STDFA`) instructions can be used to write to CMT registers. An attempt to access a CMT register with any other instruction results in a *data_access_exception* trap.

2.3 Private Processor Registers

There are three private registers used for logical processor identification.

2.3.1 LP ID Register (`ASI_CORE_ID`)

The LP ID register is a read-only, private register that holds the ID value assigned by hardware to each implemented logical processor. The ID value is unique within the CMT.

The LP ID register corresponds to a bit offset for corresponding bit mask CMT registers (like LP Enable register). Many of the CMT-specific registers provide a bit mask wherein each bit corresponds to an individual logical processor. For these registers, the LP_ID field indicates which bit of a bit mask corresponds to a specific logical processor.

Name: ASI_CORE_ID
ASI 0x63, VA[63:0] == 0x10,
Read-Only, Privileged Access

As described in the TABLE 2-1, the LP ID register has two fields.

TABLE 2-1 LP ID Register

Bit	Field	Description
[63:22]	*Reserved*	Reserved for future implementation.
[21:16]	MAX_LP_ID	Max LP ID, which gives the logical processor ID value of the highest numbered implemented, but not necessarily enabled, logical processor in this CMT processor. For the UltraSPARC IV+ processor, the value of this field is 1 because there are two logical processors.
[15:6]	*Reserved*	Reserved for future implementation.
[5:0]	LP_ID	A LP_ID field, which represents this logical processor's number, as assigned by the hardware. The LP ID is encoded in 6-bits. In the UltraSPARC IV+ processor, one logical processor has a value of 6'b000000; the other logical processor has a value of 6'b000001.

2.3.2 LP Interrupt ID Register (ASI_INTR_ID)

The LP Interrupt ID register, described in TABLE 2-2, is added to support the Sun Fireplane Interconnect interrupt transaction. This register is used to differentiate between logical processors when sending interrupts. This private register is used by software to assign a 10-bit interrupt ID to a logical processor that is unique within the system. This is important to enable logical processors to receive interrupts. The ID in this register is used by other logical processors and other bus agents to address interrupts to this specific logical processor. It is also used by this logical processor to identify the source of interrupts it issues to other logical processors and bus agents. It is expected to be changed only at boot or reconfiguration time.

Name: ASI_INTR_ID
ASI 0x63, VA[63:0] == 0x00,
Read-Write, Privileged Access

Note – The UltraSPARC IV+ processor sets the Sun™ Fireplane Interconnect MID[9:5] to SID_U and MID[4:0] to SID_L. The source of MID[9:0] is the ASI_INTR_ID[9:0] of the logical processor issuing the interrupt.

TABLE 2-2 LP Interrupt ID Register Fields

Bits	Field	Description
[63:10]	*Reserved*	Reserved for future implementation.
[9:0]	Int_ID	The Int_ID is used as the source or target logical processor identities in a Sun Fireplane Interconnect interrupt transaction. In a Sun Fireplane Interconnect interrupt transaction, the source logical processor identity is placed in the Sun Fireplane Interconnect Address bus bits[38:29], and the target logical processor identity is placed in Address bus bits[23:14].

Note – If the Int ID of the two logical processors in an UltraSPARC IV+ processor are not unique in a system, then the behavior of the logical processor when an interrupt specifying that ID is sent or received is undefined.

2.3.3 CESR (Cluster Error Status Register) ID Register

The CESR ID register, summarized in TABLE 2-3, provides support for a tightly clustered system. This register contains an 8-bit field, CESR_ID, which uniquely identifies a logical processor in a tightly clustered system. Certain transactions append this value into the transaction. This allows software at a remote node or within the cluster switch to associate the initiating logical processor with the transaction.

The CESR ID register should only be used with the appropriate cluster interconnect and the corresponding cluster specific software support. The specific value to encode in the CESR ID register is platform−specific. When not used in a cluster architecture, this register should always be programmed to zero.

Name: `ASI_CESR_ID`
ASI 0x63, VA[63:0]==0x40,
Read-Write, Privileged Access

TABLE 2-3 CESR ID Register

Bit	Field	Description
[63:8]	*Reserved*	Reserved for future implementation.
[7:0]	CESR_ID	The CESR_ID field is an 8-bit CESR ID in the bus transaction. For a RBIO/WBIO transaction, CESR[7:0] is encoded appropriately.

Note – The CESR_ID only affects the Sun Fireplane Interconnect RBIO and WBIO transactions. It does not affect other types of Sun Fireplane Interconnect transactions.

2.4 Disabling and Suspending Logical Processors

The CMT programming model provides the ability to disable or temporarily suspend logical processors. This section describes the interface for probing which logical processors are available, enabled, and not suspended. This section also describes the interface for enabling/disabling and suspending/running logical processors. The registers described in this section are shared between logical processors.

2.4.1 LP Available Register (`ASI_CORE_AVAILABLE`)

The LP Available register is a shared register that indicates the number of logical processors implemented in a CMT processor and which logical processor numbers are assigned to them.

Name: ASI_CORE_AVAILABLE
ASI 0x41, VA[63:0]==0x00,
Read-Only, Privileged

The LP Available register is a read-only register with fields in which each bit position corresponds to a logical processor. Bit[0] represents LP 0; bit[1] represents LP 1.

If a bit position in the register is asserted (1), the corresponding logical processor is implemented and is functional in the CMT processor. If a bit position in the register is not asserted (0), the corresponding logical processor is not implemented or was permanently disabled at manufacturing time. An implemented logical processor is a logical processor that can be enabled and used.

In the UltraSPARC IV+ processor, this register is always read as 2'b11.

TABLE 2-4 shows the format of the LP Available register. Each bit represents one logical processor: bit 0 for LP 0, bit 1 for LP 1, and so on. If a logical processor is available (or implemented), then the hardware will set the corresponding bit 1. Otherwise, the hardware sets bit 0. In the UltraSPARC IV+ processor, bit[1] and bit[0] will be set to 1; bits[63:2] are always 0.

TABLE 2-4 LP Available Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

2.4.2 Enabling and Disabling Logical Processors

The CMT programming model allows logical processors to be enabled and disabled. Enabling or disabling a logical processor is a special operation that requires a system reset for updates. Disabled logical processors produce no architectural effects observable by other logical processors, and do not participate in cache coherency. Any transaction issued to a disabled logical processor, such as an interrupt, results in an "unmapped" reply or a time-out.

2.4.2.1 LP Enable Status Register (ASI_CORE_ENABLE_STATUS)

The LP Enable Status register is a shared register that indicates whether each logical processor is currently enabled. The register is a read-only register with a single 64-bit field (assuming a maximum of 64 logical processors per CMT processor) in which each bit corresponds to a possible logical processor. The UltraSPARC IV+ processor has two software-visible logical processors.

Name: ASI_CORE_ENABLE_STATUS
ASI 0x41, VA[63:0]==0x10,
Read-Only, Privileged

Bit[0] and bit[1] represents LP 0 and LP 1, respectively. If a bit in the register is asserted (1), the corresponding logical processor is implemented and enabled. A logical processor not implemented in a CMT device, indicated as "not available" in the LP Available register, cannot be enabled and its corresponding enabled bit in this register will be 0. A logical processor that is suspended is still considered enabled.

TABLE 2-5 shows the format of the LP Enable Status register. Each bit represents one logical processor. A bit set to 1 indicates the corresponding logical processor is enabled; if set to 0, it is disabled. In the UltraSPARC IV+ processor, bit[0] and bit[1] are defined for LP 0 and LP 1, respectively. Bits[63:2] are reserved and read as 0.

TABLE 2-5 LP Enable Status Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

A logical processor disabled by programming the LP Enable register (it requires a power-on-reset or system reset for the updates to the LP Enable register to take effect) is considered not enabled. A logical processor suspended for debug or diagnostics is considered enabled.

State After Reset

The LP Enable Status register changes only at system resets or power-on-reset. The logical processor enable status register value is set by hardware to the value of the LP Enable register at the deassertion of reset.

2.4.2.2 LP Enable Register (ASI_CORE_ENABLE)

The LP Enable register, illustrated in TABLE 2-6, is used by software to enable/disable logical processor(s). The enable/disable action takes effect only when a power-on-reset or a system reset (Soft POR) is deasserted.

Name: ASI_CORE_ENABLE
ASI 0x41, VA[63:0]==0x20,
Privileged, Read-Write

TABLE 2-6 LP Enable Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

The LP Enable register is a 64-bit register. Each bit of the register represents one logical processor, with bit[0] representing LP 0, and bit[1] representing LP 1. A bit set to 1 means a logical processor should be enabled after the next system reset and a bit set to 0 means a logical processor should be disabled after the next reset.

Note – Bits[63:2] are forced to 0 since their corresponding logical processors are not implemented in the UltraSPARC IV+ processor.

If a bit in the LP Available register is 0 (unavailable), hardware forces the corresponding bit in the LP Enable register to 0 and ignores attempts to write “1” to that bit. Since the UltraSPARC IV+ processor always has both logical processors available, this scenario does not exist in the UltraSPARC IV+ processor.

Note – A disabled logical processor will not respond to any transaction issued to it. The sender should encounter an unmapped reply or a timeout error.

In the UltraSPARC IV+ processor, if both bits 1 and 0 are set to 0, then both logical processors will be disabled after a Hard/Soft POR.

State After Reset

The value of the LP Enable register is set to the value of the LP Available register at the assertion of a power-on-reset. The value of the LP Enable register remains unchanged during all other resets, including system resets, or equivalent resets.

2.4.3 Suspending and Running Logical Processors

Suspended logical processors can be set to run later. The suspending and running of logical processors can be performed at arbitrary points in time and, unlike disabling a logical processor, a system reset is not required. There may be an arbitrarily long, but bounded, delay from when a logical processor is directed to suspend until the change takes effect. There is a LP Running Status register that can be used to determine if a logical processor has completed the process of becoming suspended.

A suspended logical processor does not execute instructions and does not initiate any transactions on its own. A suspended logical processor does remain coherent with the system. To remain coherent, a suspended logical processor fully participates in cache coherency and can generate transactions in response to coherency requests from other logical processors on the same or different CMT processor. When a logical processor is set to run, it continues execution with the instruction that was next to be executed when the logical processor was suspended. It is transparent to the software running on a logical processor that it was ever suspended.

An interrupt to a suspended logical processor behaves the same as if the logical processor was too busy to accept the interrupt. For example, if an interrupt buffer is available, the interrupt is ACK’ed and a trap is taken only when the logical processor is set to run. If, however, no interrupt buffer is available, the interrupt is NACK’ed.

The STICK and TICK counters will continue to count while a logical processor is suspended. Suspending logical processors is intended for critical diagnostic and recovery code. The interference with performance monitors using the TICK or STICK counters should not be a general issue. Using the TICK or STICK counter to detect the suspending of a logical processor is not recommended.

2.4.3.1 LP Running Register (`ASI_CORE_RUNNING`)

The LP Running register is a shared register, used by software to suspend and run selected logical processors. When a logical processor is suspended, the logical processor stops executing new instructions and will not initiate transactions except in response to a coherency transaction initiated

by another logical processor. There may be an arbitrarily long, but bounded, delay from when the LP Running register is updated until the corresponding logical processor(s) actually suspends or is set to run.

The LP Running register, is described in TABLE 2-7, is used by software to suspend selected logical processors.

Name: `ASI_CORE_RUNNING_RW`
ASI 0x41, VA[63:0]==0x50,
Privileged, Read-Write

Name: `ASI_CORE_RUNNING_W1S`
ASI 0x41, VA[63:0]==0x60,
Privileged, Write-Only (Write-One to Set)

Name: `ASI_CORE_RUNNING_W1C`
ASI 0x41, VA[63:0]==0x68,
Privileged, Write-Only (Write-One to Clear)

TABLE 2-7 LP Running Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

The LP Running register is a 64-bit register. Each bit of the register represents one logical processor, with bit[0] representing LP 0, and bit[1] representing LP 1.

Once a logical processor is set to suspend, the logical processor will stop fetching instructions, complete the instructions in the logical processor and the instruction buffers, and then become idle. When the logical processor is set to run, it continues execution from the point it was suspended.

A logical processor is allowed to suspend itself. A logical processor that suspends itself should follow the ASI write by a `FLUSH` instruction. This satisfies the ASI writing rules and guarantees that the logical processor will be suspended and no instructions will be executed following the `FLUSH` if the logical processor is successfully suspended. The `FLUSH` instruction itself may be asserted before or after the logical processor is suspended.

Note – The UltraSPARC IV+ processor will not allow software to suspend both logical processors. An update to the LP Running register that would cause both logical processors to become suspended results only in the suspension of the logical processor not making the request, with the logical processor making the update automatically set to run instead by hardware.

To minimize the need for synchronization between logical processors in writing to this register, separate virtual addresses are provided to set and reset the bits of this register. This, combined with the reset setting, means that the need for special interlocking on the register is not necessary.

When writing to this register, there is a choice between writing an exact value and modifying individual bits. When a logical processor suspends itself, a write to the clear bit VA should be used. When a logical processor wants to become the only logical processor active, it is more appropriate to write the desired value directly to the direct access VA, since this eliminates the need for the set and clear operations required when writing a specific value to the register.

State After Reset

On assertion of power on reset or system reset (Soft POR), the LP Running register will be initialized such that all the logical processors are suspended, except the logical processor with the lowest number, which instead is marked “enabled” in the LP Enable Status register. This provides an integrated “boot master” logical processor for systems without a System Controller (SC), reducing bootbus contention. The logical processors suspended by the reset should be set to run by the master logical processor at the proper time in the booting process.

2.4.3.2 LP Running Status Register (ASI_CORE_RUNNING_STATUS)

Since there is a delay from when a logical processor is directed to suspend until it actually becomes suspended, the LP Running Status register is provided to indicate when a logical processor actually becomes suspended. The LP Running Status register is a shared, read-only register where each bit indicates if the corresponding logical processor is active.

In the UltraSPARC IV+ processor, a logical processor is considered suspended successfully if the following conditions are satisfied:

1. No instruction in the instruction queue and logical processor.
2. No pending I-cache fetch, D-cache load, D-cache store, P-cache load, and W-cache eviction requests.
3. No requests in the Store Queue.

Note – A D-cache load is considered finished if the D-cache has received the data.

Name: ASI_CORE_RUNNING_STATUS
ASI 0x41, VA[63:0]==0x58,
Privileged, Read-Only

TABLE 2-8 LP Running Status Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

As shown in TABLE 2-8, the LP Running Status register is a 64-bit register. Each bit of the register represents a logical processor, with bit[0] representing LP 0 and bit[1] representing LP 1.

For any bit set to 1 in the LP Running register, the corresponding bit needs to be 1 in the LP Running Status register.

Note – For one suspend command to a logical processor, the corresponding bit of the specified logical processor in the LP Running Status register will have only one transition from 1 to 0.

The LP Enable, LP Running, and LP Running Status registers are mainly used to support debug and diagnostics. The LP Running register is also used to support booting.

State After Reset

The value of the LP Running Status register is the same as the value of the LP Running register at the end of a system reset.

2.5 Reset Handling

Each reset is handled differently in a CMT processor. Some resets apply to all the logical processors, some apply to an individual logical processor, and some apply to an arbitrary subset. The following sections address how each type of reset is handled with respect to having multiple logical processors integrated into a package. In general, the reset nomenclature used is consistent with UltraSPARC IV+ processors. Future processors may have a different classification of resets; in which case, those processors should extend this model appropriately.

2.5.1 Private Resets (SIR and WDR Resets)

The only resets that are limited to a single logical processor are the private resets internally generated by a logical processor. An UltraSPARC IV+ processor has a number of resets of this class. These types of resets are generated by an individual logical processor and are not propagated to the other logical processors on a CMT processor.

2.5.2 Full-CMT Resets (System Reset)

There is a class of resets that are generated by an external agent and apply to all the logical processors in a CMT processor. These include any resets associated with fundamental reconfigurations of the CMT processor. Current SPARC processors have a single system reset, of which power-on-reset is a special case. System reset is required for certain reconfigurations of the processor. Future processors may have multiple resets that replace the single system reset of current processors.

The power-on and system resets (or their equivalents in future processors) are sent to all logical processors in a CMT processor. All logical processors, except the lowest enabled logical processor, are set, by default, to the suspended state at the beginning of a system reset. The one logical processor that is set to run becomes the default master logical processor, which should arbitrate for the bootbus, if necessary (i.e., if multiple CMT processors share the same bootbus). The master logical processor should enable (set to run) the other logical processors at the proper time in the booting process.

2.5.3 Partial CMT Resets (XIR Reset)

There is a class of resets that are generated by an external agent and apply to an arbitrary improper subset of the logical processors within a CMT processor (any number of the LPs included, from zero to all). The UltraSPARC IV+ processors have, in addition to a single global system reset, a single eXternally Initiated Reset (XIR) signal. This is a reset intended to reset a specific processor in a system, primarily for diagnostic and recovery purposes. Future processors may have multiple resets that replace the single XIR reset of current processors.

For this class of resets there must be a mechanism to specify which subset of logical processors should be reset. There are two possible ways to specify the subset. The first way to specify the subset is to have a steering register that is set up ahead of time to specify the subset of logical processors. For systems using an XIR reset, the XIR Steering register described in *XIR Steering Register (ASI_XIR_STEERING)* on page 21 should be used.

The second way to specify the subset is to specify the subset concurrently with delivering the reset across the interface used for communicating the reset. This method would require that the interface used for communicating resets supports sending packets of information along with the resets.

2.5.3.1 XIR Steering Register (ASI_XIR_STEERING)

The XIR reset can be steered only to specific logical processors under the control of the XIR Steering register described in TABLE 2-9.

Name: ASI_XIR_STEERING
ASI 0x41, VA[63:0]==0x30,
Privileged, Read-Write

TABLE 2-9 XIR Steering Register (Shared)

Bit	Field	Description
[63:2]	Mandatory value	Should be 0.
[1]	LP 1	This bit represents LP 1.
[0]	LP 0	This bit represents LP 0.

The XIR Steering register is a 64-bit register out of which only bits[1:0] are used in the UltraSPARC IV+ processor. Each bit of the register represents one logical processor, with bit[0] representing LP 0, and bit[1] representing LP 1. An XIR is blocked to a logical processor if the corresponding bit is 0. Hardware will force a 0 for unimplemented logical processors.

State After Reset

At the end of a system reset (or equivalent reset), the value of the XIR reset is equal to the value of the LP Enable Status register (which in turn is equal to the value of the LP Enable register).

2.6 Private and Shared Registers Summary

The UltraSPARC IV+ processor implements the following private and shared registers.

2.6.1 Implementation Registers

TABLE 2-10 and TABLE 2-11 summarize the private and shared registers, respectively.

TABLE 2-10 The UltraSPARC IV+ Processor Private Registers

ASI Value	ASI Name	Access	VA	Description
0x63	ASI_INTR_ID	RW	0x00	Interrupt ID register
0x63	ASI_CORE_ID	R	0x10	LP ID register
0x63	ASI_CESR_ID	RW	0x40	CESR ID register

TABLE 2-11 The UltraSPARC IV+ Processor Shared Registers

ASI Value	ASI Name	Access	VA	Description
0x41	ASI_CORE_AVAILABLE	R	0x00	LP Available register
0x41	ASI_CORE_ENABLE_STATUS	R	0x10	LP Enable Status register
0x41	ASI_CORE_ENABLE	RW	0x20	LP Enable register, Read-Write
0x41	ASI_XIR_STEERING	RW	0x30	XIR Steering register, Read-Write
0x41	ASI_CORE_RUNNING_RW	RW	0x50	LP Running register, Read-Write
0x41	ASI_CORE_RUNNING_W1S	W	0x60	LP Running register, Write One Set
0x41	ASI_CORE_RUNNING_W1C	W	0x68	LP Running register, Write One Clear
0x41	ASI_CORE_RUNNING_STATUS	R	0x58	LP Running Status register
0x41	ASI_CMT_ERROR_STEERING	RW	0x40	Error Steering register, Read-Write

---

Note – ASI accesses to the registers must use `LDXA/STXA/LDDFA/STDFA` instructions. Using another type of load or store instruction will cause a *data_access_exception* trap (with SFSR.FT = 8, illegal ASI value, VA, RW, or size). Attempt to access these registers while in non-privileged mode will cause a *privileged_action* trap (with SFSR.FT = 1, privilege violation). A non-aligned access will cause a *mem_address_not_aligned* trap. If the instruction is `LDDFA/STDFA` and if the address is aligned to a 32-bit boundary but not to a 64-bit boundary, then the trap type will be *LDDF/STDF_mem_address_not_aligned*.

3

Caches, Cache Coherency and Diagnostics

This chapter describes the caches, cache coherency and the diagnostics in the following sections:

Chapter Topics

3.1 Cache Organization

This section describes the different types of cache organizations found in the UltraSPARC IV+ processor: virtually-indexed, physically-tagged (VIPT), physically-indexed, physically-tagged (PIPT), and virtually-indexed, virtually-tagged (VIVT) caches.

3.1.1 Cache Overview

The UltraSPARC IV+ processor supports three levels of cache. TABLE 3-1 summarizes the cache organization of the UltraSPARC IV+ processor level 1(L1) (I-cache, D-cache, P-cache, W-cache), level 2 (L2), and level 3 (L3) caches. The cache organization is discussed in details in subsequent sections.

TABLE 3-1 The UltraSPARC IV+ Processor Cache Organization

Cache	Size	Line Size	Subblock (Number of subblocks)	Set Associativity	Replacement Policy	Organization	Data Protection	
							ECC	Parity
Instruction cache	64 KB	64-byte	Yes (two 32-byte subblocks)	4-way	Pseudo-Random	VIPT	None	Tag Data
Data cache	64 KB	32-byte	No	4-way	Pseudo-Random	VIPT	None	Tag Data
Prefetch cache	2 KB	64-byte	Yes (two 32-byte subblocks)	4-way	Sequential	VIVT	None	Data
Write-cache	2 KB	64-byte	No	Fully associative	FIFO	PIPT	None	None
L2-cache	2 MB	64-byte	No	4-way	Pseudo-LRU	PIPT	Tag Data	None
L3-cache	32 MB	64-byte	No	4-way	Pseudo-LRU	PIPT	Tag Data	SRAM (address)

3.1.2 Virtually-Indexed, Physically-Tagged (VIPT) Caches

The Instruction cache (I-cache) and the Data cache (D-cache) are virtually-indexed, physically-tagged caches. The I-cache and D-cache have no references to context information. Virtual addresses index into the cache tag and data arrays while accessing the I-MMU/D-MMU. The resulting tag is compared against the translated physical address to determine a cache hit.

Note – A side effect inherent in a virtual indexed cache is address aliasing. See *Address Aliasing Flushing* on page 28.

3.1.2.1 Instruction Cache (I-cache)

The instruction cache is a 64 KB, 4-way set-associative cache with a 64-byte line size. Each I-cache line is divided into two 32-byte subblocks with separate valid bits. The I-cache is a write-invalidate cache. It uses a pseudo-random replacement policy. I-cache tag and data arrays are parity protected.

Instruction fetches bypass the I-cache in the following cases:

- The I-cache enable (IC) bit in the Data Cache Unit Control Register is not set (DCUCR.IC = 0)
- The I-MMU is disabled (DCUCR.IM = 0)and the CV bit in the Data Cache Unit Control Register is not set (DCUCR.CV = 0)
- The processor is in RED_state
- The fetch is mapped by the I-MMU as being non-virtual-cacheable

The I-cache snoops stores from other processors or DMA transfers, as well as stores in the same processor and block store commits.

The FLUSH instruction is not required to maintain coherency. Stores and block store commits invalidate the I-cache but do not flush instructions that have already been prefetched into the logical processor. A FLUSH, DONE, or RETRY instruction can be used to flush the logical processor.

Note – If a program changes I-cache mode to I-cache-ON from I-cache-OFF, then the next instruction fetch always causes an I-cache miss even if it is supposed to hit. This rule applies even when the DONE instruction turns on the I-cache by changing its status from RED_state to normal mode.

3.1.2.2 Data Cache (D-cache)

The data cache is a 64 KB, 4-way set-associative cache with 32-byte line size. It is a write-through, non write-allocate cache. The D-cache uses a pseudo-random replacement policy. D-cache tag and data arrays are parity protected.

Data accesses bypass the D-cache if the D-cache enable (DC) bit in the Data Cache Unit Control Register is not set (DCUCR.DC = 0). If the D-MMU is disabled (DCUCR.DM = 0), then cacheability in the D-cache is determined by the CP and CV bits. If the access is mapped by the D-MMU as non-virtual-cacheable, then load misses will not allocate in the D-cache. For more information on the DM, CP, or CV bits, see *Data Cache Unit Control Register (DCUCR)* on page 273.

A non-virtual-cacheable access may access data in the D-cache from an earlier cacheable access to the same physical block unless the D-cache is disabled.

Note – Software must flush the D-cache when changing a physical page from cacheable to non-cacheable (see *Cache Flushing* on page 28).

3.1.3 Physically-Indexed, Physically-Tagged Caches (PIPT)

The Write cache, Level-2 (L2) cache, and Level-3 (L3) cache are physically-indexed, physically-tagged (PIPT) caches. These caches have no references to virtual address and context information. The operating system needs no knowledge of such caches after initialization, except for stable storage management and error handling.

3.1.3.1 Write Cache (W-cache)

The Write cache is a 2 KB, fully-associative cache with 64-byte line size. The W-cache uses a FIFO (First-In-First-Out) replacement policy. The UltraSPARC IV+ processor's W-cache has no parity protection.

The W-cache is included in the L2-cache, however, write data is not immediately updated in the L2-cache. The L2-cache line gets updated either when the line is evicted from the W-cache, or when a primary cache of either logical processor sends read request to the L2-cache, in which case, the L2-cache probes the W-cache and sends the data from the W-cache to the primary cache and subsequently updates the L2-cache line. Since the W-cache is inclusive in the L2-cache, flushing the L2-cache ensures that the W-cache has also been flushed.

3.1.3.2 L2-cache and L3-cache

The L2-cache is a unified 2MB, 4-way set-associative cache with 64-byte line size. It is a writeback, write-allocate cache and uses a pseudo-LRU replacement policy. The L2-cache includes the contents of the I-cache, the D-cache, and the W-cache in both logical processors. Thus, invalidating a line in the L2-cache will also invalidate the corresponding line(s) in the I-cache, D-cache, or W-cache. The inclusion property is not maintained between the L2-cache and the P-cache.

The L3-cache is a unified 32MB, 4-way set associative cache with 64-byte line size. It is a writeback, write-allocate cache and uses a pseudo-LRU replacement policy. The L3-cache is a dirty victim cache. When a line comes into the processor, it is loaded in the L2-cache and the appropriate L1-cache(s). When a line (both clean and dirty) is evicted from the L2-cache, it is written back to the L3-cache. The L2-cache and L3-caches are mutually exclusive, i.e., a given cache line can exist either in the L2-cache or the L3-cache, but not in both.

The tag and the data arrays of both the L2-cache and the L3-cache are ECC protected.

Instruction fetches bypass the L2-cache and L3-cache in the following cases:

- The I-MMU is disabled (DCUCR.IM = 0) *and* the CP bit in the Data Cache Unit Control Register is not set (DCUCR.CP = 0)
- The processor is in RED_state
- The access is mapped by the I-MMU as non-physical-cacheable

Data accesses bypass the L2-cache and L3-cache if the D-MMU is disabled (DCUCR.DM = 0) or if the access is mapped by the D-MMU as non-physical-cacheable (unless `ASI_PHYS_USE_EC` is used).

The system must provide a non-cacheable, scratch memory region for booting code use until the MMUs are enabled.

Block loads and block stores, which load or store a 64-byte block of data from memory to the Floating Point Register file, do not allocate into the L2-cache or L3-cache. Prefetch Read Once instructions (prefetch fcn = 1, 21), which load a 64-byte block of data into the P-cache, do not allocate into the L2-cache and L3-cache.

3.1.4 Virtually-Indexed, Virtually-Tagged (VIVT) Caches

The prefetch cache (P-cache) is virtually-indexed, virtually-tagged for cache reads. However, it is physically-indexed, physically-tagged (PIPT) for snooping purposes.

3.1.4.1 Prefetch cache (P-cache)

The prefetch cache is a 2 KB, 4-way set-associative cache with 64-byte line size. Each cache line is divided into two 32-byte subblocks with separate valid bits. The P-cache is a write-invalidate cache and uses a sequential replacement policy (i.e., the ways are replaced in sequential order). The P-cache data array is parity protected. The P-cache needs to be flushed only for error handling.

The P-cache can be used to hide memory latency and increase memory-level parallelism by prefetching data into the P-cache. Prefetches can be generated by an autonomous hardware prefetch engine or by software prefetch instructions.

Hardware Prefetching

The hardware prefetch engine in the UltraSPARC IV+ processor automatically starts prefetching the next cache line from the L2-cache after a floating-point (FP) load instruction hits the P-cache.

When a floating-point load misses both the D-cache and the P-cache, either 32 bytes (if, from memory) or 64 bytes (if from L2 or L3) of data is installed in the P-cache. Each P-cache line contains a fetched_mode bit that indicates how the P-cache line was installed - by software prefetch instruction or hardware prefetch mechanism. When a load hits the P-cache, and the fetched_mode bit of the P-cache line indicates that it was not brought in by a software prefetch, the hardware prefetch engine attempts to prefetch the next 64-byte cache line from the L2-cache. Depending on the prefetch stride, the next line to be prefetched can be at a 64, 128, or 192-byte offset from the P-cache line that initiated the prefetch. Thus, when a floating-point load at address A hits the P-cache, a hardware prefetch to address A+64, A+128, or A+192 will be initiated.

A hardware prefetch request will be dropped in the following cases:

- the data already exists in the P-cache.
- the prefetch queue is full.
- the hardware prefetch address is the same as one of the outstanding prefetches.
- the prefetch address is not in the same 8 KB boundary as the line that initiated the prefetch.
- the request misses the L2-cache.

To enable hardware prefetching, the Prefetch Cache Enable (PE) bit (bit 45) and the Hardware Prefetch Enable (HPE) bit (bit 44) in the Data Cache Unit Control Register (DCUCR) must be set. The Programmable P-cache Prefetch Stride (PPS) bits (bit [51:50]) of DCUCR determine the prefetch stride when hardware prefetch is enabled. See *Data Cache Unit Control Register (DCUCR)* on page 273 for details.

Software Prefetching

The UltraSPARC IV+ processor supports software prefetching through the `PREFETCH(A)` instructions. Software prefetching can prefetch floating point data into the P-cache, or L2-cache, or both depending on the type of prefetch instruction used. Software prefetching can also hide memory latency for integer instructions by bringing in integer data into the L2-cache (the data brought into the P-cache is useless in this case, since integer instructions cannot use the AX pipe).

Note – To enable the use of software prefetching, the Software Prefetch Enable (SPE) bit (bit 43) as well as the PE bit in DCUCR must be set. If the P-cache is disabled, or the P-cache is enabled without the SPE bit being set (i.e., PE = 1, SPE = 0), all software prefetch instructions will be treated as NOPs.

The `PREFETCH` instruction with fcn = 16 can be used to invalidate or flush a P-cache entry. This fcn can be used to invalidate special non-cacheable data after the data is loaded into registers from the P-cache.

Note – `PREFETCH` with fcn = 16 cannot be used to prefetch non-cacheable data. It is used only to invalidate the P-cache line. In particular, it is used to invalidate the P-cache data that was non-cacheable and was prefetched through other software prefetch instructions.

3.2 Cache Flushing

Data in the I-cache, D-cache, P-cache, W-cache, L2-cache, and L3-cache can be flushed by invalidation of the entry in the cache. Modified data in the W-cache, L2-cache, and L3-cache must be written back to memory when flushed.

Cache flushing is required in the following cases:

- A D-cache flush is needed when a physical page is changed from (virtually) cacheable to (virtually) non-cacheable or when an illegal address alias is created. Flushing is done with a displacement flush (see *Displacement Flushing* on page 29) or by use of ASI accesses.
- An L2-cache flush is needed for stable storage. Flushing is done with either a displacement flush or a store with `ASI_BLK_COMMIT`. Flushing the L2-cache will flush the corresponding blocks from W-cache also. See *Committing Block Store Flushing* on page 29.
- An L3-cache flush is needed for stable storage. Flushing is done with either a displacement flush or a store with `ASI_BLK_COMMIT`.
- L2, L3, Data, Prefetch, and Instruction cache flushes may be required when an ECC error occurs on a read from the Sun Fireplane Interconnect or the L3-cache. *AFSR Register and AFSR_EXT Register* on page 180 describes the case when a flush on an error is required. When an ECC error occurs, invalid data may be written into one of the caches and the cache lines must be flushed to prevent further corruption of data.

3.2.1 Address Aliasing Flushing

A side effect inherent in a virtually-indexed cache is illegal address aliasing. Aliasing occurs when multiple virtual addresses map to the same physical address.

Note – Since the I-cache and D-cache are indexed with the virtual address bits and the caches are larger than the minimum page size, it is possible for the different aliased virtual addresses to end up in different cache blocks. Such aliases are illegal because updates to one cache block will not be reflected in aliased cache blocks.

For example, consider two virtual addresses A and B, with the same VA[12:0] bits and different VA[13] bit, map to the same physical address (PA[12:0] = VA[12:0]). Now, if both A and B are loaded into the D-cache, they will be mapped to different D-cache blocks, since the D-cache index (VA[13:5]) for A and B are different due to bit[13]. Such address aliasing is illegal, since stores to one aliased block (say A) will not update the other aliased block (B) as they are mapped to different blocks.

Normally, software avoids illegal aliasing by forcing aliases to have the same address bits, known as *virtual color*, up to an alias boundary. The minimum alias boundary is 16 KB. This size may increase in future designs.

When the alias boundary is violated, software must flush the I-cache or the D-cache if the page was virtual cacheable. In this case, only one mapping of the physical page can be allowed in the I-MMU or D-MMU at a time.

Alternatively, software can turn off virtual caching of illegally aliased pages. Doing so allows multiple mapping of the alias to be in the I-MMU or D-MMU and avoids flushing of the I-cache or the D-cache each time a different mapping is referenced.

Note – A change in virtual color when allocating a free page does not require a D-cache flush because the D-cache is write-through.

3.2.2 Committing Block Store Flushing

Stable storage must be implemented by software cache flush. Examples of stable storage are battery-backed memory and a transaction log. Data that are present and modified in the L2-cache, L3-cache, or W-cache must be written back to the stable storage.

Two ASIs (`ASI_BLK_COMMIT_PRIMARY` and `ASI_BLK_COMMIT_SECONDARY`) perform these writebacks efficiently when software can ensure exclusive write access to the block being flushed. These ASIs write back the data to memory from the Floating Point Registers and invalidate the entry in the cache. The data in the Floating Point Registers must first be loaded by a block load instruction. A `MEMBAR #Sync` instruction can be used to ensure that the flush is complete.

3.2.3 Displacement Flushing

Cache flushing can also be accomplished by a displacement flush. This procedure reads a range of addresses that map to the corresponding cache line being flushed, forcing out modified entries in the local cache.

Note – The range of read-only addresses must be mapped in the MMU before starting a displacement flush; otherwise, the TLB miss handler may put new data into the caches.

Diagnostic ASI accesses to the D-cache, L2-cache or L3-cache can be used to invalidate a line, but they are not an alternative to displacement flushing. The invalidated line will not be written back to the next level of cache when these ASI accesses are used. Specifically, data (clean or modified) in the L2-cache will not be written back to the L3-cache and the modified data in the L3-cache will not be written back to memory.

3.2.4 Prefetch Cache Flushing

A context switch flushes the P-cache. When a write to a context register takes place, all entries in the P-cache are invalidated. The entries in the prefetch queue also get invalidated such that data for outstanding prefetch requests will not get installed in the P-cache once the data is returned.

3.3 Coherence Tables

The set of tables in this section describes the cache coherence protocol that governs the behavior of the processor on the Sun Fireplane Interconnect.

3.3.1 Processor State Transition and the Generated Transaction

Tables in this section summarize the following:

- Hit/Miss, State Change, and Transaction Generated for Processor Action
- Combined Tag / MTag States

TABLE 3-2 defines the terms used in the subsequent tables.

Derivation of DTags, CTags, and MTags from Combined Tags is shown in TABLE 3-5.

TABLE 3-2 Definitions of the Terms

Term	Meaning
MOESI	A cache-coherence protocol. M = modified, dirty data with no outstanding shared copy; O = owned, dirty data with outstanding shared copy(s); E = exclusive, clean data with no outstanding shared copy; S = shared, clean data with outstanding shared copy(s); I = invalid, invalid data.
MOOsESI	An SSM mode cache-coherence protocol. M = modified, dirty data with no outstanding shared copy; O = owned, dirty data with (potentially) outstanding shared copy(s) in the local SSM domain; Os = owned, dirty data with (potentially) outstanding shared copy(s) in other SSM domains; E = exclusive, clean data with no outstanding shared copy; S = shared, clean data with outstanding shared copy(s); I = invalid, invalid data.
RTO	Read To Own Transaction
RTS	Read To Share Transaction
RTOR	Read To Own Remote
RTSM	Read To Share MTag

TABLE 3-3 Hit/Miss, State Change, and Transaction Generated for Processor Action *(1 of 2)*

Combined State	MODE	Processor action					
		Load	**Store/ Swap**	**Block Load**	**Block Store**	**Block Store Commit**	**Write Prefetch**
I	~SSM	miss: RTS	miss: RTO	miss: RS	miss: WS	miss: WS	miss: RTO
	SSM & LPA	miss: RTS	miss: RTO	miss: RS	miss: R_WS	miss: R_WS	miss: RTO
	SSM & LPA & retry	MTag miss: R_RTS	MTag miss: R_RTO	MTag miss: R_RS	invalid	invalid	MTag miss: R_RTO
	SSM & ~LPA	miss: R_RTS	miss: R_RTO	miss: R_RS	miss: R_WS	miss: R_WS	miss: R_RTO
E	~SSM	hit	hit: E->M	hit	hit: E->M	miss: WS	hit
	SSM & LPA	hit	hit: E->M	hit	hit: E->M	miss: R_WS	hit
	SSM & LPA & retry	invalid					
	SSM & ~LPA	hit	hit: E->M	hit	hit: E->M	miss: R_WS	hit
S	~SSM	hit	miss: RTO	hit	miss: WS	miss: WS	hit
	SSM & LPA	hit	MTag miss: RTO	hit	miss: R_WS	miss: R_WS	hit
	SSM & LPA & retry	invalid	MTag miss: R_RTO	invalid	invalid	invalid	invalid
	SSM & ~LPA	hit	MTag miss: R_RTO	hit	miss: R_WS	miss: R_WS	hit

TABLE 3-3 Hit/Miss, State Change, and Transaction Generated for Processor Action *(2 of 2)*

Combined State	MODE	Processor action					
		Load	Store/ Swap	Block Load	Block Store	Block Store Commit	Write Prefetch
O	~SSM	hit	miss: RTO	hit	miss: WS	miss: WS	hit
	SSM & LPA	hit	MTag miss: RTO	hit	miss: R_WS	miss: R_WS	hit
	SSM & LPA & retry	invalid	MTag miss: R_RTO	invalid	invalid	invalid	invalid
	SSM & ~LPA	hit	MTag miss: R_RTO	hit	miss: R_WS	miss: R_WS	hit
Os (Legal only in SSM mode)	~SSM	invalid					
	SSM & LPA	hit	MTag miss: R_RTO	hit	miss: R_WS	miss: R_WS	hit
	SSM & LPA & retry	invalid	MTag miss: R_RTO	invalid	invalid	invalid	invalid
	SSM & ~LPA	hit	MTag miss: R_RTO	hit	miss: R_WS	miss: R_WS	hit
M	~SSM	hit	hit	hit	hit	miss: WS	hit
	SSM & LPA	hit	hit	hit	hit	miss: R_WS	hit
	SSM & LPA & retry	invalid					
	SSM & ~LPA	hit	hit	hit	hit	miss: R_WS	hit

TABLE 3-4 Combined Tag/MTag States

MTag State: CTag State	gI	gS	gM
cM	I	Os	M
cO	I	Os	O
cE	I	S	E
cS	I	S	S
cI	I	I	I

TABLE 3-5 Deriving DTags, CTags, and MTags from Combined Tags

Combined Tags (CCTags)	DTag	CTag	MTag
I	dI	cI	gI
E	dS	cE	gM
S	dS	cS	gS
O	dO	cO	gM
Os	dO	cO	gS
M	dO	cM	gM

3.3.2 Snoop Output and Input

TABLE 3-6 summarizes snoop output and DTag transition; TABLE 3-7 summarizes snoop input and CIQ operation queueing.

Note – The symbol " ~ " implies "not."

TABLE 3-6 Snoop Output and DTag Transition ***(1 of 3)***

Snooped Request	DTag State	Shared Output	Owned Output	Error Output	Next DTag State	Action for Snoop Pipeline
own RTS (for data)	dI	0	0	0	dT	own RTS wait data
	dS	1	0	1	dS	Error
	dO	1	0	1	dO	Error
	dT	1	0	1	dT	Error
own RTS (for instructions)	dI	0	0	0	dS	own RTS inst wait data
	dS	1	0	1	dS	Error
	dO	1	0	1	dO	Error
	dT	1	0	1	dT	Error
foreign RTS	dI	0	0		dI	none
	dS	1	0		dS	none
	dO	1	1		dO	foreign RTS copyback
	dT	1	0		dS	none
own RTO	dI	0	0		dO	own RTO wait data
	dS & ~SSM	1	1		dO	own RTO no data
	dS & SSM	0	0		dO	own RTO wait data
	dO	1	1		dO	own RTO no data
	dT	1	1	1	dO	Error
foreign RTO	dI	0	0		dI	none
	dS	0	0		dI	foreign RTO invalidate
	dO	0	1		dI	foreign RTO copyback-invalidate
	dT	0	0		dI	foreign RTO invalidate

TABLE 3-6 Snoop Output and DTag Transition *(2 of 3)*

Snooped Request	DTag State	Shared Output	Owned Output	Error Output	Next DTag State	Action for Snoop Pipeline
own RS	dI	0	0		dI	own RS wait data
	dS	0	0	1	dS	Error
	dO	0	0	1	dO	Error
	dT	0	0	1	dT	Error
foreign RS	dI	0	0		dI	none
	dS	0	0		dS	none
	dO	0	1		dO	foreign RS copyback-discard
	dT	0	0		dT	none
own WB	dI	0	1		dI	own WB (cancel)
	dS	0	1		dI	own WB (cancel)
	dO	0	0		dI	own WB
	dT	0	1	1	dI	Error
foreign WB	dI	0	0	0	dI	none
	dS	0	0	0	dS	none
	dO	0	0	0	dO	none
	dT	0	0	0	dT	none
foreign RTSM	dI	0	0		dI	none
	dS	0	0		dS	foreign RTSM
	dO	0	1		dS	fRTSM copyback
	dT	0	0		dS	foreign RTSM
foreign RTSU	dI	0	0		dI	none
	dS	0	0		dS	none
	dO	0	1		dS	fRTSU copyback
	dT	0	0		dS	none
foreign RTOU	dI	0	0		dI	none
	dS	0	0		dI	foreign RTOU invalidate
	dO	0	1		dI	fRTOU copyback invalidate
	dT	0	0		dI	foreign RTSU invalidate
foreign UGM	dI	0	0		dI	none
	dS	0	0		dS	none
	dO	0	1		dO	foreign RS copyback-discard
	dT	0	0		dT	none

TABLE 3-6 Snoop Output and DTag Transition *(3 of 3)*

Snooped Request	DTag State	Shared Output	Owned Output	Error Output	Next DTag State	Action for Snoop Pipeline
own RTSR (issued by SSM device)	dI	0	0		dO	own RTSR wait data
	dS	0	0	0	dO	own RTSR wait data
	dO	0	0	1	dO	own RTSR wait data, Error
	dT	0	0	1	dT	own RTSR wait data, Error
foreign RTSR	dI	0	0		dI	none
	dS	1	0		dS	none
	dO	1	1		dS	foreign RTSR
	dT	1	0		dS	none
own RTOR (issued by SSM device)	dI	0	0		dO	own RTOR wait data
	dS	0	0		dO	own RTOR wait data
	dO	0	0		dO	own RTOR wait data
	dT	0	0	1	dO	Error
foreign RTOR	dI	0	0		dI	none
	dS	0	0		dI	foreign RTOR invalidate
	dO	0	0		dI	foreign RTOR invalidate
	dT	0	0		dI	foreign RTOR invalidate
own RSR	dI	0	0		dI	own RSR wait data
	dS	0	0	1	dS	Error
	dO	0	0	1	dO	Error
	dT	0	0	1	dT	Error
foreign RSR	dI	0	0		dI	none
	dS	0	0		dS	none
	dO	0	0		dO	none
	dT	0	0		dT	none
own WS	dI	0	0		dI	own WS
	dS	0	0		dI	own invalidate WS
	dO	0	0		dI	own invalidate WS
	dT	0	0		dI	own invalidate WS
foreign WS	dI	0	0		dI	none
	dS	0	0		dI	invalidate
	dO	0	0		dI	invalidate

NOTE: A blank entry in the “Error Output” column indicates that there is no corresponding error output.

TABLE 3-7 summarizes the Snoop input and CTQ operation queued.

TABLE 3-7 Snoop Input and CIQ Operation Queued

Action from Snoop Pipeline	Shared Input	Owned Input	Error (out)	Operation Queued in CIQ
own RTS wait data	1	x		RTS Shared
	0	0		RTS ~Shared
	0	1	1	RTS Shared, Error
own RTS inst wait data	x	x		RTS Shared
foreign RTS copyback	x	1		copyback
	x	0	1	copyback, Error
own RTO no data	1	x		RTO nodata
	0	x	1	RTO nodata, error
own RTO wait data	1	x	1	RTO data, error
	0	x		RTO data
foreign RTO invalidate	x	x		invalidate
foreign RTO copyback-invalidate	x	0	1	copyback-invalidate, Error
	0	1		copyback-invalidate
	1	1		invalidate
own RS wait data	x	x		RS data
foreign RS copyback-discard	x	0	1	Error
	x	1		copyback-discard
foreign RTSM copyback	x	0	1	RTSM copyback, Error
	x	1		RTSM copyback
foreign RTSU copyback	x	0	1	RTSU copyback, Error
	x	1		RTSU copyback
foreign RTOU invalidate	x	x	1	invalidate
foreign RTOU copyback-invalidate	x	0	1	copyback-invalidate, Error
	0	1		copyback-invalidate
own RTSR wait data	1	x		RTSR shared
	0	x		RTSR~shared
own RTOR wait data	x	x		RTOR data
foreign RTOR invalidate	x	x		invalidate
own RSR	x	x		RS data
own WS	x	x		own WS
own WB	x	x		own WB
own invalidate WS	x	x		own invalidate WS
invalidate	x	x		invalidate

An “X” in the table represents don’t-cares. A blank entry in the “Error (out)” column indicates that there is no corresponding error output.

3.3.3 Transaction Handling

TABLE 3-8 in this section summarizes handling of the following:

- Transactions at the head of CIQ
- No snoop transactions
- Transactions internal to the UltraSPARC IV+ processor

TABLE 3-8 Transaction Handling at Head of CIQ *(1 of 2)*

Operation at Head of CIQ	CCTag	MTag (in/out)	Error	Retry	Next CCTag
RTS Shared	I	gM (in)			S
		gS (in)			S
		gI (in)		1	I
	M,O, E,S,Os	x	1		
RTS ~Shared	I	gM (in)			E
		gS (in)			S
		gI (in)		1	I
	M,O, E,S,Os	x (in)	1		
RTSR Shared	I	gM (in)			O
		gS (in)			Os
		gI (in)	1	1	I
	M,O, E,S,Os	x	1		
RTSR ~Shared	I	gM (in)			M
		gS (in)			Os
		gI (in)	1	1	I
	M,O, E,S,Os	x(in)	1		
RTO nodata	I,M,E, Os	none	1		
	O, S	none			M
RTO data & ~SSM	M,E,O,Os,S	x (in)	1		
	I	gM (in)			M
		gS (in)		1	Os
		gI (in)		1	I
RTO data & SSM	M, E, Os, O	x(in)	1		
	I, S	gM(in)			M
		gS(in)		1	Os
		gi(in)		1	I
RTOR data	M,E	x (in)	1		
	O	gM (in)	1		O
	S,Os, I	gM (in)			M
	S,O,Os, I	gS (in)	1	1	Os
		gI (in)	1	1	I

TABLE 3-8 Transaction Handling at Head of CIQ *(2 of 2)*

Operation at Head of CIQ	CCTag	MTag (in/out)	Error	Retry	Next CCTag
foreign RTSR	I	none			I
	M, O	none	1		no change
	E,Os,S	none			S
foreign RTSM	I	x (in)	1		
	M,O,Os	none	1		S
	E,S	none			S
RTSM copyback	M,O	gM (out)			S
	Os	gS (out)			S
	E,S,I		1		
RTSU copyback	M,O	gM (out)			S
	Os	gS (out)			S
	E,S,I		1		
copyback	M,O	gM (out)			O
	Os	gS (out)			Os
	I	gI (out)			I
	E,S	gM (out)	1		S
invalidate	x				I
copyback-invalidate	M,O	gM (out)			I
	Os	gS (out)			I
	I	gI (out)			I
	E,S	gM (out)	1		I
copyback-discard	M,O	gM (out)			no change
	Os	gS (out)			Os
	I	gI (out)			I
	E,S	gM (out)	1		no change
RS data	x x x	gM(in) gS(in) gI(in)		 1	no change no change no change
own WS	x	gM(out)			I
own WB	M,O	gM (out)			I
	Os	gS (out)			I
	I	gI (out)			I
	S,E	gM (out)	1		I
own invalidate WS	x	gM(out)			I

TABLE 3-9 summarizes Memory controller actions for SSM RMW (Read Modify Write) transactions.

TABLE 3-9 Memory controller actions for SSM RMW transactions

SSM RMW transaction	OwnIn	Memory Controller Action
foreign RTSU	0	Perform an atomic RMW with MTag = gS on the SDRAM through the DCDS, and read data is delivered to the Sun Fireplane Interconnect
	1	Read cancellation, no atomic MTag update is scheduled by memory controller
foreign RTOU	0	Perform an atomic RMW with MTag = gI on the SDRAM through the DCDS, and read data is delivered to the Sun Fireplane Interconnect
	1	Perform a memory write with MTag = gI (do not care data)
foreign UGM	0	Perform an atomic RMW with MTag = gM to the home memory
	1	WS will be issued from SSM device

3.4 Diagnostics Control and Accesses

The diagnostics control and data registers are accessed through Load/Store Alternate (`LDXA/STXA`) instructions.

Note – Attempts to access these registers while in non-privileged mode cause a *privileged_action* exception (with SFSR.FT = 1, privilege violation). User accesses can be accomplished through system calls to these facilities. See *I/D Synchronous Fault Status Register (SFSR)* in the *UltraSPARC III Cu Processor User's Manual* for SFSR details.

A store (`STXA | STDFA`) to any internal debug or diagnostic register requires a `MEMBAR #Sync` before another load instruction is executed. Furthermore, the `MEMBAR` must be executed in or before the delay slot of a delayed control transfer instruction of any type. This requirement is not just to guarantee that result of the `store` is seen but is also imposed because the store may corrupt the load data if there is not an intervening `MEMBAR #Sync`.

For more predictable behavior, it may be desirable to park the other logical processor when performing ASI accesses to a shared resource. If the other logical processor is not parked, it may perform operations making use of and/or modifying the shared resource being accessed. The UltraSPARC IV+ processor does not allow simultaneous ASI write accesses to shared resources by both logical processors. The other logical processor will be parked or disabled when performing write accesses to any of the ASIs described in this chapter.

3.5 Instruction Cache Diagnostic Accesses

Three I-cache diagnostic accesses are supported:

- Instruction cache instruction fields access
- Instruction cache tag/valid fields access
- Instruction cache snoop tag fields access

In the following ASI descriptions, "---" means reserved and unused bits. These bits are treated as don't care for ASI write operations, and read as zero by ASI read operations.

3.5.1 Instruction Cache Instruction Fields Access

ASI 66_{16}, per logical processor

Name: `ASI_ICACHE_INSTR`

TABLE 3-10 Instruction Cache Instruction Access Address Format

Bits	Field	Description
[63:17]	Mandatory value	Should be 0.
[16:15]	IC_way	This 2-bit field selects a way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3
[14:3]	IC_addr	This 12-bit index, which corresponds to VA[13:2] of the instruction address, selects a 32-bit instruction and associated predecode bits and parity bit
[2:0]	Mandatory value	Should be 0.

The data format for the instruction cache instruction fields is shown in TABLE 3-11.

TABLE 3-11 Instruction Cache Instruction Access Data Format

Bits	Field	Description
[63:43]	Mandatory value	Should be 0.
[42]	IC_parity	Odd-parity bit of the 32-bit instruction field plus 9 predecode bits
[41:32]	IC_predecode[9:0]	10 predecode bits associated with the instruction field
[31:0]	IC_instr	32-bit instruction field

IC_predecode[4:0] represents the following pipes:

TABLE 3-12 Definition of predecode bits[4:0]

Bits	Field
[4]	FM
[3]	FA
[2]	MS
[1]	BR
[0]	AX

IC_predecode[9:5] represents the following:

TABLE 3-13 Definition of predecode bits[9:5]

[9]	[8]	[7]	[6]	[5]	IC_instr [30]	IC_instr [28]	Annul bit[a]	
0	0	0	0	0	0	-	0	not a cti
1	-	-	-	-	-	-	-	cti (done, retry, jmpl, return, br, call)
1	0	-	-	-	-	-	-	done/retry
1	1	-	-	-	-	-	-	dcti (jmpl, call, return, br)
1	1	0	0	-	-	-	-	regular jmpl
1	1	0	1	0	-	-	-	pop RAS JMPL, return
1	1	0	1	1	-	-	-	push RAS JMPL
1	1	1	-	-	1	-	-	call
1	1	1	0	-	1	-	-	call w/o push RAS (followed by restore or write %o7)
1	1	1	0	-	0	-	0	unconditional
1	1	1	0	-	0	-	1	unconditional, annul
1	1	1	0	-	0	0	0	bn
1	1	1	0	-	0	0	1	bn, a
1	1	1	0	-	0	1	0	ba
1	1	1	0	-	0	1	1	ba, a
1	1	1	1	-	0	-	0	bc
1	1	1	1	-	0	-	1	bc, c

IC_parity:
Odd-parity bit of the 32-bit instruction field plus 9 predecode bits. IC_predecode[6] is not parity protected due to implementation considerations.

IC_instr[10:0] and IC_predecode[5] are not parity protected for PC-relative instructions.

IC_predecode[7] is used to determine whether an instruction is PC-relative or not. Since this PC-relative status bit is used to determine which instruction bits will be parity protected, if IC_predecode[7] is flipped, a non-PC-relative CTI will be treated as PC-relative CTI, or vice versa, when computing the parity value. Since the parity computation for the two types of instructions are different, a trap may or may not occur. To allow code to be written to check the operation of the parity detection hardware, the following equation can be used:

*For a non-PC-relative instruction (*IC_predecode[7] == 0):
IC_parity = XOR(IC_instr[31:0], IC_predecode[9:7, 5:0])

*For a PC-relative instruction (*IC_predecode[7]==1):
IC_parity = XOR(IC_instr[31:11], IC_predecode[9:7, 4:0])

The PC-relative instructions are `BPcc`, `Bicc`, `BPr`, `CALL`, `FBfcc` and `FBPfcc`, where IC_predecode[7] == 1.

Note – After ASI read/write to ASI 66_{16}, 67_{16}, 68_{16}, 69_{16}, $6A_{16}$, $6E_{16}$, or $6F_{16}$ instruction cache consistency may be broken, even if the instruction cache is disabled. The reason is that invalidates to the instruction cache may collide with the ASI load/store. Thus, before these ASI accesses, the instruction cache must be turned off. Then, before the instruction cache is turned on again, all of the instruction cache valid bits must be cleared to keep cache consistency.

3.5.2 Instruction Cache Tag/Valid Fields Access

ASI 67_{16}, per logical processor

Name: `ASI_ICACHE_TAG`

The address format for the instruction cache tag and valid fields are shown in TABLE 3-14.

TABLE 3-14 Instruction Cache Tag/Valid Access Address Format

Bits	Field	Description
[63:17]	Mandatory value	Should be 0.
[16:15]	IC_way	A 2-bit field that selects a way (4-way associative). 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[14:7]	IC_addr	IC_addr [14:7] corresponds to VA[13:6] of the instruction address. It is used to index the physical tag, microtag and valid/load predict bit arrays.
[6:5]	IC_addr	Since the I-Cache line size is 64bytes, sub-blocked as two 32bytes, IC_addr[6] is used to select a sub-block of the valid/load predict bit (LPB) array. This sub-block selection is not needed for physical and microtag arrays as they are common for both sub-blocks. Only one half of the load predict bit data from a cache line can be read in each cycle. Hence IC_addr[5] is used to select the upper or lower half of the load predict bits. IC_addr[5] = 0 Corresponds to the upper half of the load predict bits IC_addr[5] = 1 Corresponds to the lower half of the load predict bits The valid bit is always read out along with the load predict bits. Note: IC_addr[6:5] is a don't care in accessing physical tag and microtag arrays.
[4:3]	IC_tag	Instruction cache tag number: 00, 01, and 10. IC_tag selects whether physical tag, microtag, or valid/load predict bit array is accessed.
[2:0]	Mandatory value	Should be 0.

3.5.2.1 IC_tag: I-cache tag numbers

TABLE 3-15 through TABLE 3-19 illustrate the meaning of the tag numbers in IC_tag. In the tables, Undefined means the value of these bits is undefined on reads and must be masked off by software.

00 = Physical Address Tag.

Access I-cache physical address tag. The data format for I-cache physical address tag is shown in TABLE 3-15.

TABLE 3-15 Data Format for I-cache Physical Address Tag Field

Bits	Field	Description
[63:38]	Mandatory value	Should be 0.
[37]	Parity	Parity is the odd parity of the IC_tag fields.
[36:8]	IC_tag	IC_tag is the 29-bit physical tag field (PA[41:13] of the associated instructions).
[7:0]	Undefined	The value of these bits is undefined on reads and must be masked off by software.

01 = Microtag

Access I-cache microtag. The data format for the I-cache microtag is shown in TABLE 3-16.

TABLE 3-16 Data Format for I-cache Microtag Field

Bits	Field	Description
[63:46]	Mandatory value	Should be 0.
[45:38]	IC_utag	IC_utag is the 8-bit virtual microtag field (VA[21:14] of the associated instructions).
[37:0]	Undefined	The value of these bits is undefined on reads and must be masked off by software.

Note – The I-cache microtags must be initialized after power-on reset and before the instruction cache is enabled. For each of the four ways of each index of the instruction cache, the microtags must contain a unique value. For example, for index 0, the four microtags could be initialized to 0, 1, 2, and 3, respectively. The values need not be unique across indices; in the previous example 0, 1, 2, and 3 could also be used in cache index 1, 2, 3, and so on.

---

10 = Valid/predict tag

Access I-cache Valid/Load Predict Bits.

Note – Any write to the I-cache Valid/Predict array will update both sub-blocks simultaneously. Reads can, however, happen individually to any sub-block.

TABLE 3-17 shows the write data format for I-cache Valid/Load Predict Bits (LPB)

TABLE 3-17 Format for Writing I-cache Valid/Predict Tag Field Data

Bits	Field	Description
[63:56]	Mandatory value	Should be 0.
[55]	Valid1	Valid1 is the Valid bit for the 32-byte sub-block given by IC_addr[14:7] with IC_addr[6]=1.
[54]	Valid0	Valid0 is the valid bit for the other 32-byte sub-block given by the same IC_addr[14:7], but with IC_addr[6]=0.
[53:46]	IC_vpred1[7:0]	IC_vpred1 is the 8-bit LPB for eight instructions starting at the 32-byte boundary align address given by IC_addr[14:7] with IC_addr[6] = 1.
[45:38]	IC_vpred0[7:0]	IC_vpred0 are the LPB bits for eight instructions of the other 32-byte sub-block given by the same IC_addr[14:7], but with IC_addr[6]=0.
[37:0]	Undefined	The value of these bits is undefined on reads and must be masked off by software.

TABLE 3-18 shows the read data format for the upper bits of the I-cache Valid/LPB array.

TABLE 3-18 Format for Reading Upper Bits of Valid/Predict Tag Field Data

Bits	Field	Description
[63:51]	Mandatory value	Should be 0.
[50]	Valid	Valid is the Valid bit for the 32-byte sub-block.
[49:46]	IC_vpred[7:4]	IC_vpred is the upper 4- LPB bits for the eight instructions starting at the 32-byte boundary align address given by IC_addr.
[45:0]	Mandatory value	Should be 0.

TABLE 3-19 shows the read data format for the lower bits of the I-cache Valid/LPB array.

TABLE 3-19 Format for Reading Lower Bits of Valid/Predict Tag Field Data

Bits	Field	Description
[63:51]	Mandatory value	Should be 0.
[50]	Valid	Valid is the Valid bit for the 32-byte sub-block.
[49:46]	IC_vpred[3:0]	IC_vpred is the lower 4- LPB bits for the eight instructions starting at the 32-byte boundary align address given by IC_addr.
[45:0]	Undefined	The value of these bits is undefined on reads and must be masked off by software.

3.5.3 Instruction Cache Snoop Tag Fields Access

ASI 68_{16}, per logical processor

Name: ASI_ICACHE_SNOOP_TAG

TABLE 3-20 The address format for the I-cache snoop tag

Bits	Field	Description
[63:16]	Mandatory value	Should be 0.
[16:15]	IC_way	This 2-bit field selects a way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[14:7]	IC_addr	This 8-bit index (VA[13:6]) selects a cache tag.
[6:0]	Mandatory value	Should be 0.

Data Format of Instruction Cache Snoop Tag is described below in TABLE 3-21.

TABLE 3-21 The Data Format of I-cache Snoop Tag

Bits	Field	Description
[63:38]	Undefined	The value of these bits is undefined on reads and must be masked off by software.
[37]	IC_snoop_tag_parity	Odd parity value of the IC_snoop_tag fields.
[36:8]	IC_snoop_tag	The 29-bit physical tag field (PA[41:13] of the associated instructions).
[7:0]	Undefined	The value of these bits is undefined on reads and must be masked off by software.

3.6 Instruction Prefetch Buffer Diagnostic Accesses

3.6.1 Instruction Prefetch Buffer Data field Accesses

ASI 69_{16}, per logical processor

Name: ASI_IPB_DATA

The address format of the instruction prefetch buffer (IPB) data array access is shown in TABLE 3-22.

TABLE 3-22 Instruction Prefetch Buffer Data access Address Format

Bits	Field	Description
[63:10]	Mandatory value	Should be 0.
[9:3]	IPB_addr	IPB_addr[9:7] is a 3-bit index (VA[9:7]) that selects one entry of the 8-entry prefetch data. IPB_addr[6] is used to select a 32-byte sub-block of the 64-byte cache line and IPB_addr[5:3] is used to select one instruction from the 32-byte sub-block.
[2:0]	Mandatory value	Should be 0.

The instruction prefetch buffer data format is shown in TABLE 3-23.

TABLE 3-23 Instruction Prefetch Buffer Data Format

Bits	Field	Description
[63:43]	Mandatory value	Should be 0.
[42]	IPB_parity	The parity bit for IPB instructions are computed the same way as the I-cache parity bit.
[41:32]	IPB_predecode	This is similar to instruction cache data format.
[31:0]	IPB_instr	This is similar to instruction cache data format.

3.6.2 Instruction Prefetch Buffer Tag field Accesses

ASI $6A_{16}$, per logical processor

Name: `ASI_IPB_TAG`

The address format of the instruction prefetch buffer tag array access is shown in TABLE 3-24.

TABLE 3-24 Instruction Prefetch Buffer Tag/Valid Field Read Access Data Format

Bits	Field	Description
[63:10]	Mandatory value	Should be 0.
[9:6]	IPB_addr	IPB_addr[9:7] is a 3-bit index (VA[9:7]) that selects one entry of the 8-entry instruction prefetch buffer tag. During instruction prefetch buffer tag reads or writes, the valid bit of one of the 32-byte sub-blocks of that entry will also be read or written. IPB_addr[6] is used to select the valid bit of the desired sub-block of an instruction prefetch buffer entry.
[5:0]	Mandatory value	Should be 0.

The data format of the instruction prefetch buffer tag array access is shown in TABLE 3-25.

TABLE 3-25 Instruction Prefetch Buffer Tag Field Write Access Data Format

Bits	Field	Description
[63:42]	Mandatory value	Should be 0.
[41]	IPB_valid	IPB_valid represents the valid bit of the 32-byte sub-block of an instruction prefetch buffer entry.
[40:8]	IPB_tag	IPB_tag represents the 33-bit physical tag.
[5:0]	Mandatory value	Should be 0.

Since the IPB_tag array is small, it is not parity protected.

3.7 Branch Prediction Diagnostic Accesses

3.7.1 Branch Predictor Array Accesses

ASI $6F_{16}$, per logical processor

Name: `ASI_BRANCH_PREDICTION_ARRAY`

The address format of the branch-prediction array access is shown in TABLE 3-26.

TABLE 3-26 Branch Prediction Array Access Address Format

Bits	Field	Description
[63:16]	Mandatory value	Should be 0.
[15:3]	BPA_addr	BPA_addr is a 13-bit index (VA[15:3]) that selects a branch prediction array entry.
[2:0]	Mandatory value	Should be 0.

The branch prediction array entry is shown in TABLE 3-27.

TABLE 3-27 Branch Prediction Array Data Format

Bits	Field	Description
[63:4]	Undefined	The value of these bits is undefined on reads and must be masked off by software.
[3:2]	PNT_Bits	The two predict bits if the last prediction was NOT_TAKEN.
[1:0]	PT_Bit	The two predict bits if the last prediction was TAKEN.

3.7.2 Branch Target Buffer Accesses

ASI $6E_{16}$, per logical processor

Name: `ASI_BTB_DATA`

The address format of the branch target buffer access is described in TABLE 3-28.

TABLE 3-28 Branch Target Buffer Access Address Format

Bits	Field	Description
[63:10]	Mandatory value	Should be 0.
[9:5]	BTB_addr	BTB_addr is a 5-bit index (VA[9:5]) that selects a branch target buffer entry
[4:0]	Mandatory value	Should be 0.

Branch Target Buffer array entry is described below and described in TABLE 3-29.

TABLE 3-29 Branch Target Buffer Data Format

Bits	Field	Description
[63:2]	Target Address	The address bits of the predicted target instruction.
[1:0]	*Reserved*	These two bits are unused.

Note – The Branch Target Buffer is not ASI accessible in RED_state.

3.8 Data Cache Diagnostic Accesses

Five D-cache diagnostic accesses are supported:

- Data cache data fields access
- Data cache tag/valid fields access
- Data cache microtag fields access
- Data cache snoop tag access
- Data cache invalidate

3.8.1 Data Cache Data Fields Access

ASI 46_{16}, per logical processor

Name: `ASI_DCACHE_DATA`

TABLE 3-30 Data Cache Data/Parity Access Address Format

Bits	Field	Description
[63:17]	Mandatory value	Should be 0.
[16]	DC_data_parity	A 1-bit index that selects a data (= 0) or a parity (= 1).
[15:14]	DC_way	A 2-bit index that selects an associative way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[13:3]	DC_addr	An 11-bit index that selects a 64-bit data field.
[2:0]	Mandatory value	Should be 0.

The address format for D-cache data access is shown in TABLE 3-31.

The data formats for D-cache data access when DC_data_parity = 0 and 1 are shown in TABLE 3-31 and TABLE 3-32, respectively. DC_parity is 8-bit data parity (odd parity).

TABLE 3-31 Data Cache Data Access Data Format

Bits	Field	Description
[63:0]	DC_data	DC_data is 64-bit data.

TABLE 3-32 Data Cache Data Access Data Format When DC_data_parity = 1

Bits	Field	Description
[63:8]	Mandatory value	Should be 0.
[7:0]	DC_parity	DC_parity is 8-bit data parity (odd parity).

A `MEMBAR #Sync` is required before *and* after a load or store to `ASI_DCACHE_DATA`.

TABLE 3-33 Data parity bits

Data Bits	Parity Bit
[63:56]	7
[55:48]	6
[47:40]	5
[39:32]	4
[31:24]	3
[23:16]	2
[15:8]	1
[7:0]	0

The TABLE 3-33 shows the data parity bits and the corresponding data bits that are parity protected.

Note – During ASI writes to DC_data, parity bits are not generated from the data.

3.8.2 Data Cache Tag/Valid Fields Access

ASI 47_{16}, per logical processor

Name: `ASI_DCACHE_TAG`

The address format for the D-cache Tag/Valid fields is shown in TABLE 3-34.

TABLE 3-34 Data Cache Tag/Valid Access Address Format

Bits	Field	Description
[63:16]	Mandatory value	Should be 0.
[15:14]	DC_way	A 2-bit index that selects an associative way (4-way associative) 2’b00: Way 0, 2’b01: Way, 2’b10: Way 2, 2’b11: Way3.
[13:5]	DC_addr	A 9-bit index that selects a tag/valid field (512 tags).
[4:0]	Mandatory value	Should be 0.

TABLE 3-35 Data Cache Tag/Valid Access Data Format

Bits	Field	Description
[63:31]	Mandatory value	Should be 0.
[30]	DC_tag_parity	The 1-bit odd-parity bit of DC_tag.
[29:1]	DC_tag	The 29-bit physical tag (PA[41:13] of the associated data).
[0]	DC_valid	The 1-bit valid field.

Note – A `MEMBAR #Sync` is required before and after a load or store to `ASI_DCACHE_TAG.`

During ASI writes, DC_tag_parity is not generated from data bits [29:1], but data bit 30 is written as the parity bit. This will allow testing of D-cache tag parity error trap.

3.8.3 Data Cache Microtag Fields Access

ASI 43_{16}, per logical processor

Name: `ASI_DCACHE_UTAG`

The address format for the D-cache microtag access is shown in TABLE 3-36.

TABLE 3-36 Data Cache Microtag Access Address Format

Bits	Field	Description
[63:16]	Mandatory value	Should be 0.
[15:14]	DC_way	A 2-bit index that selects an associative way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[13:5]	DC_addr	A 9-bit index that selects a tag/valid field (512 tags).
[4:0]	Mandatory value	Should be 0.

The data format for D-cache microtag access is shown in TABLE 3-37.

TABLE 3-37 Data Cache Microtag Access Data Format

Bits	Field	Description
[63:8]	Mandatory value	Should be 0.
[7:0]	DC_utag	DC_utag is the 8-bit virtual microtag (VA[21:14] of the associated data).

Note – A `MEMBAR #Sync` is required before and after a load or store to `ASI_DCACHE_UTAG`.

The data cache microtags must be initialized after power-on reset and before the data cache is enabled. For each of the four ways of each index of the data cache, the microtags must contain a unique value; for example, for index 0, the four microtags could be initialized to 0, 1, 2, and 3. respectively. The values need not be unique across indices; in the previous example 0, 1, 2, and 3 could also be used in cache index 1, 2, 3, and so on.

3.8.4 Data Cache Snoop Tag Access

ASI 44_{16}, per logical processor

Name: `ASI_DCACHE_SNOOP_TAG`

The address format for the D-cache snoop tag fields is shown in TABLE 3-38.

TABLE 3-38 Data Cache Snoop Tag Access Address Format

Bits	Field	Description
[63:16]	Mandatory value	Should be 0.
[15:14]	DC_way	A 2-bit index that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[13:5]	DC_addr	A 9-bit index that selects a snoop tag field (512 tags).
[4:0]	Mandatory value	Should be 0.

The data format for D-cache snoop tag access is shown in TABLE 3-39.

TABLE 3-39 Data Cache Snoop Tag Access Data Format

Bits	Field	Description
[63:31]	Mandatory value	Should be 0.
[30]	DC_snoop_tag_parity	Odd-parity bit of DC_snoop_tag.
[29:1]	DC_snoop_tag	The 29-bit physical snoop-tag (PA[41:13] of the associated data, PA[42] is always 0 for cacheable).
[0]	Mandatory value	Should be 0.

Note – A `MEMBAR #Sync` is required before *and* after a load or store to `ASI_DCACHE_SNOOP_TAG`.

During ASI writes, DC_snoop_tag_parity is not generated from data bits [29:1], but data bit 30 is written as the parity bit. This will allow testing of D-cache snoop tag parity error trap.

3.8.5 Data Cache Invalidate

ASI 42_{16}, per logical processor

Name: `ASI_DCACHE_INVALIDATE`

A store that uses the Data Cache Invalidate ASI invalidates a D-cache line that matches the supplied physical address from the data cache. A load to this ASI returns an undefined value and does not invalidate a D-cache line.

The address format for D-cache invalidate is shown in TABLE 3-40.

TABLE 3-40 Data Cache Invalidate Address Format

Bits	Field	Description
[63:41]	Mandatory value	Should be 0.
[42:5]	Physical_Address	D-cache line matching Physical_Address is invalidated. If there is no matching D-cache entry, then the ASI store is a NOP.
[4:0]	Mandatory value	Should be 0.

3.9 Write Cache Diagnostic Accesses

Three W-cache diagnostic accesses are supported:

- W-cache diagnostic state register access
- W-cache diagnostic data register access
- W-cache diagnostic tag register access

3.9.1 Write Cache Diagnostic State Register Access

ASI $38_{16,}$ per logical processor

Name: `ASI_WCACHE_STATE`

The address format for W-cache diagnostic state register access is shown in TABLE 3-41.

TABLE 3-41 Write Cache Diagnostic State Access Address Format

Bits	Field	Description
[63:11]	Mandatory value	Should be 0.
[10:6]	WC_entry	A 5-bit index (VA[10:6]) that selects a W-cache entry for ASI writes. These bits are don't care for ASI reads.
[5:0]	Mandatory value	Should be 0.

The data format for W-cache diagnostic state register write access is shown in TABLE 3-42.

TABLE 3-42 Write Cache Diagnostic State Access Write Data Format

Bits	Field	Description
[63:2]	Mandatory value	Should be 0.
[1:0]	wcache_state	A 2-bit W-cache state field encoded as follows: 2'b00 = Invalid, 2'b10 = Owner, 2'b11 = Modified

The data format for W-cache diagnostic state register read access is showed in TABLE 3-43.

TABLE 3-43 Write Cache Diagnostic State Access Read Data Format

Bits	Field	Description
[63:0]	{entry31_state[1:0], entry30_state[1:0], ..., entry2_state[1:0], entry1_state[1:0], entry0_state[1:0]}	The 2-bit state of all 32 W-cache entries - entry 0 through entry 31.

Note – A `MEMBAR #Sync` is required before and after a load or store to `ASI_WCACHE_STATE`.

3.9.2 Write Cache Diagnostic Data Register Access

ASI 39_{16}, per logical processor

Name: `ASI_WCACHE_DATA`

The address format for W-cache diagnostic data access is shown in TABLE 3-44.

TABLE 3-44 Write Cache Diagnostic Data Access Address Format

Bits	Field	Description
[63:12]	Mandatory value	Should be 0.
[11]	WC_ecc_error	A 1-bit entry that selects between ecc_error and data array: 1= ecc_error, 0 = data.
[10:6]	WC_entry	A 5-bit index (VA[10:6]) that selects a W-cache entry.
[5:3]	WC_dbl_word	A 3-bit field that selects one of 8 doublewords, read from the Data Return. When reading from ECC, bit[5] determines which bank of ecc bit it is reading from.
[2:0]	Mandatory value	Should be 0.

TABLE 3-45 Write Cache Diagnostic Data Access Data Format

Bits	Field	Description
[63:0]	wcache_data	The data format for W-cache diagnostic data access when WC_ecc_error = 0 and wcache_data is a doubleword of W-cache data.

In TABLE 3-46 bit[63] represents the ecc_error bit and the rest of the data are don't cares.

TABLE 3-46 Write Cache Diagnostic Data Access Data Format

Bits	Field	Description
[63]	ecc_error	The data format for W-cache diagnostic data access when WC_ecc_error=1. The ecc_error bit, if set, indicates that an ECC error occurred when the corresponding W-cache line was loaded from the L2-cache.
[62:0]	*Reserved*	Reserved for future implementation.

Note – A `MEMBAR #Sync` is required before and after a load or store to `ASI_WCACHE_DATA`.

3.9.3 Write Cache Diagnostic Tag Register Access

ASI $3A_{16}$, per logical processor

Name: `ASI_WCACHE_TAG`

The address format for W-cache diagnostic tag register access is shown in TABLE 3-47.

TABLE 3-47 Write-Cache Tag Register Access Address Format

Bits	Field	Description
[63:11]	*Reserved*	Reserved for future implementation.
[10:6]	WC_entry	A 5-bit index (VA[10:6]) that selects a W-cache entry.
[5:0]	*Reserved*	Reserved for future implementation.

The data format for W-cache diagnostic tag register access is shown in TABLE 3-48.

TABLE 3-48 Write-Cache Tag Register Access Data Format

Bits	Field	Description
[63:37]	Undefined	Must Be zero. Note: Writing a nonzero value to this field may generate an undefined result. Software should not rely on any specific behavior.
[36:0]	WC_physical_tag	A 37-bit physical tag (PA[42:6]) of the associated data.

Note – A `MEMBAR #Sync` is required before and after a load or store to `ASI_WCACHE_TAG`.

3.10 Prefetch Cache Diagnostic Accesses

Four P-cache diagnostic accesses are supported:

- P-cache status data register access
- P-cache diagnostic data register access
- P-cache virtual tag/valid fields access

- P-cache snoop tag register access

3.10.1 Prefetch Cache Status Data Register Access

ASI 30_{16}, per logical processor

Name: `ASI_PCACHE_STATUS_DATA`

The address format of the P-cache status data register access is shown in TABLE 3-49.

TABLE 3-49 Prefetch Cache Status Data Access Address Format

Bits	Field	Description
[63:11]	*Reserved*	Reserved for future implementation.
[10:9]	PC_way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[8:6]	PC_addr	A 3-bit index (VA[8:6]) that selects a P-cache entry.
[5:0]	*Reserved*	Reserved for future implementation.

The data format of P-cache status data register is shown in TABLE 3-50.

TABLE 3-50 Data Format Bit Description

Bits	Field	Description
[63:58]	*Reserved*	Reserved for future implementation.
[57:50]	Parity_bits	Data Array Parity bits (odd parity).
[49]	Prefetch_Que_empty	A read-only field, accessed through an ASI read. (Not used in SRAM test mode) 1 Corresponds to empty 0 Corresponds to not empty.
[48:0]	pcache_data	49 bits of status from selected entry, as follows: BitFieldAttribute [48:46]pg_size[2:0]Page size 45pg_w Writable 44pg_priv Privileged 43pg_ebit Side effect 42pg_cp Cacheable Physical 41pg_cv Cacheable Virtual 40pg_le Invert endianness 39pg_nfo No fault 38PG_sn No snoop (not used) [37:1]paddr[42:6]PA[42:6] 0fetched Software Prefetch =1 Hardware prefetch=0

TABLE 3-51 shows the parity bits in the P-cache status data array and the corresponding P-cache data bytes that are parity protected.

TABLE 3-51 P-cache status data array

P-cache Status Data Array bit	P-cache Data Bytes
[50]	[63:56] (corresponding to VA[5:3] = 111).
[51]	[55:48] (corresponding to VA[5:3] = 110).
[52]	[47:40] (corresponding to VA[5:3] = 101).
[53]	[39:32] (corresponding to VA[5:3] = 100).
[54]	[31:24] (corresponding to VA[5:3] = 011).
[55]	[23:16] (corresponding to VA[5:3] = 010).
[56]	[15:8] (corresponding to VA[5:3] = 001).
[57]	[7:0] (corresponding to VA[5:3] = 000).

3.10.2 Prefetch Cache Diagnostic Data Register Access

ASI 31_{16}, per logical processor

Name: `ASI_PCACHE_DATA`

The address format for P-cache diagnostic data register access is shown in TABLE 3-52.

TABLE 3-52 Prefetch Cache Diagnostic Data Access Address Format

Bits	Field	Description
[63:58]	*Reserved*	Reserved for future implementation.
[10:9]	PC_way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[8:6]	PC_addr	A 3-bit index (VA[8:6]) that selects a P-cache entry.
[5:3]	PC_dbl_word	A 3-bit field that selects one of 8 doublewords, read from the Data Return.
[2:0]	*Reserved*	Reserved for future implementation.

The data format for P-cache diagnostic data register access is shown in TABLE 3-53.

TABLE 3-53 Prefetch Cache Diagnostic Data Access Data Format

Bits	Field	Description
[63:0]	pcache_data	pcache_data is a doubleword of P-cache data.

3.10.3 Prefetch Cache Virtual Tag/Valid Fields Access

ASI 32_{16}, per logical processor

Name: `ASI_PCACHE_TAG`

The address format for P-cache virtual tag/valid fields access is shown in TABLE 3-54.

TABLE 3-54 Prefetch Cache Tag Register Access Address Format

Bits	Field	Description
[63:12]	*Reserved*	Reserved for future implementation.
[11]	PC_port	A 1-bit field that selects port 0 or port 1 of the dual-ported RAM. Both ports give the same values. This bit is used only during manufacture testing.
[10:9]	PC_way	A 2-bit entry that selects a way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[8:6]	PC_addr	A 3-bit index (VA[8:6]) that selects a P-cache tag entry.
[5:0]	*Reserved*	Reserved for future implementation.

The data format for P-cache tag/valid fields access is shown in TABLE 3-55.

TABLE 3-55 Prefetch Cache Tag Register Access Data Format

Bits	Field	Description
[63:62]	*Reserved*	Reserved for future implementation.
[61]	PC_bank0_valid	Valid bit for RAM bits [511:256].
[60]	PC_bank1_valid	Valid bit for RAM bits [255:0].
[59:58]	context	nucleus_cxt and secondary_cxt bits of the load instruction.
[57:0]	PC_virtual_tag	58-bit virtual tag (VA[63:6]) of the associated data.

The P-cache keeps a 2-bit context tag, bit[59:58] of the P-cache tag register. The two bits are decoded as follows:

00 Primary Context
01 Secondary Context
10 Nucleus Context
11 Not used

A nucleus access will never hit P-cache for a primary or a secondary context entry even if it has the same VA.

Moving between nucleus, primary, or secondary context does not require P-cache invalidation.

All entries in the P-cache are invalidated on a write to a context register. The write invalidates the prefetch queue such that once the data is returned from the external memory unit, it is not installed in the P-cache.

3.10.4 Prefetch Cache Snoop Tag Register Access

Name: `ASI_PCACHE_SNOOP_TAG`

The address format of P-cache snoop tag register access is shown in TABLE 3-56.

TABLE 3-56 Prefetch Snoop Tag Access Address Format

Bits	Field	Description
[63:12]	*Reserved*	Reserved for future implementation.
[11]	PC_port	A 1-bit field that selects port 0 or port 1 of the dual-ported RAM. Both ports give the same values, and this bit is used only during manufacture testing.
[10:9]	PC_way	A 2-bit entry that selects a way (4-way associative) 2'b00: Way 0, 2'b01: Way, 2'b10: Way 2, 2'b11: Way3.
[8:6]	PC_addr	A 3-bit index (VA[8:6]) that selects a P-cache entry.
[5:0]	*Reserved*	Reserved for future implementation.

The data format for P-cache snoop tag register access is shown in TABLE 3-57.

TABLE 3-57 Prefetch Cache Snoop Tag Access Data Format

Bits	Field	Description
[63:38]	*Reserved*	Reserved for future implementation.
[37]	PC_valid_bit	A 1-bit field that indicates a valid physical tag entry.
[36:0]	PC_physical_tag	The 37-bit physical tag of associated data.

3.11 L2-cache Diagnostics & Control Accesses

Separate ASIs are provided for reading and writing the L2-cache tag and data as well as the L2-cache control register. This section describes three L2-cache accesses:

- L2-cache control register
- L2-cache tag/state/LRU/ECC field diagnostics accesses
- L2-cache data/ECC field diagnostic accesses

3.11.1 L2-cache Control Register

ASI $6D_{16}$, Read and Write, Shared by both logical processors

VA = 0

Name: `ASI_L2CACHE_CONTROL`

The data format for L2-cache control register is shown in TABLE 3-58.

TABLE 3-58 L2-cache Control Register ***(1 of 2)***

Bits	Field	Description
[63:16]	*Reserved*	Reserved for future implementation.
[15]	Queue_timeout_detected	If set, indicates that one of the queues that need to access the l2/l3 pipeline was detected to timeout. After hard reset, this bit will be read as 1'b0.
[14]	WC1_status	If set, indicates that the W-cache of LP1 was stopped. After hard reset, this bit will be read as 1'b0.
[13]	WC0_status	If set, indicates that the W-cache of LP0 was stopped. After hard reset, this bit will be read as 1'b0.
[12]	Queue_timeout _disable	If set, disables the hardware logic that detects the progress of a queue. After hard reset, this bit will be read as 1'b0. Programmers should set this bit to ensure livelock-free operation if the throughput of the L2/L3 pipelines is reduced from 1/2 to 1/16 or 1/32. The throughput is reduced whenever the L2L3 arbiter enters the single issue mode under one of the following conditions: • The L2L3arb_single_issue_en field of the L2-cache control register is set. • The L2-cache is disabled by setting the L2_off field of the L2-cache control register. • The Write-cache of an enabled logical processor is disabled.
[11:9]	Queue_timeout	Controls the timeout period of the queues: timeout period = 2^(7 + 2*Queue_timeout) system cycles, where, Queue_timeout = 000, 001, ... 110, 111 This gives a timeout period ranging from 128 system cycles to 2M system cycles. After hard reset, this bit will be read as 3'b111.
[8]	L2_off	If set, disables the on-chip L2-cache. A separate one entry address and data buffer behave as a one-entry L2-cache for debugging purposes when the L2-cache is disabled. After hard reset, this bit will be read as 1'b0.
[7]	Retry_disable	If set, disables the logic that stops the W-cache(s) of one or both LPs. After hard reset, this bit will be read as 1'b0. It is recommended that programmers set this bit to avoid livelocks if the Write-cache of an enabled logical processor is enabled.
[6:5]	Retry_debug_counter	If Retry_disable is not set, these bits control the retry counter which stops the W-cache(s): 00: 1024 retries will stop the W-cache(s) 01: 512 retries will stop the W-cache(s) 10: 256 retries will stop the W-cache(s) 11: 128 retries will stop the W-cache(s) After hard reset, these bits will be read as 2'b00.
[4]	L2L3arb_single_issue_frequency	Controls the frequency of issue in the single issue mode: 0: one transaction every 16 cycles 1: one transaction every 32 cycles After hard reset, will be read as 1'b0.

TABLE 3-58 L2-cache Control Register *(2 of 2)*

Bits	Field	Description
[3]	L2L3arb_single_issue_en	If set, the L2L3 arbiter enters the single issue mode, where a transaction to L2 and/or L3 pipeline(s), except snoop and ASI_SRAM_FAST_INIT_SHARED transactions, will be issued every 16 or 32 cycles, as specified by L2L3arb_single_issue_frequency. After hard rest, this bit will be read as 1'b0.
[2]	L2_split_en	If set, enables the following scheme: for misses generated by LP0, allocate only in ways 0 and 1 of the L2-cache; and for misses generated by LP1, allocate only in ways 2 and 3 of the L2-cache. After hard rest, this bit will be read as 1'b0.
[1]	L2_data_ecc_en	If set, enables ECC checking on L2-cache data bits. After hard rest, this bit will be read as 1'b0.
[0]	L2_tag_ecc_en	If set, enables ECC checking on L2-cache tag bits. If ECC checking is disabled, there will never be ECC errors due to L2-cache tag access. However, ECC generation and write to L2-cache tag will still occur in correct manner. After hard rest, this bit will be read as 1'b0.

Note – Bit[63:16] of the L2-cache control register are reserved. They are treated as don't care for ASI write operations and read as zero for ASI read operations.

The L2_off field in the L2-cache control register should not be changed during run-time. Otherwise the behavior is undefined. The rest of the fields can be programmed during run time without breaking the functionality and they will take effect without requiring a system reset.

Bits[15:13] (Queue_timeout_detected, WC1_status, WC0_status bits) are sticky status bits. They are set by the hardware when the associated event occurs. Multiple bits can be set by the hardware. These bits are not ASI writable. An ASI read to the L2-cache control register will reset these bits to 0.

The value of Queue_timebout (bits [11:9]) should not be programmed to result in a timeout value longer than what the TOL field of the Fireplane Configuration Register is programmed to result in for CPQ timeout.

3.11.1.1 Notes on L2-cache Off Mode

- If the Write-cache is disabled in an enabled logical processor or if the L2-cache is disabled (i.e., L2_off =1), the L2L3arb_single_issue_en field in the L2-cache control register is ignored and the L2L3 arbiter hardware will automatically switch to the single issue mode.
- If the L2-cache is disabled, software should disable L2_tag_ecc_en, L2_data_ecc_en (i.e., no support for L2-cache tag/data ECC checking, reporting and correction), and L2_split_en in the L2-cache control register. EC_ECC_en in the L3-cache control register (ASI 75_{16}) should also be disabled in the L2-cache off mode, since L3-cache is a victim cache. Besides, L2-cache LRU replacement scheme will not work and only one pending miss is allowed when L2-cache is disabled. The inclusion property between L2-cache and primary caches will sustain.
- The one entry address and data buffer used in L2-cache off mode will be reset by hard reset. System reset or `ASI_SRAM_FAST_INIT_SHARED` (ASI $3F_{16}$) will not take any effect.

3.11.2 L2-cache Tag Diagnostic Access

ASI $6C_{16}$, Read and Write, Shared by both LPs

Name: `ASI_L2CACHE_TAG`

The L2-cache Tag Access Address format is shown in TABLE 3-59.

TABLE 3-59 L2-cache Tag Access Address Format

Bits	Field	Description
[63:24]	*Reserved*	Reserved for future implementation
[23]	disp_flush	specifies the type of access: 1'b1: displacement flush - write back the selected L2-cache line, both clean and modified, to L3-cache and invalidate the line in L2-cache. When this bit is set, the ASI data portion is unused. 1'b0: direct L2-cache tag diagnostic access - read/write the tag, state, ECC, and LRU fields of the selected L2-cache line. In L2-cache on mode, if the line to be displacement flushed is in NA state (see *Notes on "NA" Cache State* on page 64), it will not be written out to L3-cache. The line remains in NA state in L2-cache. In L2-cache off mode, displacement flush is treated as a NOP. **Note:** For L2-cache displacement flush, use only LDXA (STXA has NOP behavior). Since L2-cache will return garbage data to the MS pipeline, it is recommended to use "ldxa [reg_address]ASI_L2CACHE_TAG, %g0" instruction format.
[22]	hw_ecc_gen_en	On read, this bit is don't care. On write, if set, it enables hardware ECC generation logic inside L2-cache tag SRAM. When this bit is not set, the ECC generated by hardware ECC generation logic is ignored -- what is specified in the ECC field of the ASI data portion will be written into the ECC entry.
[21]	Mandatory value	Should be 0.
[20:19]	way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way 1, 2'b10: Way 2, 2'b11: Way 3.
[18:6]	index	A 13-bit index (PA[18:6]) that selects an L2-cache entry.
[5:0]	Mandatory value	Should be 0.

The data format for L2-cache tag diagnostic access (disp_flush=0) is described in TABLE 3-60.

TABLE 3-60 L2-cache Tag Access Data Format

Bit	Field	Description
[63:43]	Mandatory value	Should be 0.
[42:19]	Tag	A 24-bit tag(PA[42:19])
[18:15]	Mandatory value	Should be 0.
[14:6]	ECC	A 9-bit ECC entry that protects L2-cache tag and state. The ECC bits protect all 4-ways of a given L2-cache index.
[5:3]	LRU	A 3-bit entry that records the access history of the 4 ways of a given L2-cache index. If L2_split_en in the L2-cache control register (ASI 6D16) is not set, the LRU is as described below. The LRU-pointed way will not be picked for replacement if the corresponding state is "NA". LRU[2:0] = 000way0 is the LRU LRU[2:0] = 001way1 is the LRU LRU[2:0] = 010way0 is the LRU LRU[2:0] = 011way1 is the LRU LRU[2:0] = 100way2 is the LRU LRU[2:0] = 101way2 is the LRU LRU[2:0] = 110way3 is the LRU LRU[2:0] = 111way3 is the LRU If L2_split_en in the L2-cache control register (ASI 6D16) is set, the LRU is as described below. LRU[2] is ignored and the logical processor ID of the logical processor that issues the request is used instead. {LP, LRU[1:0]} = 000 way0 is the LRU {LP, LRU[1:0]} = 001 way1 is the LRU {LP, LRU[1:0]} = 010 way0 is the LRU {LP, LRU[1:0]} = 011 way1 is the LRU {LP, LRU[1:0]} = 100 way2 is the LRU {LP, LRU[1:0]} = 101 way2 is the LRU {LP, LRU[1:0]} = 110 way3 is the LRU {LP, LRU[1:0]} = 111 way3 is the LRU
[2:0]	State	A 3-bit L2-cache state entry. The 3 state bits are encoded as follows: state[2:0] = 000 Invalid state[2:0] = 001 Shared state[2:0] = 010 Exclusive state[2:0] = 011 Owner state[2:0] = 100 Modified state[2:0] = 101 NA "Not Available" (see *Notes on "NA" Cache State* on page 64) state[2:0] = 110 Owner/Shared state[2:0] = 111 *Reserved*

Note – Bit[63:43] and bit[18:15] of `ASI_L2CACHE_TAG` data are treated as don't care for ASI write operations and read as zero for ASI read operations.

On reads to `ASI_L2CACHE_TAG`, regardless of the hw_ecc_gen_en bit, the tag and the state of the specified entry (given by the index and the way) will be read out along with the ECC and the LRU of the corresponding index. A common ECC and LRU is shared by all 4 ways of an index.

On writes to ASI_L2CACHE_TAG, the tag and the state will always be written into the specified entry. The LRU will always be written into the specified index. For the ECC field, if hw_ecc_gen_en is set, the ECC field of the data register is don't care. If the hw_ecc_gen_en is not set, the ECC specified in the data register will be written into the index. The hardware generated ECC will be ignored in that case.

Note – For the one entry address buffer used in the L2-cache off mode, no ASI diagnostic access is supported. The L2-cache tag array can be accessed as usual in the L2-cache off mode. However, the returned value is not guaranteed to be correct since the SRAM can be defective and this may be the reason to turn off the L2-cache.

3.11.2.1 Notes on L2-cache Tag ECC

- The ECC value of a zero L2-cache tag is also zero. Thus, after ASI_SRAM_FAST_INIT_SHARED (STXA $3F_{16}$), the ECC value is correct and all lines will be in the INVALID state.
- L2-cache tag ECC checking is carried out regardless of the L2-cache line is valid or not.
- If L2-cache tag diagnostic access encounters an L2-cache tag CE, the returned data will not be corrected, and raw L2-cache tag data will be returned regardless of L2_tag_ecc_en in L2-cache control register (ASI $6D_{16}$) is set or not.
- If there is an ASI write request to the L2-cache tag (does not include displacement flush) and the ASI write request wins the L2L3 pipeline arbitration, it will be sent to the L2L3 pipeline to access L2 tag and L3 tag at the same time. Within 15 cycles, if there is another request (I-Cache, D-Cache, P-cache, SIU snoop, or SIU copyback request) to the same cache index following the ASI request, the second access will get incorrect tag ECC data. It will not have any issues if two accesses are to the different index. To avoid the problem, the following procedure should be followed when software uses ASI write to L2-cache tag to inject L2-cache tag error.

3.11.2.2 Procedure For Writing ASI_L2CACHE_TAG:

1. Park the other logical processor.
2. Wait for the parking logical processor to be parked.
3. Turn off kernel pre-emption.
4. Block interrupts on this processor.
5. Displacement flush all 4 ways in L2-cache for the index to be error injected.
6. Load some data into the L2-cache.
7. Locate the data in the L2-cache and associated tag.
8. Read the L2-cache tag ECC (using ASI L2-cache tag read access).
9. Corrupt the tag ECC.
10. Store the tag ECC back (using ASI L2-cache tag write access).
11. Re-enable interrupts.
12. Unpark the other logical processor.

The reason to displacement flush all 4 ways is to guarantee that foreign snoop will have no effect to the index during ASI L2-cache tag write access even if the hazard window exists in the hardware.

3.11.2.3 Notes on L2-cache LRU Bits

- The LRU entry for a given L2-cache index is based on the following algorithm:
- For each 4-way set of L2 data blocks, a 3-bit structure is used to identify the least recently used "way". Bit[2] tracks which one of the two 2-way sets (way3/2 is one set; way1/0 is the other) is the least recently used. Bit[1] tracks which way of the of 2-way set (way3 and way2) is the least recently used. Bit[0] tracks which way of the other 2-way set (way1 and way0) is the least recently used. The LRU bits are reset to 3'b000 and updated based on the following scheme:

  bit[2] = 1 if hit in way1 or way0;

  bit[1] = 1 if hit in way2 or it remains 1 when not hit in way3;

  bit[0] = 1 if hit in way0 or it remains 1 when not hit in way1
- An example of LRU decoding: LRU = 3'b000 means way1 and way0 are less recently used than way3 and way2; way2 is less recently used than way3; way0 is less recently used than way1. This algorithm is like a binary tree, which is suitable for the split-cache mode as well. LRU bits are updated during fill or hits. No updates can happen during replacement calculation.
- LRU bits are not ECC protected.

3.11.2.4 Notes on "NA" Cache State

- The "NA (Not Available)" cache state is introduced to enhance the RAS feature and testability in the L2 and L3-caches. When a cache line is in "NA" state, it means this way will be excluded from the replacement algorithm. It can be used in the following scenarios.

1. During run time, the operating system can selectively disable specific index+way in L2 and L3-cache tag SRAMs based on soft error reporting in the L2 and L3-cache data SRAMs.
2. During run time, the operating system can make a 4-way set associative L2 and/or L3-cache behave like a direct-mapped, 2-way or 3-way set associative cache by programming NA state to certain ways of all indices.
3. During multi-probe phase, the tester can detect and store the bitmap information of the L2-cache data SRAM and disable certain index+way by writing NA state in the L2-cache tag SRAM.

- To ensure a smooth transition between "Available" (i.e., all other states accept "NA") and "Not Available" state during run time, the ASI write should follow the steps defined in *Procedure For Writing ASI_L2CACHE_TAG:* on page 63.
- When L2_split_en is disabled in L2-cache control register (ASI $6D_{16}$) or EC_split_en is disabled in L3-cache control register (ASI 75_{16}), the OS cannot program all 4 ways of an index to be in NA state. When L2_split_en or EC_split_en is set, the OS has to make sure at least one way among way0 and way1 of an index is available. Similarly, the same restriction applies for way2 and way3.

3.11.3 L2-cache Data Diagnostic Access

ASI $6B_{16}$, Read and Write, Shared by both logical processors

VA[63:22] = 0

VA[21] = ECC_sel

VA[20:19] = way

VA[18:6] = index

VA[5:3] = xw_offset

VA[2:0] = 0

Name: ASI_L2CACHE_DATA

The address format for L2-cache data diagnostic access is shown in TABLE 3-61.

TABLE 3-61 L2-cache Data Diagnostic Access

Bits	Field	Description
[63:22]	Mandatory value	Should be 0.
[21]	ECC_sel	If set, access the ECC portion of the L2-cache line based on VA[5]. If not, access the data portion of the line based on VA[5:3].
[20:19]	way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way 1, 2'b10: Way 2, 2'b11: Way 3.
[18:6]	index	A 13-bit index (PA[18:6]) that selects an L2-cache entry.
[5:3]	xw_offset	A 3-bit double-word offset. VA[5:3] = 000 selects L2_data[511:448] VA[5:3] = 001 selects L2_data[447:384] VA[5:3] = 010 selects L2_data[383:320] VA[5:3] = 011 selects L2_data[319:256] VA[5:3] = 100 selects L2_data[255:192] VA[5:3] = 101 selects L2_data[191:128] VA[5:3] = 110 selects L2_data[127:64] VA[5:3] = 111 selects L2_data[63:0]
[2:0]	Mandatory value	Should be 0.

During a write to ASI_L2CACHE_DATA with ECC_sel = 0, ECC check bits will not be written because there is no ECC generation circuitry in the L2 data write path and the data portion will be written based on the address/data format. During an ASI read with ECC_sel = 0, the data portion will be read out based on the address format.

During an ASI write with ECC_sel = 1, ECC check bits will be written based on the ASI address/data format. During an ASI read with ECC_sel = 1, the ECC check bits will be read out based on the address format. VA[4:3] in the address format is not used when ECC_sel = 1, because ECC is 16B boundary and ASI_L2CACHE_DATA returns the ECC for both the high and the low 16B data based on VA[5].

The data format for L2-cache data diagnostic access when ECC_sel = 0 and ECC_sel = 1 is shown in TABLE 3-62 and TABLE 3-63, respectively.

TABLE 3-62 L2-cache Data Access Data Format when ECC_sel = 0

Bits	Field	Description
[63:0]	L2_data	64-bit L2-cache data of a given index+way+xw_offset.

TABLE 3-63 L2-cache data access Data Format when ECC_sel = 1

Bits	Field	Description
[17:9]	L2_ecc_hi	When VA[5] = 0, L2_ecc_hi corresponds to 9-bit ecc for L2 data[511:384]; when VA[5] = 1, L2_ecc_hi corresponds to 9-bit ecc for L2 data[255:128].
[8:0]	L2_ecc_lo	When VA[5] = 0, L2_ecc_lo corresponds to 9-bit ecc for L2 data[383:256]; when VA[5] = 1, L2_ecc_lo corresponds to 9-bit ecc for L2 data[127:0].

Note – If L2-cache data diagnostic access encounters a L2-cache data CE, the returned data will not be corrected, and raw L2-cache data will be returned regardless of L2_data_ecc_en in L2-cache control register (ASI $6D_{16}$) is set or not.

For the one entry data buffer used in the L2-cache off mode, no ASI diagnostic access is supported. The L2-cache data array can be accessed as usual in the L2-cache off mode. However, the returned value is not guaranteed to be correct since the SRAM can be defective and this may be the reason to turn off the L2-cache.

3.12 L3-cache Diagnostic & Control Accesses

Separate ASIs are provided for reading and writing the L3-cache tag and data SRAMs as well as the L3-cache control register. This section describes the following L3-cache accesses:

- L3-cache Control Register
- L3-cache data/ECC field diagnostic accesses
- L3-cache tag/state/LRU/ECC field diagnostics accesses
- L3-cache SRAM mapping

3.12.1 L3-cache Control Register

ASI 75_{16}, Read and Write, Shared by both logical processors

VA = 0_{16}

Name: `ASI_L3CACHE_CONTROL`

The data format for L3-cache control register is shown in TABLE 3-64.

TABLE 3-64 L3-cache Control Register Access Data Format *(1 of 3)*

Bits	Field	Description
[63:45]	Mandatory value	Should be 0.
[44:40]	siu_data_mode	SIU data interface mode; required to be set for different system clock ratio and L3-cache mode based on *Secondary L3-cache Control Register* on page 69. After hard reset, these bits will be read as $0F_{16}$.
[39]	ET_off	If set, disables on-chip L3-cache tag SRAM for debugging. After hard reset, this bit will be read as 1'b0.
[33]	EC_PAR_force_LHS	If set, flips the least significant bit of address to the left hand side SRAM DIMM of L3 -cache. After hard reset, this bit will be read as 1'b0.
[32]	EC_PAR_force_RHS	If set, flips the least significant bit of address to the right hand side SRAM DIMM of L3-cache. After hard reset, this bit will be read as 1'b0.
[31]	EC_RW_grp_en	If set, enables the read bypassing write logic in L3-cache data access. After hard reset, this bit will be read as 1'b0.
[30]	EC_split_en	If set, enables L2-cache writebacks originating from logical processor 0 to write into way 0 and way 1 of L3-cache and L2-cache writebacks originating from logical processor 1 to write into way 2 and way 3 of L3-cache. After hard reset, this bit will be read as 1'b0.
[26]	pf2_RTO_en	If set, enables sending RTO on PREFETCH, fcn=2, 3, 22, 23. After hard reset, this bit will be read as 1'b0.
[25]	ET_ECC_en	If set, enables ECC checking on L3-cache tag bits. If disabled, then there will never be ECC errors due to L3-cache tag access. However, ECC generation and write to L3-cache tag ECC array will still occur in correct manner. After hard reset, this bit will be read as 1'b0.
[24]	EC_assoc	L3-cache data SRAM architecture. This bit is hardwired to 1'b0 in the UltraSPARC IV+ processor. 1'b0 = Late Write SRAM (Hard_POR value) 1'b1 = Late Select SRAM (not supported)
[38], [23]	addr_setup	Address setup cycles prior to SRAM rising clock edge. [38][23] == 00 = 1 cycle (not supported) [38][23] == 01 = 2 cycles (POR value) [38][23] == 10 = 3 cycle [38][23] == 11 = unused

TABLE 3-64 **L3-cache Control Register Access Data Format** *(2 of 3)*

Bits	Field	Description
[37], [29], [22:21]	trace_out	Address trace out cycles [37][29][22:21] == 0000 = 3 cycles (not supported) [37][29][22:21] == 0001 = 4 cycles (not supported) [37][29][22:21] == 0010 = 5 cycles [37][29][22:21] == 0011 = 6 cycles [37][29][22:21] == 0100 = 7 cycles [37][29][22:21] == 0101 = 8 cycles (POR value) [37][29][22:21] == 0110 = 9 cycles [[37]29][22:21] == 0111 = 10 cycles [37][29][22:21] == 1000 = 11 cycles [37][29][22:21] == 1001 = 12 cycles [37][29][22:21] == 1010 = unused [37][29][22:21] == 1011 = unused [37][29][22:21] == 1100 = unused [37][29][22:21] == 1101 = unused [37][29][22:21] == 1110 = unused [37][29][22:21] == 1111 = unused
[36], [19:17]	trace_in	Data trace in cycles [36][19:17] == 0000 = 2 cycles (not supported) [36][19:17] == 0001 = 4 cycles (not supported) [36][19:17] == 0010 = 5 cycles [36][19:17] == 0011 = 6 cycles [36][19:17] == 0100 = 3 cycles (not supported) [36][19:17] == 0101 = 7 cycles [36][19:17] == 0110 = 8 cycles (POR value) [36][19:17] == 0111 = 9 cycles [36][19:17] == 1000 = 10 cycles [36][19:17] == 1001 = 11 cycles [36][19:17] == 1010 = 12 cycles [36][19:17] == 1011 = unused [36][19:17] == 1100 = unused [36][19:17] == 1101 = unused [36][19:17] == 1110 = unused [36][19:17] == 1111 = unused
[35], [16]	EC_turn_rw	L3-cache data turnaround cycle (read-to-write) [35][16] == 00 = 1 SRAM cycle (not supported) [35][16] == 01 = 2 SRAM cycles (POR value) [35][16] == 10 = 3 SRAM cycles [35][16] == 11 = unused

TABLE 3-64 **L3-cache Control Register Access Data Format** *(3 of 3)*

Bits	Field	Description
[28], [14:13]	EC_size	L3-cache size. The size specified here affects the size of the EC_addr field in the L3-cache Data Register. These bits are hardwired to 3'b100 in the UltraSPARC IV+ processor. [28][14:13] == 000 = 1MB L3-cache size (not supported)[28][14:13] == 001 = 4MB L3-cache size (not supported)[28][14:13] == 010 = 8 MB L3-cache size (not supported)[28][14:13] == 011 = 16 MB L3-cache size (not supported) [28][14:13] == 100 = 32 MB L3-cache size (hardwired to 3'b100) [28][14:13] == 101 = unused [28][14:13] == 110 = unused [28][14:13] == 111 = unused
[34], [27], [12:11]	EC_clock	L3-cache clock ratio. [34][27][12:11] == 0000 selects 3:1 L3-cache clock ratio.(not supported) [34][27][12:11] == 0001 selects 4:1 L3-cache clock ratio.(not supported) [34][27][12:11] == 0010 selects 5:1 L3-cache clock ratio. [34][27][12:11] == 0011 selects 6:1 L3-cache clock ratio. [34][27][12:11] == 0100 selects 7:1 L3-cache clock ratio. [34][27][12:11] == 0101 selects 8:1 L3-cache clock ratio. (POR value) [34][27][12:11] == 0110 selects 9:1 L3-cache clock ratio. [34][27][12:11] == 0111 selects 10:1 L3-cache clock ratio. [34][27][12:11] == 1000 selects 11:1 L3-cache clock ratio. [34][27][12:11] == 1001 selects 12:1 L3-cache clock ratio. [34][27][12:11] == 1010 unused [34][27][12:11] == 1011 unused. [34][27][12:11] == 1100 unused [34][27][12:11] == 1101 unused [34][27][12:11] == 1110 unused [34][27][12:11] == 1111 unused
[20]	*Reserved*	This bit is hardwired to 1'b0 in the UltraSPARC IV+ processor.
[15]	*Reserved*	This bit is hardwired to 1'b0 in the UltraSPARC IV+ processor.
[10]	EC_ECC_en	If set, enables ECC checking on L3-cache data bits. After hard reset, this bit will be read as 1'b0.
[9]	EC_ECC_force	If set, forces EC_check[8:0] onto L3-cache data ECC bits. After hard reset, this bit will be read as 1'b0.
[8:0]	EC_check	ECC check vector to force onto ECC bits. After hard reset, this bit will be read as 9'b0.

Note – Bit[63:45], bit[20], and bit[15] of the L3-cache control register are treated as don't care for ASI write operations and read as zero for ASI read operations.

Hardware will automatically use the default (POR) value if an un-supported value is programmed in L3-cache control register.

3.12.2 Secondary L3-cache Control Register

ASI 75_{16}, Read and Write, Shared by both logical processors

VA = 8_{16},

The UltraSPARC IV+ processor does not implement this register because low-power mode is not supported in the UltraSPARC IV+ processor. For stores, it is treated as a NOP. For loads, it is the same as ASI = 75_{16}, VA = 0_{16}.

3.12.3 L3-cache Data/ECC Fields Diagnostics Accesses

ASI 76_{16} (WRITING) / ASI $7E_{16}$ (READING), Shared by both logical processors

VA[63:25] = 0

VA[24:23] = EC_way

VA[22:5] = EC_addr

VA[4:0] = 0

Name: `ASI_L3CACHE_W` / `ASI_L3CACHE_R`

The address format for L3-cache data diagnostic access is described below in TABLE 3-65.

TABLE 3-65 L3-cache data diagnostic access

Bits	Field	Description
[63:25]	Mandatory value	Should be 0.
[24:23]	EC_way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way 1, 2'b10: Way 2, 2'b11: Way 3.
[22:5]	EC_addr	The size of this field is determined by the EC_size field specified in the L3-cache control register. The UltraSPARC IV+ processor supports a 32MB L3-cache, and therefore, uses an 18-bit index (PA[22:5]) plus a 2-bit way [24:23] to read/write a 32-byte field from the L3-cache to/from the L3-cache Data Staging registers (discussed in *L3-cache Data Staging Registers* on page 70).
[4:0]	Mandatory value	Should be 0.

Note – The off-chip L3-cache data SRAM ASI accesses can take place regardless of the on-chip L3-cache tag SRAM is on or off.

3.12.4 L3-cache Data Staging Registers

ASI (74_{16}), Read and Write, Shared by both logical processors

VA[63:6] = 0

VA[5:3] = Staging Register Number

VA[2:0] = 0

Name: `ASI_L3CACHE_DATA`

The address format for L3-cache data staging register access is shown in TABLE 3-66.

TABLE 3-66 L3-cache data staging register access

Bits	Field	Description
[63:6]	Mandatory value	Should be 0.
[5:3]	data_register_number	Selects one of five staging registers: 000: EC_data_3 001: EC_data_2 010: EC_data_1 011: EC_data_0 100: EC_data_ECC 101: unused 110: unused 111: unused
[2:0]	Mandatory value	Should be 0.

The data format for the L3-cache data staging register data access is shown in TABLE 3-67.

TABLE 3-67 L3-cache data staging register data access

Bits	Field	Description
[63:0]	EC_data_N	64-bit staged L3-cache data EC_data_0[63:0] corresponds to VA[5:3] = 011 EC_data_1[127:64] corresponds toVA[5:3] = 010 EC_data_2[191:128] corresponds to VA[5:3] = 001 EC_data_3[255:192] corresponds to VA[5:3] = 000

The data format for the L3-cache data staging register ECC access is shown in TABLE 3-68.

TABLE 3-68 L3-cache data staging register ECC access

Bits	Field	Description
[63:18]	*Reserved*	Reserved for future implementation.
[17:9]	EC_data_ECC_hi	9 bit L3-cache data ECC field for the high 16-byte(PA[4] = 0, L3 data[255:128]).
[8:0]	EC_data_ECC_lo	9 bit L3-cache data ECC field for the low 16-byte(PA[4] = 1, L3 data[127:0]).

Note – L3-cache data staging registers are shared by both logical processors. When a logical processor accesses L3-cache data staging registers, software must guarantee the other logical processor will not access the L3-cache data staging registers. It can be done by parking or disabling the other logical processor before accessing L3-cache data staging registers. If both logical processors access the L3-cache data staging registers, the behavior is undefined.

If L3-cache data diagnostic access encounters a L3-cache data CE, the returned data will be corrected if EC_ECC_en in L3-cache control register (ASI 75_{16}) is asserted, otherwise the raw L3-cache data will be returned.

3.12.5 Direct L3-cache Tag Diagnostics Access and Displacement Flush

ASI $4E_{16}$, Read and Write, Shared by both logical processors

VA[63:27] = 0
VA[26] = disp_flush
VA[25] = hw_gen_ecc
VA[24:23] = EC_way

VA[22:6] = EC_tag_addr
VA[5:0] = 0

Name: `ASI_L3CACHE_TAG`

The address format for L3-cache tag diagnostic access is shown in TABLE 3-69.

TABLE 3-69 L3-cache tag diagnostic access

Bits	Field	Description
[63:27]	Mandatory value	Should be 0.
[26]	disp_flush	Specifies the type of access: 1'b1: displacement flush -- if the specified L3-cache line is dirty (i.e., state is equal to Modified or Owned or Owned Shared), write the line back to memory and invalidate the L3-cache line; if the line is clean (i.e., state is equal to Shared or Exclusive), update the state to Invalid state. The ASI data portion is unused when this bit is set. 1'b0: direct L3-cache tag access - read/write the tag, state, ECC, and LRU fields of the selected L3-cache line. In L3-cache tag SRAM on mode, if the line to be displacement flushed is in NA state, it will not be written out to the memory. The line remains in NA state in L3-cache. In L3-cache tag SRAM off mode, displacement flush is treated as a NOP. **Note:** For L3-cache displacement flush, use only LDXA (STXA has NOP behavior). Since L3-cache will return garbage data to the MS pipeline, it is recommended to use "ldxa [reg_address]ASI_L3CACHE_TAG, %g0" instruction format.
[25]	hw_gen_ecc	On read, this bit is don't care. On write, if the bit is set, write the 9-bit ECC field of the selected L3-cache tag entry with the value calculated by the hardware ECC generation logic inside the L3 tag SRAM. When this bit is not set, the ECC generated by hardware ECC generation logic is ignored, what is specified in the ECC field of the ASI data register will be written into the ECC field of the selected L3-cache line.
[24:23]	EC_way	A 2-bit entry that selects an associative way (4-way associative). 2'b00: Way 0, 2'b01: Way 1, 2'b10: Way 2, 2'b11: Way 3.
[22:6]	EC_tag_addr	Specifies the L3-cache tag set (PA[22:6]) for the read/write access.
[5:0]	Mandatory value	Should be 0.

The data format for L3-cache tag diagnostic access when disp_flush = 0 is shown in TABLE 3-70.

TABLE 3-70 L3-cache Tag Diagnostic Access

Bits	Field	Description
[63:44]	Mandatory value	Should be 0.
[43:24]	EC_Tag	A 20-bit tag (PA[42:23])
[23:15]	Mandatory value	Should be 0.
[14:6]	ECC	A 9-bit ECC entry that protects L3-cache tag and state. The ECC bits protect all 4-ways of a given L3-cache index.
[5:3]	LRU	A 3-bit entry that records the access history of the 4 ways of a given L3-cache index. If EC_split_en in the L3-cache control register (ASI 7516) is not set, the LRU is as described below. The LRU-pointed way will not be picked for replacement if "NA" is the state. LRU[2:0] = 000way0 is the LRU LRU[2:0] = 001way1 is the LRU LRU[2:0] = 010way0 is the LRU LRU[2:0] = 011way1 is the LRU LRU[2:0] = 100way2 is the LRU LRU[2:0] = 101way2 is the LRU LRU[2:0] = 110way3 is the LRU LRU[2:0] = 111way3 is the LRU If EC_split_en is set, the LRU is as described below. LRU[2] is ignored and logical processor ID of the logical processor which triggers the request is used. {LP, LRU[1:0]} = 000way0 is the LRU {LP, LRU[1:0]} = 001way1 is the LRU {LP, LRU[1:0]} = 010way0 is the LRU {LP, LRU[1:0]} = 011way1 is the LRU {LP, LRU[1:0]} = 100way2 is the LRU {LP, LRU[1:0]} = 101way2 is the LRU {LP, LRU[1:0]} = 110way3 is the LRU {LP, LRU[1:0]} = 111way3 is the LRU
[2:0]	EC_state	3-bit L3-cache state field. The bits are encoded as follows: state[2:0] = 000 Invalid state[2:0] = 001 Shared state[2:0] = 010 Exclusive state[2:0] = 011 Owner state[2:0] = 100 Modified state[2:0] = 101 NA (Not Available) (see *Notes on "NA" Cache State* on page 64) state[2:0] = 110 Owner/Shared state[2:0] = 111 Reserved

---

Note – Bit[63:44] and bit[23:15] of `ASI_L3CACHE_TAG` data are treated as don't care for ASI write operations and read as zero for ASI read operations.

On reads to `ASI_L3CACHE_TAG`, regardless of the hw_ecc_gen_en bit, the tag and the state of the specified entry (given by the index and the way) will be read out along with the ECC and the LRU of the corresponding index. A common ECC and LRU is shared by all 4 ways of an index.

On writes to `ASI_L3CACHE_TAG`, the tag and the state will always be written into the specified entry. LRU will always be written into the specified index. For the ECC field, if hw_ecc_gen_en is set, the ECC field of the data register is don't care. If the hw_ecc_gen_en is not set, the ECC specified in the data register will be written into the index. The hardware generated ECC will be ignored in that case.

Note – The L3-cache tag array can be accessed as usual through ASI when ET_off is set in the L3-cache control register (ASI 75_{16}). However, the returned value is not guaranteed to be correct since the SRAM can be defective and this may be one reason to turn off the L3-cache.

3.12.5.1 Notes on L3-cache Tag ECC

- The ECC for L3-cache tag is generated using a 128-bit ECC generator based on Hsiao's SEC-DED-S4ED code:
  ECC[8:0] = 128bit_ECC_generator(*l3tag_ecc_din*)

  where,

  l3tag_ecc_din[127:0] = {18'b0, tag2[19], tag0[19], tag2[18], tag0[18], ..., tag2[1], tag0[1], tag2[0], tag0[0], state2[2], state0[2], state2[1], state0[1], state2[0], state0[0], 18'b0, tag3[19], tag1[19], tag3[18], tag1[18], ..., tag3[1], tag1[1], tag3[0], tag1[0], state3[2], state1[2], state3[1], state1[1], state3[0], state1[0] }.
- The ECC value of a zero L3-cache tag is also zero. Thus, after `ASI_SRAM_FAST_INIT_SHARED` (`STXA` $3F_{16}$), the ECC value is correct and all lines will be in INVALID state. L3-cache tag ECC checking is carried out regardless of the L3-cache line is valid or not.
- If L3-cache tag diagnostic access encounters an L3-cache tag CE, the returned data will not be corrected, and raw L3-cache tag data will be returned, regardless of ET_ECC_en in the L3-cache control register is set or not.
- If there is an ASI write request to L3-cache tag (does not include displacement flush) and the ASI write request wins L2L3 pipeline arbitration, it will be sent to the L2L3 pipeline to access the L2 tag and L3 tag at the same time. Within 15 cycles, if there is a request (I-cache, D-cache, P-cache, SIU snoop, or SIU copyback request) to the same cache index following the ASI request, then the second request will get incorrect tag ECC data. It will not have any issues if two accesses are to the different index. To avoid this problem, the following procedures will be followed when software uses ASI write to L3-cache tag to inject L2-cache tag error.

3.12.5.2 Procedure For Writing `ASI_L3CACHE_TAG`:

1. Park the other logical processor.
2. Wait for the parking logical processor to be parked.
3. Turn off kernel pre-emption.
4. Block interrupts on this processor.
5. Displacement flush all 4 ways in L2-cache and L3-cache for the index to be error injected.
6. Load some data into the L3-cache.
7. Locate the data in the L3-cache and associated tag.
8. Read the L3-cache tag ECC (using ASI L3-cache tag read access).
9. Corrupt the tag ECC.
10. Store the tag ECC back (using ASI L3-cache tag write access).
11. Re-enable interrupts.
12. Unpark the other logical processor.

The reason to displacement flush all 4 ways in both L2-cache and L3-cache is to guarantee that foreign snoop will have no effect to the index during ASI L3-cache tag write access even if the hazard window exists in the hardware.

3.12.5.3 Notes on L3-cache LRU Bits

- The L3-cache LRU algorithm is described below:

For each 4-way set of L3 data blocks, a 3-bit structure is used to identify the least recently used way. Bit[2] tracks which one of the two 2-way sets (way3/2 is one group; way1/0 is the other) is the least recently used. Bit[1] tracks which way of the 2-way set (way3 and way2) is the least recently used. Bit[0] tracks which way of the other 2-way set (way1 and way0) is the least recently used. The LRU bits are reset to 3'b000 and updated based on the following scheme:

bit[2] = 1 if hit in way1 or way0;

bit[1] = 1 if hit in way2 or it remains 1 when not hit in way3;

bit[0] = 1 if hit in way0 or it remains 1 when not hit in way1

- LRU bits are not ECC protected.

3.12.6 L3-cache SRAM Mapping

The following hashing algorithm is used to determine which L3-cache lines are mapped to which SRAM blocks:

L3_SRAM_addr[24:5] = {*PA*[22:5], *way*[1:0]}

where, *PA* and *way* are the physical address and the way of the L3-cache line to be mapped and *L3_SRAM_addr* is the address of the L3-cache SRAM block to which the L3-cache line is mapped.

3.13 Summary of ASI Accesses in L2/L3 Off Mode

TABLE 3-71 summarizes the different ASI accesses to L2 and L3 tag/data SRAMs in the L2/L3 off mode.

TABLE 3-71 ASI access to shared SRAMs in L2/L3 off mode

SRAM	ASI Write	displacement flush	ASI Read	Tag Update
L2tag SRAM	the same as L2 on	NOP	same as L2 on	No
L2data SRAM	the same as L2 on	NOP	same as L2 on	N/A
L3tag SRAM	the same as L3 on	NOP	same as L3 on	No
L3data SRAM	the same as L3 on	NOP	same as L3 on	N/A

3.14 ASI SRAM Fast Init

To speed up cache and on-chip SRAM initialization, the UltraSPARC IV+ processor leverages the SRAM manufacturing hooks to initialize (zero out) these SRAM contents.

ASI 40_{16} (`ASI_SRAM_FAST_INIT`) will initialize all the SRAM structures that are associated with a particular logical processor - I-cache cluster, D-cache cluster, P-cache cluster, W-cache cluster, Branch Prediction Array, and TLBs. TLBs must be turned off before the `ASI_SRAM_FAST_INIT` is launched.

ASI $3F_{16}$ (`ASI_SRAM_FAST_INIT_SHARED`) will initialize all the SRAM structures that are shared between the two logical processors - L2-cache tag, L2-cache data, and L3-cache tag SRAMs.

FIGURE 3-1 shows how the D-MMU of a particular logical processor sends the initialization data to all 3 loops in parallel for ASI 40_{16} or sends the initialization data to the shared loop for ASI $3F_{16}$. Loop 2 will initialize the D-cache data, physical tag, snoop tag, microtag arrays. Loop 3 will initialize the three D-TLBs. Loop 1 will initialize the 2 I-TLBs as well as the I-cache physical tag,

snoop tag, valid/predict, Microtag arrays, etc. For the shared loop, D-MMU will continuously dispatch a total of 2^{22} ASI write requests to flush all the entries of L2-cache tag, L2-cache data, and the L3-cache tag SRAM structures on the ASI chain in a pipe lined fashion.

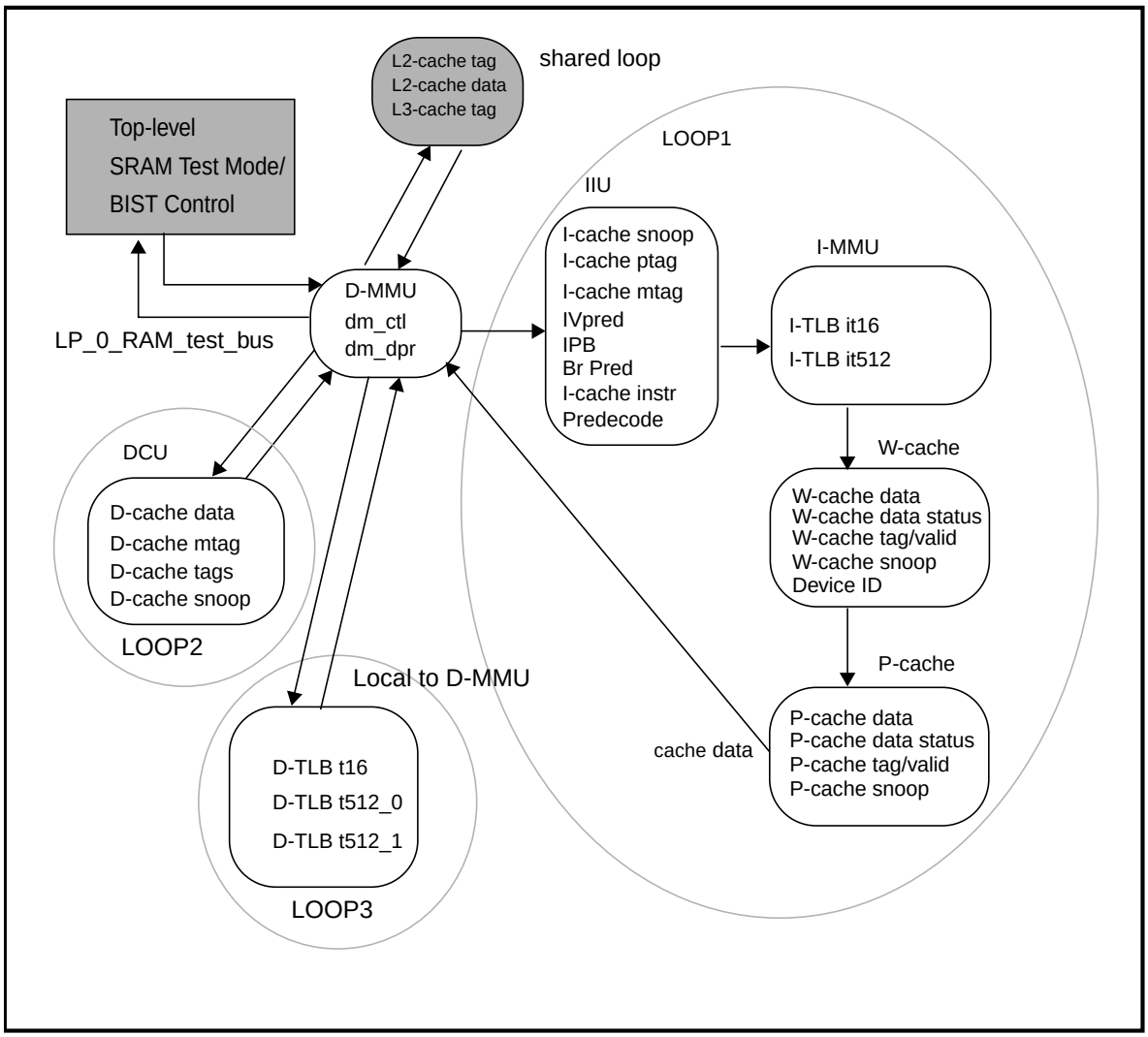

FIGURE 3-1 Fast init ASI 40_{16} Goes Through Three loops in Pipeline Fashion

For the shared loop, since L3-cache tag SRAM has the maximum entries (2^{19}), D-MMU sends 2^{22} SRAM addresses(VA[3] to VA[24]) to initialize the cache. The address is incremented every cycle in regular SRAM test mode or every other cycle in BIST control mode. When there are no more ASI write request to be issued, the state machine enters a wait state where it waits for the last return signal from the chain to arrive, which is exactly a fixed 21-cycle delay. Once the 21st cycle is reached, the STQ is signaled to retire the store ASI $3F_{16}$ instruction.

Therefore, the total cycles for the ASI $3F_{16}$ execution is at most 2^{22} x 2 + 21 = 8388629 cycles, or about 0.007 s*econds* for a 1200MHz UltraSPARC IV+ processor, or 0.004 *seconds* for a 2400 MHz UltraSPARC IV+ processor. Thus, all-cache structures are initialized in 10 milliseconds.

3.14.1 ASI_SRAM_FAST_INIT Definition

ASI 40_{16}, Write only, Per logical processor

VA[63:0] = 0_{16}

Name: ASI_SRAM_FAST_INIT

Description:
This ASI is used *only* with the STXA instruction to quickly initialize almost all on-chip SRAM structures that are associated with each logical processor. A single STXA ASI 40_{16} will cause the store data value to be written to *all* on-chip SRAM entries of (within one logical processor):

- I-cache: Inst/Predecode, Physical Tag, Snoop Tag, Valid/Predict, Microtag arrays
- IPB: Inst/Predecode, Tag, Valid
- BP: Branch Predictor Array, Branch Target Buffer
- D-cache: Data, Physical Tag, Snoop Tag, Microtag arrays
- P-cache: Data, Status Data, Tag/Valid, Snoop Tag arrays
- W-cache: Data, Status Data, Tag/Valid, Snoop Tag arrays
- D-TLB: t16, t512_0, t512_1 arrays
- I-TLB: it16, it512 arrays

Usage:
`stxa` %g0, [%g0]ASI_SRAM_FAST_INIT.

The 64-bit store data must be zero. Initializing the SRAMs to non-zero value could have unwanted side effects. This STXA instruction must be surrounded (preceded and followed) by `MEMBAR #Sync` to guarantee that:

- All Sun Fireplane Interconnect transactions have completed before the SRAM initialization begins.
- SRAM initialization fully completes before proceeding.

Note – During the SRAM initialization, caches and TLBs are considered unusable and incoherent. The ASI_SRAM_FAST_INIT instruction should be located in non-cacheable address space.

3.14.1.1 Code Sequence for ASI_SRAM_FAST_INIT

In the UltraSPARC IV+ processor, ASI_SRAM_FAST_INIT copies the content of the ASI_IMMU_TAG_ACCESS register to I-MMU tag assuming this register contains functional content. However, if ASI_SRAM_FAST_INIT operation were to be invoked right after a reset, the uninitialized content of ASI_IMMU_TAG_ACCESS register will cause the I-TLB tag to not initialize. As a result, the expected result would not have been accomplished.

The UltraSPARC IV+ processor also requires that there are no outstanding requests of any kind before ASI_SRAM_FAST_INIT is invoked. In order to guarantee that there are no outstanding requests before ASI_SRAM_FAST_INIT and to avoid an I-MMU tag initialization problem, the following code sequence must be executed in atomic form:

```
set                 30_16, %g1
membar #Sync        stxa
%g0,                [%g1]ASI_IMMU_TAG_ACCESS
membar #Sync
.align 16
nop
```

```
flush               %g0
stxa                %g0, [%g0]ASI_SRAM_FAST_INIT
membar  #Sync
```

Note – The requirement of executing this code sequence in atomic form is to ensure that there would not be any trap handler sneaking in between to initialize the ASI_IMMU_TAG_ACCESS register to other unwanted value.

After ASI_SRAM_FAST_INIT, D-cache and I-cache microtag arrays have to be initialized with unique values among 4 same-index entries from different banks.

Traps:
data_access_exception trap for ASI 40_{16} use other than with STXA instruction.

3.14.2 ASI_SRAM_FAST_INIT_Shared Definition

ASI $3F_{16}$, Write only, Shared by both logical processors

VA[63:0]=0_{16}

Name: ASI_SRAM_FAST_INIT_SHARED

Description:
This ASI is used *only* with STXA instruction to quickly initialize the on-chip SRAM structures that are shared between the two logical processors in the CMT device. A single STXA ASI $3F_{16}$ will cause the store data value to be written in *all* shared on-chip SRAM entries of:

L2-cache: Data and Tag arrays.

L3-cache: Tag array

Usage:
stxa %g0,[%g0]ASI_SRAM_FAST_INIT_SHARED

The 64-bit store data must be zero. Initializing the SRAMs to non-zero value could have unwanted side effects. This STXA instruction must be surrounded (preceded and followed) by MEMBAR #Sync to guarantee that:

- All Sun Fireplane Interconnect transactions have completed before the SRAM initialization begins.
- SRAM initialization fully completes before proceeding.

Note – Only one logical processor should issue ASI_SRAM_FAST_INIT_SHARED. The other logical processor must be parked or disabled.

During the SRAM initialization, caches and TLBs are considered unusable and incoherent. The ASI_SRAM_FAST_INIT_SHARED instruction should be located in non-cacheable address space.

ASI_SRAM_FAST_INIT_SHARED will be issued at the default L2L3 arbitration throughput, which is once every two cycles regardless of the L2L3arb_single_issue_en in L2-cache control register (ASI $6D_{16}$) or WE in DCUCR (ASI 45_{16}).

Traps:
data_access_exception trap for ASI $3F_{16}$ use other than with STXA instruction.

3.15 OBP Backward Compatibility/Incompatibility

TABLE 3-72 summarizes the UltraSPARC IV+ processor OBP (Open Boot PROM) backward compatibility list.

TABLE 3-72 The UltraSPARC IV+ processor OBP Backward Compatibility List *(1 of 2)*

No.	New Feature	The UltraSPARC IV+ processor Bits	Hard _POR State	The UltraSPARC IV+ processor Behavior on UltraSPARC OBP/ Software
1.	ASI_L2CACHE_CTRL ($6D_{16}$)	Queue_timeout _disable	1'b0	The logic to detect the progress of a queue is disabled.
		Queue_timeout	3'b111	Queue timeout period is set at maximum value of 2M system cycles.
		L2_off	1'b0	L2-cache is enabled.
		Retry_disable	1'b0	The logic to measure request queue progress and put L2L3 arbiter in single issue mode is enabled.
		Retry_debug _counter	2'b00	The logic in L2L3 arbiter will count 1024 retries before entering the single-issue mode.
		L2L3arb_single _issue_frequency	1'b0	If L2L3 arbiter is in single-issue mode, the dispatch rate is one per 16 cycles.
		L2L3arb_single _issue_en	1'b0	L2L3 arbiter will dispatch a request every other cycle.
		L2_split_en	1'b0	All four ways in L2-cache can be used for replacement triggered by either logical processor.
		L2_data_ecc_en	1'b0	No ECC checking on L2-cache data bits.
		L2_tag_ecc_en	1'b0	Default to L2-cache Tag ECC checking disabled; NO error/ trap ever generated due to L2-cache Tag access.

TABLE 3-72 The UltraSPARC IV+ processor OBP Backward Compatibility List *(2 of 2)*

No.	New Feature	The UltraSPARC IV+ processor Bits	Hard _POR State	The UltraSPARC IV+ processor Behavior on UltraSPARC OBP/ Software
2.	ASI_L3CACHE_CTRL (75_{16})	siu_data_mode	5’b01111	Default to 8-8-8 L3-cache mode with 8:1 Sun Fireplane Interconnect clock ratio.
		ET_off	1’b0	The on-chip L3-cache tag RAM is enabled
		EC_PAR_force _LHS	1’b0	No bit flip on the address to the left hand side L3-cache SRAM DIMM.
		EC_PAR_force _RHS	1’b0	No bit flip on the address to the right hand side L3-cache SRAM DIMM.
		EC_RW_grp_en	1’b0	No read-bypass-write feature in L3-cache access.
		EC_split_en	1’b0	All four ways in L3-cache can be used for replacement triggered by either logical processor.
		pf2_RTO_en	1’b0	No prefetch request can send RTO on the bus.
		ET_ECC_en	1’b0	Default to L3-cache Tag ECC checking disabled -- NO error/ trap ever generated due to L3-cache Tag access.
		EC_assoc	1’b0	Late write SRAM; hardwired to 1’b0.
		addr_setup	2’b01	Default to 2 cycle SRAM addr setup w.r.t. SRAM clock.
		trace_out	4’b0101	Default to 8 cycles (8-8-8 L3-cache mode).
		trace_in	4’b0110	Default to 8 cycles (8-8-8 L3-cache mode).
		EC_turn_rw	2’b01	Default to 2 SRAM cycles between read-]write.
		EC_early	1’b0	Late write SRAM.(default); hardwired to 1’b0.
		EC_size	3’b100	Default too 32MB L3-cache size; hardwired to 3’b100.
		EC_clock	4’b0101	Default to 8:1 L3-cache clock ratio (8-8-8 L3-cache mode).
		EC_ECC_en	1’b0	No ECC checking on L3-cache data bits.
		EC_ECC_force	1’b0	ECC bits for writing are not picked from the L3-cache control register.
		EC_check	9’b0	ECC bits to be forced onto L3-cache data bus.

4

Reset and RED_state

This chapter defines and describes the RED_state (Reset, Error, and Debug state) in the following sections:

Chapter Topics

In the UltraSPARC IV+ processor, RED_state, externally initiated reset (XIR), watchdog reset (WDR), and software-initiated reset (SIR) can apply to one logical processor but not the other, while a hard power-on reset (hard POR) or a system reset (soft POR) always applies to both logical processors.

4.1 RED_state Characteristics

A reset or trap that sets PSTATE.RED (including a trap occurring while in RED_state) clears the Data Cache Unit Control Register, including the enable bits for the I-cache, D-cache, I-MMU, D-MMU, and the virtual and physical watchpoints. The characteristics of RED_state include the following:

- The default access in RED_state is non-cacheable, so there must be non-cacheable scratch memory somewhere in the system.
- The D-cache, watchpoints, and D-MMU can be enabled by software in RED_state, but any trap will disable them again.
- The I-MMU and consequently the I-cache are always disabled in RED_state. Disabling overrides the enable bits in the DCU control register.
- When PSTATE.RED is explicitly set by a software write, there are no side effects other than that the I-MMU is disabled. Software must create the appropriate state itself.
- A trap when TL = MAXTL - 1 immediately brings the processor into RED_state. In addition, any trap that occurs while TL = MAXTL immediately brings the processor into *error_state*. Upon *error_state* entry, the processor automatically recovers through watchdog reset into RED_state.
- A trap to *error_state* immediately triggers watchdog reset.

- A SIR instruction generates a *software_initiated_reset (SIR)* trap on the corresponding logical processor.
- Trapping to software_initiated_reset causes an *SIR* trap on the corresponding logical processor and brings the logical processor into RED_state.
- The External Reset pin generates an *externally_initiated_reset (XIR)* trap which is used for system debug or Sun Fireplane Interconnect transactions.
- The caches continue to snoop and maintain coherence if DVMA or other processors are still issuing cacheable accesses.

Note – Exiting RED_state by setting PSTATE.RED to 0 in the delay slot of a `JMPL` is not recommended. A noncacheable instruction prefetch can be made to the `JMPL` target, which may be in a cacheable memory area. This condition could result in a bus error on some systems and cause an *instruction_access_error* trap. The trap can be masked by setting the NCEEN bit in the ESTATE_ERR_EN register to 0, but this approach will mask all non-correctable error checking. Exiting RED_state with `DONE` or `RETRY` avoids the problem.

4.2 Resets

The reset priorities ranging from highest to lowest are:

- power-on resets (POR, hard or soft)
- externally initiated reset (XIR)
- watchdog reset (WDR)
- software-initiated reset (SIR).

4.2.1 Hard Power-on Reset (Hard POR, Power-on Reset, Power OK Reset)

A hard power-on reset (hard POR) occurs when the Power OK Reset (POK) pin is activated and stays asserted until the processor is within its specified operating range. When the POK pin is active, all other resets and traps are ignored. Power-on reset has a trap type of 1 at physical address offset 20_{16}. Any pending external transactions are canceled.

After a hard power-on reset, the software must initialize processor certain values, please see TABLE 4-1. In particular, the valid and microtag bits in the I-cache, the valid and microtag bits in the D-cache, and all L2/L3-cache tags and data must be cleared before the caches are enabled. The I-MMU and D-MMU TLBs must be initialized by clearing the valid bits of all TLB entries (see the *UltraSPARC III Cu Processor User's Manual*). The P-cache valid bits must be cleared before any floating-point loads are executed.

The MCU refresh control register as well as the Sun Fireplane Interconnect configuration register must be initialized after a hard power-on reset.

In SSM (Scalable Shared Memory) systems, the microtags contained in memory must be initialized before any Sun Fireplane Interconnect transactions are generated.

Note – Executing a `DONE` or `RETRY` instruction when TSTATE is uninitialized after a POR can damage the processor. The POR boot code should initialize TSTATE[3:0], using wrpr writes, before any `DONE` or `RETRY` instructions are executed.

4.2.2 System Reset (Soft POR, Sun Fireplane Interconnect Reset, POR)

A system reset occurs when the Reset pin is activated. When the Reset pin is active, all other resets and traps are ignored. System reset has a trap type of 1 at physical address offset 20_{16}. Any pending external transactions are canceled. Memory refresh continues uninterrupted during a system reset.

4.2.3 Externally Initiated Reset (XIR)

An XIR steering register controls which logical processor(s) will receive the XIR reset. Please refer to the *UltraSPARC III Cu Processor User's Manual* for general information about XIR steering register.

XIR steering register:

ASI 41_{16}, VA[63:0] = 30_{16},

Name: `ASI_XIR_STEERING`

Access: Read/Write, Privileged access ASI register

See *XIR Steering Register (ASI_XIR_STEERING)* on page 21 for details about the XIR steering register in the UltraSPARC IV+ processor.

4.2.4 Watchdog Reset (WDR) and *error_state*

Please refer to the *UltraSPARC III Cu Processor User's Manual* for the description of the watchdog reset and error_state.

4.2.5 Software-Initiated Reset (SIR)

Please refer to the *UltraSPARC III Cu Processor User's Manual* for the description of software initiated reset.

4.3 RED_state Trap Vector

When an earlier UltraSPARC processor processes a reset or trap that enters RED_state, that processor takes a trap at an offset relative to the RED_state trap vector base address (RSTVaddr) at virtual address FFFF FFFF F000 0000_{16}. In the UltraSPARC IV+ processor, this base address passes through to physical address 7FF F000 0000_{16}.

4.4 Machine States After Reset

TABLE 4-1 shows the machine states created as a result of any reset or after RED_state is entered. RSTVaddr is often abbreviated as RSTV in the table.

TABLE 4-1 Machine State After Reset and RED_state *(1 of 5)*

Name	Fields	Hard_POR	System Reset	WDR	XIR	SIR	RED_state[1]
Integer registers		Unknown	Unchanged	Unchanged			
Floating-point registers		Unknown	Unchanged	Unchanged			
RSTVaddr value		VA = FFFF FFFF F000 0000_{16} PA = 7FF F000 0000_{16}					
PC nPC		RSTV | 20_{16} RSTV | 24_{16}	RSTV | 20_{16} RSTV | 24_{16}	RSTV | 40_{16} RSTV | 44_{16}	RSTV | 60_{16} RSTV | 64_{16}	RSTV | 80_{16} RSTV | 84_{16}	RSTV | $A0_{16}$ RSTV | $A4_{16}$
PSTATE	MM RED PEF AM PRIV IE AG CLE[2] TLE[3] IG MG	0 1 1 0 1 0 1 0 0 0 0	0 1 1 0 1 0 1 0 0 0 0	0 (TSO) 1(RED_state) 1 (FPU on) 0 (Full 64-bit address) 1 (Privileged mode) 0 (Disable interrupts) 1 (Alternate globals selected) PSTATE.TLE Unchanged 0 (Interrupt globals not selected) 0 (MMU globals not selected)			
TBA[63:15]		Unknown	Unchanged	Unchanged			
Y		Unknown	Unchanged	Unchanged			
PIL		Unknown	Unchanged	Unchanged			
CWP		Unknown	Unchanged	Unchanged except for register window traps			
TT[TL]		1	1	Unchanged	3	4	trap type
CCR		Unknown	Unchanged	Unchanged			
ASI		Unknown	Unchanged	Unchanged			
TL		MAXTL	MAXTL	Min(TL+1, MAXTL)			
TPC[TL] TNPC[TL]		Unknown Unknown	Unchanged Unchanged	PC nPC	PC & ~$1F_{16}$ nPC=PC+4	PC nPC	
TSTATE	CCR ASI PSTATE CWP PC nPC	Unknown Unknown Unknown Unknown Unknown Unknown	Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged	CCR ASI PSTATE CWP PC nPC			
TICK	NPT counter	1 Restart at 0	1 Restart at 0	Unchanged Count	Unchanged Restart at 0	Unchanged Count	
CANSAVE		Unknown	Unchanged	Unchanged			
CANRESTORE		Unknown	Unchanged	Unchanged			

TABLE 4-1 Machine State After Reset and RED_state *(2 of 5)*

Name	Fields	Hard_POR	System Reset	WDR	XIR	SIR	RED_state[1]
OTHERWIN		Unknown	Unchanged	Unchanged			
CLEANWIN		Unknown	Unchanged	Unchanged			
WSTATE	OTHER NORMAL	Unknown Unknown	Unchanged Unchanged	Unchanged Unchanged			
VER	MANUF IMPL MASK MAXTL MAXWIN	$003E_{16}$ 0019_{16} Mask dependent 5 7					
FSR	all	0	0	Unchanged			
FPRS	all	Unknown	Unchanged	Unchanged			
Non-SPARC V9 ASRs							
SOFTINT		Unknown	Unchanged	Unchanged			
TICK_COMPARE	INT_DIS TICK_CMPR	1 (off) 0	1 (off) 0	Unchanged Unchanged			
STICK	NPT counter	1 0	1 0	Unchanged Count			
STICK_COMPARE	INT_DIS TICK_CMPR	1 (off) 0	1 (off) 0	Unchanged Unchanged			
PCR	S1 S0 UT (trace user) ST (trace system) PRIV (priv access)	Unknown Unknown Unknown Unknown Unknown	Unchanged Unchanged Unchanged Unchanged Unchanged	Unchanged Unchanged Unchanged Unchanged Unchanged			
PIC	all	Unknown	Unknown	Unknown			
GSR	IM others	0 Unknown	0 Unchanged	Unchanged Unchanged			
DCR	all	0	0	Unchanged			
Non-SPARC V9 ASIs							
Sun Fireplane Interconnect Information	See *RED_state and Reset Values* on page 296						
DCUCR	WE all others	0(off) 0 (off)	0(off) 0 (off)	Unchanged 0 (off)			
L2-cache Control Register	Queue_timeout all others	16'h0e00 3'b111 0	Unchanged	Unchanged			

TABLE 4-1 Machine State After Reset and RED_state *(3 of 5)*

Name	Fields	Hard_POR	System Reset	WDR	XIR	SIR	RED_state[1]
L3- cache Control Register	siu_data_mode addr_setup trace_out trace_in EC_turn_rw EC_size EC_clock all others	45'h0f0038ad0800 5'b01111 2'b01 4'b0101 4'b0110 2'b01 3'b100 4'b0101 0	Unchanged	Unchanged			
INSTRUCTION_TRAP	all	0 (off)	0 (off)	Unchanged			
VA_WATCHPOINT		Unknown	Unchanged	Unchanged			
PA_WATCHPOINT		Unknown	Unchanged	Unchanged			
I-SFSR, D-SFSR	ASI FT E CTXT PRIV W OW (overwrite) FV NF TM	Unknown Unknown Unknown Unknown Unknown Unknown Unknown 0 Unknown Unknown	Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged 0 Unchanged Unchanged	Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged Unchanged			
DMMU_SFAR		Unknown	Unchanged	Unchanged			
INTR_DISPATCH_W	all	0	0	Unchanged			
INTR_DISPATCH_STATUS	all	0	Unchanged	Unchanged			
INTR_RECEIVE	BUSY	0	Unchanged	Unchanged			
	MID	Unknown	Unchanged	Unchanged			
ESTATE_ERR_EN	all	0 (all off)	Unchanged	Unchanged			
AFAR	PA	Unknown	Unchanged	Unchanged			
AFAR_2	PA	Unknown	Unchanged	Unchanged			
AFSR	all	0	Unchanged	Unchanged			
AFSR_2	all	0	Unchanged	Unchanged			
AFSR_EXT	all	0	Unchanged	Unchanged			
AFSR__EXT_2	all	0	Unchanged	Unchanged			
Rfr_CSR	all	Unknown	Unchanged	Unchanged			
Mem_Timing_CSR	all	Unknown	Unchanged	Unchanged			
Mem_Addr_Dec	all	Unknown	Unchanged	Unchanged			
Mem_Addr_Cntl	all	Unknown	Unchanged	Unchanged			
EMU_ACTIVITY_STATUS	all	0	0	Unchanged			

TABLE 4-1 Machine State After Reset and RED_state *(4 of 5)*

Name	Fields	Hard_POR	System Reset	WDR	XIR	SIR	RED_state[1]
Other Processor-Specific States							
Processor and external cache tags, microtags and data (includes data, instruction, prefetch, and write caches)		Unknown	Unchanged	Unchanged			
Cache snooping		Enabled					
Instruction Prefetch Buffer (IPB)	data, tag Valid bits	Unchanged Unchanged	Unchanged 0	Unchanged Unchanged			
Branch Target Buffer (BTB) data		Unchanged					0
Instruction Queue		Empty					
Store Queue		Empty	Empty	Unchanged			
I-TLB, D-TLB	Mappings in #2 (2-way set-associative) Mappings in #0 (fully set-associative) E[4] bit NC[5] bit	Unknown Unknown 1 1	Unchanged Unknown and invalid 1 1	Unchanged Unchanged 1 1			
CMT ASI extensions, SHARED Registers							
ASI_CORE_AVAILABLE		Predefined value set by hardware (‘1’ for each implemented logical processor,‘0’ for other bits)					
ASI_CORE_ENABLED		value of ASI_CORE_ENABLE at the time of deassertion of reset	value of ASI_CORE_ENABLE at the time of deassertion of reset	Unchanged			
ASI_CORE_ENABLE		‘1’ for each available logical processor ‘0’ for other bits (value can be overwritten by system controller during reset)	Unchanged (value can be overwritten by system controller during reset)	Unchanged			
ASI_XIR_STEERING		value of ASI_CORE_ENABLE at the time of deassertion of reset	value of ASI_CORE_ENABLE at the time of deassertion of reset	Unchanged			

TABLE 4-1 Machine State After Reset and RED_state *(5 of 5)*

Name	Fields	Hard_POR	System Reset	WDR	XIR	SIR	RED_state[1]
ASI_CMP_ERROR_STEERING		the LP_ID of the lowest numbered enabled logical processor (as indicated by ASI_CORE_ENABLE at time of deassertion of reset)	the LP_ID of the lowest numbered enabled logical processor (as indicated by ASI_CORE_ENABLE at time of deassertion of reset)	Unchanged			
ASI_CORE_RUNNING_RW		'1' in bit position of lowest available logical processor '0' in all other bits NOTE: if the system controller changes ASI_CORE_ENABLE during the reset and disables the lowest logical processor it must update this register	'1' in bit position of lowest enabled logical processor as specified by ASI_CORE_ENABLE before the reset '0' in all other bit positions NOTE: if the system controller changes ASI_CORE_ENABLE during the reset and disables the lowest logical processor it must update this register	Unchanged			
ASI_CORE_RUNNING_STATUS		Equal to the ASI_CORE_RUNNING register	Equal to the ASI_CORE_RUNNING register	Not affected			
CMT ASI extensions, PER-logical processor Registers							
ASI_CORE_ID	Number of LPs	Predefined value set by hardware					
	LP ID	Predefined value set by hardware					
ASI_INTR_ID		Unknown	Unchanged				
ASI_SCRATCHPAD_*n*_REG		Unknown	Unchanged				

1.Processor states are only updated according to the following table if RED_state is entered because of a reset or a trap. If RED_state is entered because the PSTATE.RED bit was explicitly set to 1, then software must create the appropriate states itself

2.CLE is Current Little-Endian

3.TLE is Trap Little-Endian

4. E is side effect bit

5.NC is Non-cacheable bit

5

Performance Instrumentation and Optimization

This chapter addresses the following sections:

Chapter Topics

5.1 Introduction to Optimization

Recompiling legacy code using a specifically designed compiler and setting the correct compile flags can significantly increase performance. There are several performance features provided by newer UltraSPARC processors that can only be taken advantage of by using modern compiler technology.

Optimization is aimed at increasing the supply of as many valid instructions as possible to the grouping logic and eventually to the functional units: ALUs, FGUs, branch units, load/store pipes just to name a few. One very important optimization technique is to prefetch data, avoiding the long latencies associated with cache misses.

5.2 Instruction Stream Issues

The following section addresses these issues:

- Instruction Alignment
- Instruction Cache Timing

- Executing Code Out of the Level 2 Cache
- Translation Lookaside Buffer (TLB) Misses
- Instruction Cache Utilization
- Handling of Control Transfer Instructions (CTI) Couples
- Mispredicted Branches
- Return Address Stack

5.2.1 Instruction Alignment

To ensure that the maximum number of instructions are fetched from an access, the instructions should be appropriately aligned.

5.2.1.1 Instruction Cache Organization

The I-cache is organized as a four-way set-associative cache, with each set containing a multiple of eight instruction lines. Depending on its address, for each line of 16 instructions, up to four instructions are sent to the instruction buffer. If the address points to one of the last three instructions in the line, only this last instruction and instructions (0-2) from the end of the line are selected. Consequently, on average for random accesses, 3.25 instructions are fetched from the I-cache. For sequential accesses, the fetching rate (four instructions per cycle) matches the consuming rate of the pipeline (up to four instructions per cycle).

5.2.1.2 Branch Target Alignment

Given the restriction mentioned above regarding the number of instructions fetched from an I-cache access, it is desirable to align branch targets so that enough instructions are fetched to match the number of instructions issued in the first group of the branch target. The following examples highlight the logic behind branch target alignment:

- If the compiler scheduler indicates that the target can be grouped with only one more instruction, the target should be placed anywhere in the line except in the last slot because only one instruction would be fetched in that case. It may be beneficial to fetch more instructions, if possible.
- If the target is accessed from more than one place, it should be aligned so that it accommodates the largest possible group (first five instructions of a line).
- If accesses to the I-cache are expected to miss, it may be desirable to align targets on a 64-byte boundary, or at least the front end of a block, so that four instructions are forwarded to the next stage. Such an alignment helps assure that the maximum number of instructions can be processed between cache misses, assuming that the code does not branch out of the sequence of instructions. In fact 64-byte alignment can help instruction prefetch.

In general, it is best to align for maximum fetching by aligning branch targets on four-instruction (16-byte) boundaries. This can help ensure that the fetch bandwidth matches the issue width, which is a maximum of four instructions in the case of the UltraSPARC processor.

5.2.1.3 Branch Optimization

The UltraSPARC processors favor branch not taken conditionals. Regardless of this preference, the instruction issue remains the same and the fetch is optimized.

5.2.1.4 Impact of the Delay Slot on Instruction Fetch

Most Control Transfer Instructions (CTIs) are actually *delayed*. Control transfer takes effect one instruction after the actual CTI. The intervening delay is called the *delay slot*. The instruction following the branch, or after CTI, is always executed regardless of where the CTI directs execution (unless annulling is used). If the last instruction of a line is a branch, the next sequential line in the I-cache must be fetched even if the branch predicted is taken because the delay slot must be sent to the grouping logic. This line fetch leads to inefficient fetches because an entire L2-cache access must be dedicated to fetching the missing delay slot. Therefore do not place delayed CTIs at the end of a cache line.

5.2.1.5 Instruction Alignment for the Grouping Logic

See the *UltraSPARC III Cu Processor User's Manual* for a description of grouping logic.

5.2.1.6 Impact of Instruction Alignment on Instruction Dispatch

It is important that no two branches are in the same fetch group. If there are two branches in the same group, the second branch will end the group and will cause a refetch taking two cycles. To guarantee this does not happen in the case of uncertain instruction alignment, ensure that no two branches are within four instructions of each other.

5.2.2 Instruction Cache Timing

While accesses to the I-cache hit successfully, the pipeline rarely starves for instructions. In rare cases however, the Instruction Dispatch is unable to provide a sufficient number of instructions to keep the functional units busy. For example, a taken branch to a taken branch sequence without any instructions between the branches (except for the delay slot) could only be executed at a peak rate of two instructions per cycle.

An I-cache miss does not necessarily result in bubbles being inserted into the pipeline. Part of the I-cache miss processing, or even all of it, can be overlapped with the execution of instructions that are already in the instruction buffer and are waiting to be grouped and executed. Because the operation of the Instruction Dispatch is somewhat separated from the rest of the pipeline, the I-cache miss may have occurred when the pipeline was already stalled (for example, due to a multi-cycle integer divide, floating-point divide dependency, dependency on load data that missed the D-cache, etc.). This means that the miss (or part of it) may be transparent to the pipeline.

Note – Because of the possibility of stalling the processor when the pipeline is waiting for new instructions, try to make code routines fit in the I-cache and avoid cache misses.

The UltraSPARC processor provides instrumentation to profile a program and detect if instruction accesses generate a cache miss or a cache hit. By checkpointing the counters before and after a large section of code, combined with profiling the section of code, one can determine if the frequently executed functions hit or miss the I-cache.

5.2.3 Executing Instructions With Minimum Latency

Instructions fetched from the L2-cache or the L3-cache require fewer number of cycles than fetching an instruction from main memory. The hardware can prefetch the next eight instructions if the initial fetch was from the lower 32 bytes of a 64-byte aligned memory boundary.

5.2.4 Translation Lookaside Buffer (TLB) Misses

The TLB contains the virtual page number and the associated physical page number of the most recently accessed pages.

A TLB miss is handled by software via the translation storage buffer (TSB) and takes a large number of cycles. To minimize the frequency of TLB misses, the UltraSPARC processor provides a large number of entries in the TLB.

5.2.4.1 Impact of the Annulled Slot

Grouping rules in the *UltraSPARC III Cu Processor User's Manual* describe how the UltraSPARC processor handles instructions following an annulling branch.

Avoid scheduling `WR`(`PR`, `ASR`), `SAVE`, `SAVED`, `RESTORE`, `RESTORED`, `RETURN`, `RETRY`, and `DONE` in the delay slot and in the first three groups following an annulling branch.

5.2.5 Conditional Moves vs. Conditional Branches

The `MOVcc` and `MOVR` instructions provide an alternative to conditional branches for executing short code segments. The UltraSPARC processor differentiates the two as follows:

- Conditional branches: Distancing the `SETcc` from `Bicc` does not gain any performance. The penalty for a mispredicted branch is always eight cycles. `SETcc`, `Bicc`, and the delay slot can be grouped together as shown in FIGURE 5-1.

FIGURE 5-1 Handling of Conditional Branches

- Conditional moves: A use of the destination register for the `MOVcc` follows the same rule as a load-use. FIGURE 5-2 shows a typical example.

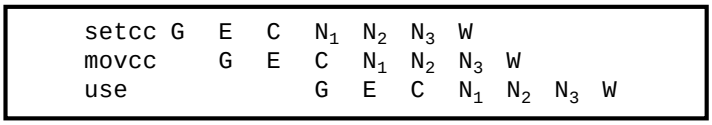

FIGURE 5-2 Handling of MOVCC

If a branch is correctly predicted, the issue rate can be higher than that of a branch that is replaced by a conditional move. If a branch is not predictable, the mispredicted penalty is significantly higher than the extra latency of a conditional move.

5.2.6 Instruction Cache Utilization

Grouping blocks that are executed frequently can effectively increase the apparent size of the I-cache. Case studies show that, often, half of the I-cache entries are never executed. Placing rarely executed code out of a line containing a frequently executed block (identified by profiling) achieves a better I-cache utilization.

5.2.7 Handling of CTI Couples

Avoid placing a CTI into the delay slot of another CTI because it will disrupt the fetch and cost many cycles.

5.2.8 Mispredicted Branches

Correctly predicted conditional branches allow the processor to group instructions from subsequent basic blocks and continue to progress speculatively until the branch is resolved. The capability of executing instructions speculatively is a significant performance boost for the UltraSPARC processor.

5.2.9 Return Address Stack (RAS)

To speed up returns from subroutines invoked through `CALL` instructions, the UltraSPARC processor dedicates an 8-deep stack to store the return address. Each time a `CALL` is detected, the return address is pushed onto this Return Address Stack. Each time a return is encountered, the address is obtained from the top of the stack and the stack is popped. The UltraSPARC processor considers a return to be a `JMPL` or `RETURN` with rs1 equal to `%o7` (normal subroutine) or `%i7` (leaf subroutine). The Return Address Stack provides a guess for the target address, so that prefetching can continue even though the address calculation has not yet been performed. `JMPL` or `RETURN` instructions using rs1 values other than `%o7` or `%i7` use the value on the top of the Return Address Stack for continuing prefetching, but they do not pop the stack.

To take full advantage of the Return Address Stack, follow the standard call and return conventions so that hardware can correctly predict the return addresses.

5.3 Data Stream Issues

The following section addresses these data stream issues:

- Data Cache Organization
- Data Cache Timing
- Data Alignment
- Using `LDDF` to Load Two Single-Precision Operands/Cycle
- Store Considerations
- Read-After-Write Hazards

5.3.1 Data Cache Organization

The D-cache is a mapped, virtually indexed, physically tagged (VIPT), write-through, non-write-allocating cache. It is logically organized as lines of 32 bytes.

5.3.2 Data Cache Timing

The latency of a load to the D-cache depends on the opcode. `LDX` and `LDUW` have two-cycle load-to-use latency while all other loads have three. For example, if the first two instructions in the instruction buffer are a load and an instruction dependent on that load, the grouping logic will break the group after the load and insert a bubble in the pipeline. It is very important to separate loads from their use.

5.3.3 Data Alignment

The SPARC V9 specification requires that all accesses be aligned on an address equal to the size of the access. Otherwise a *mem_address_not_aligned* trap is generated. This is especially important for double-precision floating-point loads, which should be aligned on an 8-byte boundary. If misalignment is determined to be possible at compile time, it is better to use two `LDF` (load floating-point, single precision) instructions and avoid the trap. The UltraSPARC processor supports single-precision loads mixed with double-precision operations, so that the case above can execute without penalty (except for the additional load). If a trap does occur, the UltraSPARC processor dedicates a trap vector for this specific misalignment, which reduces the overall penalty of the trap.

Grouping load data is desirable because a D-cache subblock can contain either four properly aligned single-precision operands or two properly aligned double-precision operands (eight and four respectively for a D-cache line). This grouping is desirable not only for improving the D-cache hit rate (by increasing its utilization density), but also for D-cache misses where, for sequential accesses, one out of two requests to the L2-cache can be eliminated.

5.3.3.1 Using LDDF to Load Two Single-Precision Operands/Cycle

The UltraSPARC processor supports single-cycle eight-byte data transfers into the floating-point register file for LDDF. Wherever possible, applications that use single-precision floating-point arithmetic heavily should organize their code and data to replace two LDFs with one LDDF. This replacement reduces the load frequency by approximately one half and cuts execution time considerably.

5.3.4 Store Considerations

The store on the UltraSPARC processor is designed so that stores can be issued even when the data is not ready. More specifically, a store can be issued in the same group as the instruction producing the result. The address of a store is buffered until the data is eventually available. Once in the store buffer, the store data is buffered until it can be completed.

The write cache can be used to exploit locality (both temporal and spatial) in the write stream.

5.3.5 Read-After-Write Hazards

Load data can be bypassed from previous stores before they become globally visible (data for load from the store queue). This bypass is specifically allowed by the total store order (TSO) memory model. Data for all types of loads cannot be bypassed from all types of stores.

All types of load instructions can get data from the store queue, except the following load instructions:

- Signed loads (`ldsb`, `ldsh`, `ldsw`)
- Atomics
- Load double to integer register file (`ldd`)
- Quad loads to integer register file
- Load from FSR register
- Block loads
- Short floating-point loads
- Loads from internal ASIs

All types of store instructions can give data to a load, except the following store instructions:

- Floating-point partial stores
- Store double from integer register file (`std`)
- Store part of atomic
- Short floating-point stores
- Stores to pages with side-effect bit set
- Stores to non-cacheable pages

When data for a load cannot be bypassed from previous stores before they become globally visible (store data is not yet retired from the store queue), the load is recirculated after the read-after-write (RAW) hazard is removed. The following conditions can cause this recirculation:

- Load data can be bypassed from more than one store in the store queue.

- The load's VA[12:0] overlaps a store's VA[12:0] and store data cannot be bypassed from the store queue.
- The load's VA[12:5] matches a store's VA[12:5] and the load misses the D-cache.
- Load is from a side-effect page (page attribute E = 1) when the store queue contains one or more stores to side-effect pages.

5.4 Performance Instrumentation

Performance instrumentation consists of processor event counters that can be used to gather statistics during program execution and calls that start and stop the gathering process. Many events can be monitored, two at a time, to gain information about the performance of the processor. Memory access and stall times, for example, can be measured using two 32-bit Performance Instrumentation Counters (PICs). The Performance Control Register (PCR) and PIC are accessed through read/write Ancillary State Register (ASR) instructions.

5.4.1 Supervisor/User Mode

Access to the PCR is privileged. Nonprivileged accesses cause a privileged_opcode trap. Software can restrict nonprivileged access to PICs by setting the PCR.PRIV field while in privileged mode. When PCR.PRIV = 1 (supervisor access only), an attempt by user software to access the PIC register causes a *privileged_action* trap. Software can control event measurements in nonprivileged or privileged modes by setting the PCR.UT (user trace) and PCR.ST (system trace) fields. The PCR has mode bits to enable the counters in privileged mode, nonprivileged mode, or either mode. The mode setting affects both counters.

5.5 Performance Control Register (PCR)

The Performance Control Register (PCR) is used to select which events to monitor and provides control for counting in privileged and/or nonprivileged modes. The 64-bit PCR is accessed through the read/write Ancillary State Register instructions (`RDASR/WRASR`). The PCR is located at ASRs 16 (10_{16}).

Two events can be measured simultaneously by setting the PIC_SL and PIC_SU fields. The counters can be enabled separately for Supervisor and User mode using UT and ST fields. The selected statistics are reflected during subsequent accesses to the PICs.

The PCR is a read/write register used to control the counting of performance monitoring events. TABLE 5-1 shows the details of the PCR. TABLE 5-2 describes the various fields of the PCR. Counts are collected in the PIC register. See *Performance Instrumentation Counter (PIC) Register* on page 99.

TABLE 5-1 Performance Control Register

ASR	R/W	Description	Reset
ASR 16_{10}	64-bit Read/Write	Privileged Mode, otherwise privileged_action trap.	0x0000.0000

TABLE 5-2 PCR Bit Description

Bit	Field	Description
[63:48]	*Reserved*	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
[47:32]	*Reserved*	Unused by the UltraSPARC IV+ processor. Read zero, write zero, or write value read previously (read-modify-write).
[31:27]	*Reserved*	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
[26:17]	*Reserved*	Unused by the UltraSPARC IV+ processor. Read zero, write zero, or write value read previously (read-modify-write).
[16:11]	SU	Selects one of up to 64 counters accessible in the upper half (bits [63:32]) of the PIC register.
[10]	*Reserved*	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
[9:4]	SL	Selects one of up to 64 counters accessible in the lower half (bits [31:0]) of the PIC register.
[3]	*Reserved*	Unused by the UltraSPARC IV+ processor. Read zero, write zero, or write value read previously (read-modify-write).
[2]	UT	User Trace Enable. If set to 1, counts events in nonprivileged mode (User).
[1]	ST	System Trace Enable. If set to 1, counts events in privileged mode (Supervisor). Notes: • If both PCR.UT and PCR.ST are set to 1, all selected events are counted. • If both PCR.UT and PCR.ST are zero, counting is disabled. • PCR.UT and PCR.ST are global fields which apply to both PIC pairs.
[0]	PRIV	Privileged. If PCR.PRIV = 1, a nonprivileged (PSTATE.PRIV = 0) attempt to access PIC (via a RDPIC or WRPIC instruction) will result in a privileged_action exception.

5.6 Performance Instrumentation Counter (PIC) Register

The 64-bit PIC is accessed through read/write Ancillary State Register instructions (`RDASR/WRASR`). PIC is located at ASRs 17 (11_{16}).

The PIC counters can be monitored during program execution to gather ongoing statistics or reconfigure during steady-state program execution to gather statistics for more than two events. The pair of 32-bit counters can accumulate over four billion events each prior to wrapping. Overflow of PICL or PICU causes a disrupting trap and SOFTINT. Active monitoring will allow the gathering software to extend the data range by periodically reading the contents of the PICs to detect and avoid overflow; an interrupt can be enabled on a counter overflow. The point at which the interrupt due to a PIC overflow is delivered may be several instructions after the instruction responsible for the overflow event. This delay is known as a *skid*. The degree of skid, a delay of a dozen or more clock cycles in length, depends on the event that caused the overflow and the state of the processor pipelines at the time the overflow occurred. It may not be possible to associate a counter overflow with the particular instruction that caused it due to the skid problem.

Each of the two 32-bit PICs can accumulate over four billion events before wrapping around. Overflow of PICL or PICU causes a disrupting trap and SOFTINT register bit 15 to be set to 1. If the overflow occurs when PSTATE.IE = 1 and PIL < 15, an *interrupt_level_15* trap is generated. Extended event logging can be accomplished by periodic reading of the contents of the PICs before each overflows. Additional statistics can be collected using the two PICs over multiple

passes of program execution. Two events can be measured simultaneously by setting the PCR.SU/PCR.SL fields along with the PCR.UT and PCR.ST fields. The selected statistics are reflected during subsequent accesses to the PICs.

The difference between the values read from the PIC on two reads reflects the number of events that occurred between register reads. Software can only rely on read-to-read PIC accesses to get an accurate count and not a write-to-read of the PIC counters. TABLE 5-3 shows the details of the PIC. TABLE 5-4 describes the various fields of the PIC.

TABLE 5-3 Performance Instrumentation Counter Register

ASR	R/W	Description	Reset
ASR 17_{10}	64-bit Read/Write **Note:** Writes are designed for diagnostic and test purposes.	Accessibility depends on PCR.PRIV bit: 0 = accessible in any mode 1 = accessible in Supervisor Mode, otherwise privileged_action trap	0x0000.0000

TABLE 5-4 PIC Register Fields

Bit	Field	Description
[63:32]	PICU	32-bit field representing the count of an event selected by the SU field of the Performance Control Register (PCR)
[31:0]	PICL	32-bit field representing the count of an event selected by the SL field of the Performance Control Register (PCR)

5.6.1 PIC Counter Overflow Trap Operation

When a PIC counter overflows, an interrupt is generated as described in TABLE 5-5.

TABLE 5-5 PIC Counter Overflow Processor Compatibility Comparison

Function	Description
PIC Counter Overflow	The counter overflow trap is triggered on the transition from value FFFF FFFF$_{16}$ to value 0. The point at which the interrupt is delivered may be several instructions after the instruction responsible for the overflow event. This situation is known as a *skid*. • The counter wraps to zero. • SOFTINT register bit 15 is set to 1. • An interrupt_level_15 trap (a disrupting trap) is generated.

5.7 Performance Instrumentation Operation

FIGURE 5-3 shows how an operating system might use the performance instrumentation features to provide event monitoring services.

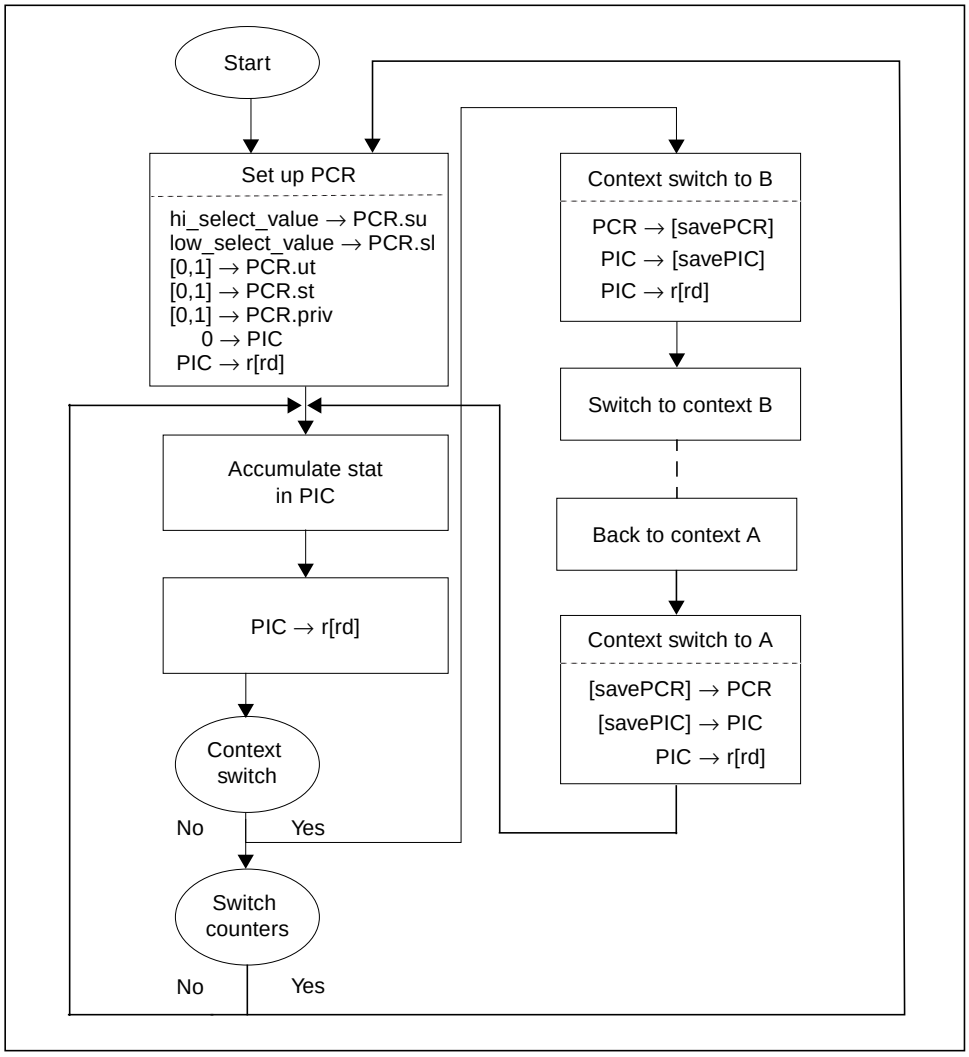

FIGURE 5-3 Operational Flow Diagram for Controlling Event Counters

Set up the PCR register as desired to select two events and the modes in which data should be collected. When more than two events need to be monitored, the program, code sequence, or code loop need to be run again with the new events enabled. It is not possible to monitor more than two events at any given time. The monitoring must consider the real effects of the computer that includes calls to the system and interrupts. When used, the PCR register is considered part of a process state and must be saved and restored when switching process contexts.

Multiple data collection times can be done while the program executes to show ongoing statistics.

5.7.1 Performance Instrumentation Implementations

Counting events and cycle stalls is sometimes complex due to dynamic conditions and cancelled activities.

5.7.2 Performance Instrumentation Accuracy

The performance instrumentation counters are designed to provide reasonable accuracy especially when used to count hundreds or thousands of events or stall cycles or when comparing the PIC counts that have recorded a similar number of events or stall cycles. Accuracy is most challenging when trying to associate an event to an instruction and when comparing PIC counts with one rarely occurring count.

When using the overflow trap, it is sometimes difficult to pinpoint the instruction that is responsible for the overflow because of the way the pipeline is designed. A delay of several instructions is possible before the overflow is able to stop the current instruction flow and fetch the trap vector. The skid for the load miss detection case is small. The skid value cannot be measured and its length depends on what event or stall cycle is being measured and what other instructions are in the pipeline.

5.8 Pipeline Counters

5.8.1 Instruction Execution and Clock Counts

The instruction execution count monitors are described in TABLE 5-6 for clock and instruction execution counts.

TABLE 5-6 Instruction Execution Clock Cycles and Counts

Counter	Description
Cycle_cnt	[PICL 00.0000 and PICU 00.0000] Counts clock cycles. This counter increments the same as the SPARC V9 TICK register, except that cycle counting is controlled by the PCR.UT and PCR.ST fields.
Instr_cnt	[PICL 00.0001 and PICU 00.0001] Counts the number of instructions completed (retired). This count does not include annulled, mispredicted, trapped, or helper instructions.

5.8.1.1 Synthesized Clocks Per Instruction (CPI)

The cycle and instruction counts can be used to calculate the average number of instructions completed per cycle:

$$CPI = Cycle\_cnt / Instr\_cnt \quad \textbf{(EQ 1)}$$

In (EQ 1), the formula refers to clock (cycles) per instruction.

5.8.2 IIU Branch Statistics

The counters listed in TABLE 5-7 record branch prediction statistics for retired non-annulled branch instructions. A retired branch in the following descriptions refers to a branch that reaches the D-stage of the pipeline without being cancelled. These counters are private counters.

TABLE 5-7 Counters for Collecting IIU Branch Statistics

Counter	Description
IU_stat_br_miss_taken [PICL]	Number of retired non-annulled conditional branches that were predicted to be taken, but in fact were not taken.
IU_stat_br_miss_untaken [PICU]	Number of retired non-annulled conditional branches that were predicted to be not taken, but in fact were taken.
IU_stat_br_count_taken [PICU]	Number of retired non-annulled conditional branches that were taken.
IU_stat_br_count_untaken [PICL]	Number of retired non-annulled conditional branches that were not taken.
IU_stat_jmp_correct_pred [PICL]	Number of retired non-annulled register indirect jumps predicted correctly. Register indirect jumps are jmpl instructions (op=2_{16}, op3=38_{16}) except for the cases treated as returns (described in IU_stat_ret_correct_pred counter below).
IU_stat_jmp_mispred [PICU]	Number of retired non-annulled register indirect jumps mispredicted.
IU_stat_ret_correct_pred [PICL]	Number of retired non-annulled returns predicted correctly. Returns include the return instruction (op=2_{16},op3=39_{16}) and the ret/retl synthetic instructions. Ret/retl are jmpl instructions (op=2_{16},op3=38_{16}) with the following format: ret: jmpl %i7+8,%g0, retl: jmpl %o7+8,%g0
IU_stat_ret_mispred [PICU]	Number of retired non-annulled returns mispredicted.

5.8.3 IIU Stall Counts

IIU stall counts, listed in TABLE 5-8, correspond to the major cause of pipeline stalls from the fetch and decode stages of the pipeline. These counters count cycles during which no instructions are dispatched (or issued) because the I-queue is empty due to various events including I-cache miss and refetching due to a second branch in a fetch group. Stalls are counted for each clock at which the associated condition is true.

The counters listed in TABLE 5-8 are all per-core counters.

TABLE 5-8 Counters for IIU stalls

Counter	Description
Dispatch0_IC_miss [PICL]	Stall cycles due to the event that no instructions are dispatched because the I-queue is empty from an I-cache miss.
Dispatch0_2nd_br [PICL]	Stall cycles due to the event that no instructions are dispatched because the I-queue is empty because there were two branch instructions in the fetch group causing the second branch in the group to be refetched.
Dispatch0_other [PICU]	Stall cycles due to the event that no instructions are dispatched because the I-queue is empty due to various other events, including branch target address fetch and various events which cause an instruction to be refetched. Note that this count does not include IIU stall cycles due to recirculation (measured by Re_* counters, see Section 5.8.5). Also, the count does not include the stall cycles due to I-cache misses (measured by Dispatch0_IC_miss) and refetch due to second branch in the fetch group (measured by Dispatch0_2nd_br).

Note – If multiple events result in IIU stall in a given cycle, only one of the counters will be incremented based on the following priority: Re_*>Dispatch0_ic_miss>Dispatch0_2nd_br>Dispatch0_other.

5.8.4 R-stage Stall Counts

The counters described in TABLE 5-9 count the stall cycles at the R-stage of the pipeline. The stalls may happen due to the unavailability of a resource or the unavailability of the result of a preceding instruction. Stalls are counted for each clock at which the associated condition is true. The counters are private counters.

TABLE 5-9 Counters for R-stage Stalls

Counter	Description
Rstall_storeQ [PICL]	Stall cycles when a store instruction which is the next instruction to be executed is stalled in the R-stage due to the store queue being full.
Rstall_FP_use [PICL]	Stall cycles when the next instruction to be executed is stalled in the R-stage waiting for the result of a preceding floating-point instruction in the pipeline that is not yet available.
Rstall_IU_use [PICL]	Stall cycles when the next instruction to be executed is stalled in the R-stage waiting for the result of a preceding integer instruction in the pipeline that is not yet available.

Note – If multiple events result in R-stage stall in a given cycle, only one of the counters will be incremented based on the following priority: Rstall_IU_use>Rstall_FP_use>Rstall_storeQ.

5.8.5 Recirculate Stall Counts

The counters listed in TABLE 5-10 count the stall cycles due to recirculation. The recirculation may happen due to non-bypassable RAW hazard, non-bypassable FPU condition, load miss, or prefetch queue full conditions. These are also private counters.

TABLE 5-10 Counters for Recirculation

Counter	Description
Re_RAW_miss [PICU]	Stall cycles due to recirculation when there is a load instruction in the E-stage of the pipeline which has a non-bypassable read-after-write (RAW) hazard with an earlier store instruction. Note that, due to implementation issue, this count also includes the stall cycles due to recirculation of prefetch requests when the prefetch queue is full (see Re_PFQ_full description in page 105). To determine the stall cycles due to non-bypassable RAW hazard only, subtract Re_PFQ_full from Re_RAW_miss, i.e., actual Re_RAW_miss = Re_RAW_miss - Re_PFQ_full.
Re_FPU_bypass[PICU]	Stall cycles due to recirculation when an FPU bypass condition that does not have a direct bypass path occurs. FPU bypass cannot occur in the following cases: (1) a PDIST instruction is followed by a dependent FP multiply/FG multiply instruction. (2) a PDIST instruction is followed by another PDIST instruction with the same destination register (WAW hazard) which in turn is followed by a dependent FP multiply/FG multiply instruction.
Re_DC_miss[PICL]	Stall cycles due to recirculation of cacheable loads that miss D-cache. This includes L2 hit, L2 miss/L3 hit, and L3 miss cases. Stall cycles from the point when a cacheable D-cache load miss reaches D stage to the point when the recirculated flow reaches D-stage again are counted. This is equivalent to the load-to-use latency of the load instruction. Note: The count does not include stall cycles for cacheable loads that recirculate due to a D-cache miss for which there is an outstanding prefetch (fcn=1) request in the prefetch queue (LAP hazard).
Re_L2_miss[PICL]	Stall cycles due to recirculation of cacheable loads that miss both D-cache and L2 cache. This includes both L3 hit and L3 miss cases. Stall cycles from the point when L2-cache miss is detected to the point when the recirculated flow reaches D-stage again are counted. Note that these stall cycles are also counted in Re_DC_miss.
Re_L3_miss [PICU]	Stall cycles due to recirculation of cacheable loads that miss D-cache, L2, and L3 cache. Stall cycles from the point when L3-cache miss is detected to the point when the recirculated flow reaches D-stage again are counted. Note that these stall cycles are also counted in Re_DC_miss and Re_L2_miss.
Re_PFQ_full [PICU]	Stall cycles due to recirculation of prefetch instructions because the prefetch queue (PFQ) was full. The count includes stall cycles for strong software prefetch instructions that recirculate when the PFQ is full. The count also includes stall cycles for any software prefetch instruction, when the PCM bit in the Data cache Unit Control Register (DCUCR) is enabled, that recirculates when the PFQ is full.
Re_DC_missovhd [PICU]	Counts the overhead of stall cycles due to D-cache load miss. Includes cycles from the point the load reaches D stage (about to be recirculated) to the point L2 cache hit/miss is reported. Note: The count does not include overhead cycles for cacheable loads that recirculate due to D-cache miss for which there is an outstanding prefetch (fcn=1) request in the prefetch queue (LAP hazard).

5.9 Cache Access Counters

Instruction, data, prefetch, write and L2, L3 cache access statistics can be collected through the counters listed in TABLE 5-11. Counts are updated by each cache access regardless of whether the access will be used.

The Instruction, data, prefetch, and write cache counters are private counters. Because the L2 and the L3 caches are shared by the two cores, some events are counted by the core which causes the access, while other events cannot be attributed to an individual core and are treated as shared events.

TABLE 5-11 Cache Access Counters ***(1 of 5)***

Counter	Description
Instruction Cache	
IC_ref [PICL]	Number of I-cache references. I-cache references are fetches of up to four instructions from an aligned block of eight instructions. Note that the count includes references for non-cacheable instruction accesses and instructions that were later cancelled due to misspeculation or other reasons. Thus, the count is generally higher than the number of references for instructions that were actually executed.
IC_fill [PICU]	Number of I-cache fills excluding fills from the instruction prefetch buffer. This is the best approximation of the number of I-cache misses for instructions that were actually executed. The count includes some fills related to wrong path instructions where the branch was not resolved before the fill took place. The count is updated for 64 Byte fills only; in some cases, the fetcher performs 32 Byte fills to I-cache.
IPB_to_IC_fill [PICL]	Number of I-cache fills from the instruction prefetch buffer (IPB). The count includes some fills related to wrong path instructions. The count is updated on 64 Byte granularity.
IC_pf [PICU]	Number of I-cache prefetch requests sent to L2 cache.
IC_L2_req [PICU]	Number of I-cache requests sent to L2 cache. This includes both I-cache miss requests and I-cache prefetch requests. The count does not include non-cacheable acesses to the I-cache. Note that some of the I-cache requests sent to L2 cache may not eventually be filled into the I-cache.
IC_miss_cancelled [PICL]	Number of I-cache miss requests cancelled due to new fetch stream.The cancellation may be due to misspeculation, recycle or other events.
ITLB_miss [PICU]	Number of I-TLB miss traps taken.
Data Cache	
DC_rd [PICL]	Number of D-cache read references by cacheable loads (excluding block loads). References by all cacheable load instructions (including LDD) are considered as 1 reference each. The count is only updated for load instructions that retired.
DC_rd_miss [PICU]	Number of cacheable loads (excluding atomics and block loads) that miss D-cache as well as P-cache (for FP loads). The count is only updated for load instructions that retired.
DC_wr [PICU]	Number of D-cache write references by cacheable stores (excluding block stores). References by all cacheable store instructions (including STD and atomic) are counted as 1 reference each. The count is only updated for store instructions that retired.

TABLE 5-11 Cache Access Counters *(2 of 5)*

Counter	Description
DC_wr_miss [PICL]	Number of D-cache write misses by cacheable stores (excluding block stores). The count is only updated for store instructions that retired. Note that hitting or missing the D-cache does not significantly impact the performance of a store.
DTLB_miss [PICU]	Number of D-TLB miss traps taken.
Write Cache	
WC_miss [PICU]	Number of W-cache misses by cacheable stores.
Prefetch Cache	
PC_miss [PICU]	Number of cacheable FP loads that miss P-cache, irrespective of whether the loads hit or miss the D-cache. The count is only updated for FP load instructions that retired.
PC_soft_hit[PICU]	Number of cacheable FP loads that hit a P-cache line that was prefetched by a software prefetch instruction, irrespective of whether the loads hit or miss the D-cache. The count is only updated for FP load instructions that retired.
PC_hard_hit[PICU]	Number of cacheable FP loads that hit a P-cache line that was fetched by a FP load or a hardware prefetch, irrespective of whether the loads hit or miss the D-cache. The count is only updated for FP load instructions that retired. Note that, if hardware prefetching is disabled (DCUCR bit HPE = 0), the counter will count the number of hits to P-cache lines that were fetched by a FP load only, since no hardware prefetches will be issued in that case.
PC_inv [PICU]	Number of P-cache lines that were invalidated due to external snoops, internal stores, and L2 evictions.
PC_rd [PICL]	Number of cacheable FP loads to P-cache. The count is only updated for FP load instructions that retired.
SW_pf_instr [PICL]	Number of retired software prefetch instructions. Note: SW_pf_instr = SW_pf_exec + SW_pf_dropped + SW_pf_duplicate.
SW_pf_exec [PICU]	Number of retired, non-trapping software prefetch instructions that completed, i.e., number of retired prefetch instructions that were not dropped due to the prefetch queue being full. The count does not include duplicate prefetch instructions for which the prefetch request was not issued because it matched an outstanding prefetch request in the prefetch queue or the request hit the P-cache.
HW_pf_exec [PICL]	Number of hardware prefetches enqueued in the prefetch queue.
SW_pf_str_exec [PICU]	Number of retired, non-trapping strong prefetch instructions that completed. The count does not include duplicate strong prefetch instructions for which the prefetch request was not issued because it matched an outstanding prefetch request in the prefetch queue or the request hit the P-cache.
SW_pf_dropped [PICU]	Number of software prefetch instructions dropped due to TLB miss or due to the prefetch queue being full.
SW_pf_duplicate [PICU]	Number of software prefetch instructions that were dropped because the prefetch request matched an outstanding request in the prefetch queue or the request hit the P-cache.
SW_pf_str_trapped [PICL]	Number of strong software prefetch instructions trapping due to TLB miss.
SW_pf_L2_installed [PICU]	Number of software prefetch instructions that installed lines in the L2 cache.
SW_pf_PC_installed [PICL]	Number of software prefetch instructions that installed lines in the P-cache. Note that both SW_pf_PC_installed and SW_pf_L2_installed can be updated by some prefetch instructions depending on the prefetch function.

TABLE 5-11 Cache Access Counters *(3 of 5)*

Counter	Description
L2 Cache **Note:** The L2 cache access counters do not include retried L2 cache requests.	
Private L2 counters	
L2_ref [PICL]	Number of L2 cache references from this core by cacheable I-cache, D-cache, P-cache, and W-cache (excluding block stores that miss L2-cache) requests. A 64 Byte request is counted as 1 reference. Note that the load part and the store part of an atomic is counted as a single reference.
L2_miss [PICU]	Number of L2 cache misses from this core by cacheable I-cache, D-cache , P-cache, and W-cache (excluding block stores) requests. This is equivalent to the number of L3-cache references requested by this core. Note that the load part and the store part of an atomic is counted as a single request. Also, the count does not include hardware prefetch requests that miss L2 cache (L2_HWPF_miss).
L2_rd_miss [PICL]	Number of L2 cache misses from this core by cacheable D-cache requests (excluding block loads and atomics).
L2_IC_miss [PICL]	Number of L2 cache misses from this core by cacheable I-cache requests. The count includes some wrong path instruction requests.
L2_SW_pf_miss [PICL]	Number of L2 cache misses by software prefetch requests from this core.
L2_HW_pf_miss [PICU]	Number of L2 cache misses by hardware prefetch requests from this core. Note that hardware prefetch requests that miss L2 cache are not sent to L3 cache; they are dropped.
L2_write_hit_RTO [PICL]	Number of L2 cache hits in O, Os, or S state by cacheable store requests from this core that do a read-to-own (RTO) bus transaction. The count does not include RTO requests for prefetch (fcn=2,3/22,23) instructions.
L2_write_miss [PICL]	Number of L2 cache misses from this core by cacheable store requests (excluding block stores). The count does not include write miss requests for prefetch (fcn=2,3/22,23) instructions. Note that this count also includes the L2_write_hit RTO cases (RTO_nodata), i.e., stores that hit L2-cache in O, Os, or S state.
L2_hit_other_half [PICL]	Number of L2 cache hits from this core to the ways filled by the other core when the cache is in the pseudo-split mode. Note that the counter does not count if the L2 cache is not in pseudo-split mode. If the L2 cache is switched from the psuedu-split mode to regular mode, the counter will retain its value.
L2_wb [PICL]	Number of L2 cache lines that were written back to the L3 cache because of requests from this core.
Shared L2 event counters	
L2_wb_sh [PICL]	Total number of L2 cache lines that were written back to the L3 cache due to requests from both cores.
L2_snoop_inv_sh [PICL]	Total number of L2 cache lines that were invalidated due to other processors doing RTO, RTOR, RTOU, or WS transactions.
L2_snoop_cb_sh [PICL]	Total number of L2 cache lines that were copied back due to other processors. The count includes copybacks due to both foreign copy-back and copy-back-invalidate requests (i.e., foreign RTS, RTO, RS, RTSR, RTOR, RSR, RTSM, RTSU, or RTOU requests).

TABLE 5-11 Cache Access Counters *(4 of 5)*

Counter	Description
L2_hit_I_state_sh [PICU]	Total number of tag hits in L2 cache when the line is in I state. The count does not include L2 cache tag hits for hardware prefetch and block store requests. This counter approximates the number of coherence misses in the L2 cache in a multiprocessor system.
L3 Cache Note: The L3 cache access counters do not include retried L3 cache requests.	
Private L3-cache counters	
L3_miss [PICU]	Number of L3 cache misses sent out to SIU from this core by cacheable I-cache, D-cache, P-cache, and W-cache (excluding block stores) requests. Note that the load part and the store part of an atomic is counted as a single reference.
L3_rd_miss [PICL]	Number of L3 cache misses from this core by cacheable D-cache requests (excluding block loads and atomics).
L3_IC_miss [PICU]	Number of L3 cache misses by cacheable I-cache requests from this core.

TABLE 5-11 Cache Access Counters *(5 of 5)*

Counter	Description
L3_SW_pf_miss [PICU]	Number of L3 cache misses by software prefetch requests from this core.
L3_write_hit_RTO [PICU]	Number of L3 cache hits in O, Os, or S state by cacheable store requests from this core that do a read-to-own (RTO) bus transaction. The count does not include RTO requests for prefetch (fcn=2,3/22,23) instructions.
L3_write_miss_RTO [PICU]	Number of L3 cache misses from this core by cacheable store requests that do a read-to-own (RTO) bus transaction. The count does not include RTO requests for prefetch (fcn=2,3/22,23) instructions. Note that this count also includes the L3_write_hit_RTO cases (RTO_nodata), i.e., stores that hit L3-cache in O, Os, or S state.
L3_hit_other_half [PICU]	Number of L3 cache hits from this core to the ways filled by the other core when the cache is in pseudo-split mode. Note that the counter is not incremented if the L3 cache is not in pseudo-split mode. If the L3 cache is switched from the psuedu-split mode to regular mode, the counter will retain its value.
L3_wb [PICU]	Number of L3 cache lines that were written back because of requests from this core.
Shared L3 Event counters	
L3_wb_sh [PICU]	Total number of L3 cache lines that were written back due to requests from both cores.
L2L3_snoop_inv_sh [PICU]	Total number of L2 and L3 cache lines that were invalidated due to other processors doing RTO, RTOR, RTOU, or WS transactions. Note that the count includes invalidations to L2 miss block, but does not include invalidations to L3 miss block.
L2L3_snoop_cb_sh [PICU]	Total number of L2 and L3 cache lines that were copied back due to other processors. The count includes copybacks due to both foreign copy-back and copy-back-invalidate requests (i.e., foreign RTS, RTO, RS, RTSR, RTOR, RSR, RTSM, RTSU, or RTOU requests). Note that the count includes copybacks from L2 miss block, but does not include copybacks from L3 miss block. The total number of cache-to-cache transfers observed within the processor can be formulated as `the` following: Cache_to_cache_transfer = $L2L3\_snoop\_cb\_sh+SI\_RTS\_src\_data_{(core0+core1)}+SI\_RTO\_src\_data_{(core0+core1)}$ For the total number of cache-to-cache transfers in an n-chip multiprocessor system, use: Cache_to_cache_transfer = $L2L3\_snoop\_cb\_sh_{chip0}+L2L3\_snoop\_cb\_sh_{chip1}+.....+L2L3\_snoop\_cb\_sh_{chip(n-1)}$
L3_hit_I_state_sh [PICL]	Total number of tag hits in L3 cache (that miss in L2) when the line is in I state. This counter approximates the number of coherence misses in the L3 cache in a multiprocessor system.

5.10 Memory Controller Counters

Memory controller statistics are collected through the counters listed in TABLE 5-12. These counters are shared counters.

TABLE 5-12 Counters for Memory Controller Statistics

Counter	Description
MC_reads_0_sh [PICL]	Number of read requests completed to memory bank 0. Note that some memory read requests correspond to transactions where some other processor's cache contains a dirty copy of the data and the data will really be provided by that processor's cache.
MC_reads_1_sh [PICL]	The same as above for bank 1.
MC_reads_2_sh [PICL]	The same as above for bank 2.
MC_reads_3_sh [PICL]	The same as above for bank 3.
MC_writes_0_sh [PICU]	Number of write requests completed to memory bank 0.
MC_writes_1_sh [PICU]	The same as above for bank 1.
MC_writes_2_sh [PICU]	The same as above for bank 2.
MC_writes_3_sh [PICU]	The same as above for bank 3.
MC_stalls_0_sh [PICL]	Number of processor cycles that requests were stalled in the MCU queues because bank 0 was busy with a previous request. The delay could be due to data bus contention, bank busy, data availability for a write, etc.
MC_stalls_1_sh [PICU]	The same as above for bank 1.
MC_stalls_2_sh [PICL]	The same as above for bank 2.
MC_stalls_3_sh [PICU]	The same as above for bank 3.

5.11 Data Locality Counters for Scalable Shared Memory Systems

Data locality performance event counters in the UltraSPARC IV+ processor improve the ability to monitor and exploit performance in Scalable Shared Memory systems where there are multiprocessor system clusters using Shared Memory Protocol that are tied to other clusters using

fabric interconnect utilizing the Scalable Shared Memory (SSM) architecture. SSM data locality counters are listed in TABLE 5-13. The SSM_new_transaction_sh counter is shared by both cores, while the other SSM counters count private events.

TABLE 5-13 SSM data locality counters

Counter	Description
SSM_new_transaction_sh [PICL]	Number of new SSM transactions (RTSU, RTOU, UGM) observed by this processor on the Fireplane Interconnect (Safari bus).
SSM_L3_wb_remote [PICL]	Number of L3 cache line victimizations from this core which generate R_WB transactions to non-LPA (remote physical address) region.
SSM_L3_miss_local [PICL]	Number of L3 cache misses to LPA (local physical address) from this core which generate an RTS, RTO, or RS transaction.
SSM_L3_miss_mtag_remote [PICL,PICU]	Number of L3 cache misses to LPA (local physical address) from this core which generate retry (R_*) transactions including R_RTS, R_RTO, and R_RS.
SSM_L3_miss_remote [PICU]	Number of L3 cache misses from this core which generate retry (R_*) transactions to non-LPA (non-local physical address) address space, or R_WS transactions due to block store (BST) / block store commit (BSTC) to any address space (LPA or non-LPA), or R_RTO due to atomic request on Os state to LPA space. Note that this counter counts more than just remote misses, as defined above. To determine the actual number of remote misses, use: L3_miss-SSM_L3_miss_local.

5.11.1 Scalable Shared Memory Systems

Typically, four to six local processors are in a system cluster and have their own local memory subsystem(s). They use a Shared Memory Protocol to maintain data coherency among themselves. Data coherency is maintained between system clusters using a directory-based Scalable Shared Memory data coherency mechanism to insure data coherency across systems with a large number of processors.

The data locality event counters are only valid for Scalable Shared Memory system architectures.

5.11.2 Event Tree

The SSM data locality event counters are illustrated in FIGURE 5-4.

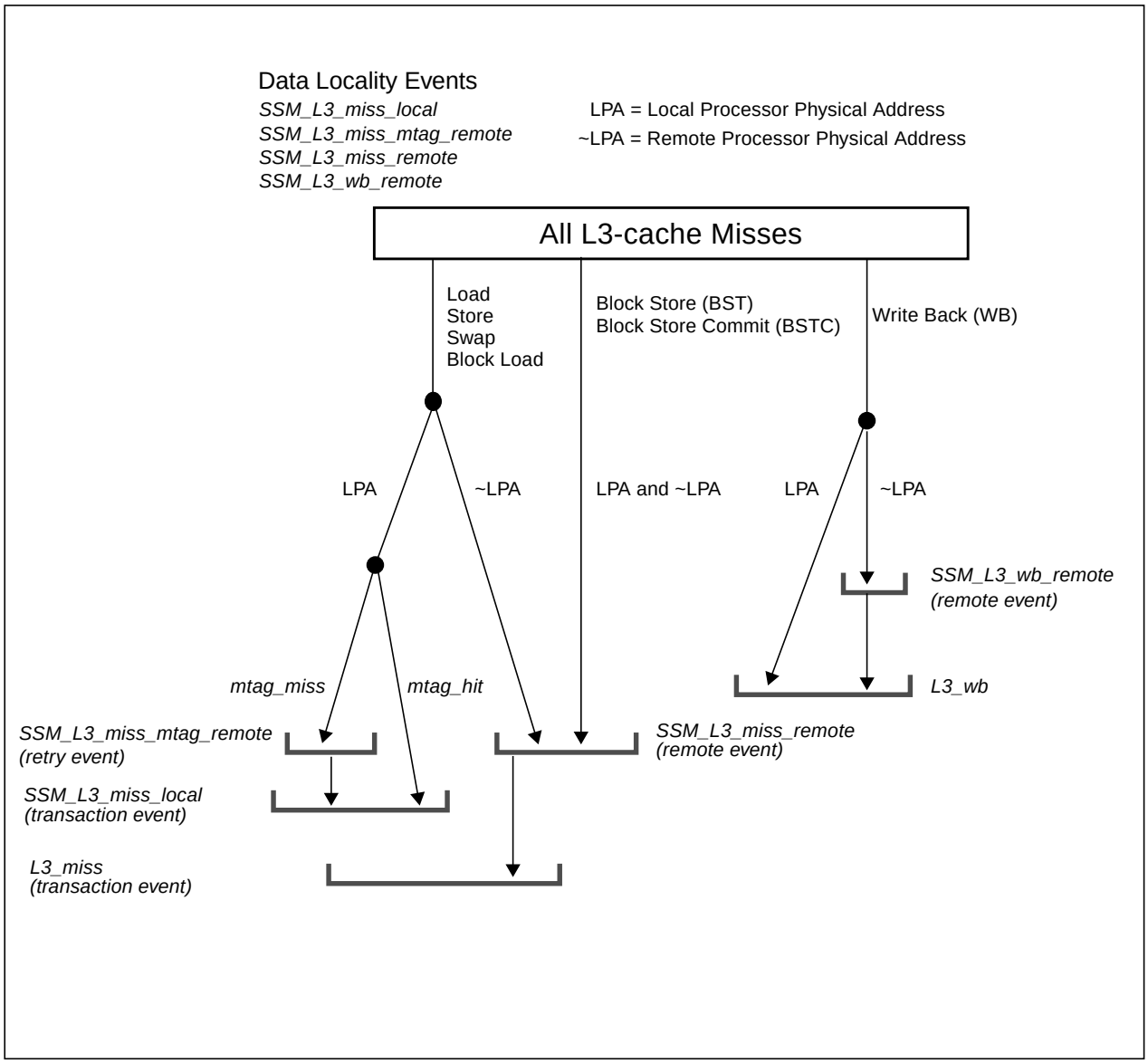

FIGURE 5-4 SSM Performance Counter Event Tree

5.11.3 Data Locality Event Matrix

TABLE 5-14 shows the data locality event matrix.

TABLE 5-14 Data Locality Events

<table>
<tr><th rowspan="2">MODE</th><th rowspan="2">Combined State</th><th colspan="6">Processor Action</th></tr>
<tr><th>Load</th><th>Store/ Swap</th><th>Block Load</th><th>Block Store</th><th>Block Store with Commit</th><th>Write Back</th></tr>
<tr><td rowspan="6">LPA</td><td>I</td><td>miss: RTS issued</td><td>miss: RTO issued</td><td>miss: RS issued</td><td>miss: R_WS issued</td><td rowspan="5">miss: R_WS issued</td><td rowspan="2">none</td></tr>
<tr><td>E</td><td rowspan="4">hit</td><td>hit: E->M</td><td rowspan="4">hit</td><td>hit: E->M</td></tr>
<tr><td>S</td><td rowspan="2">MTag miss: RTO issued</td><td rowspan="3">miss: R_WS issued</td></tr>
<tr><td>O</td><td>WB</td></tr>
<tr><td>O_s</td><td>MTag miss: R_RTO issued</td><td>WB</td></tr>
<tr><td>M</td><td colspan="4">hit</td><td>WB</td></tr>
<tr><td rowspan="5">LPA Retried</td><td>I</td><td>MTag miss: R_RTS issued</td><td>MTag miss: R_RTO issued</td><td>MTag miss: R_RS issued</td><td colspan="2">invalid</td><td>none</td></tr>
<tr><td>E</td><td colspan="6">invalid</td></tr>
<tr><td>S</td><td rowspan="3">invalid</td><td rowspan="3">MTag miss: R_RTO issued</td><td colspan="2" rowspan="3">invalid</td><td rowspan="3">none</td></tr>
<tr><td>O</td></tr>
<tr><td>O_s</td></tr>
<tr><td>M</td><td colspan="6">invalid</td></tr>
<tr><td rowspan="6">~LPA</td><td>I</td><td>miss: R_RTS issued</td><td>miss: R_RTO issued</td><td>miss: R_RS issued</td><td>miss: R_WS issued</td><td rowspan="5">miss: R_WS issued</td><td rowspan="2">none</td></tr>
<tr><td>E</td><td>hit</td><td>hit: E->M</td><td>hit</td><td>hit: E->M</td></tr>
<tr><td>S</td><td rowspan="3">hit</td><td rowspan="3">MTag miss: R_RTO issued</td><td rowspan="3">hit</td><td rowspan="3">miss: R_WS issued</td></tr>
<tr><td>O</td><td rowspan="3">miss: R_WB issued</td></tr>
<tr><td>O_s</td></tr>
<tr><td>M</td><td colspan="4">hit</td><td></td></tr>
</table>

5.11.3.1 Local Processor Physical Address (LPA) Retried Events

Retry is to issue an R_* transaction for an RTS/RTO/RS transaction that gets unexpected MTag from the SSM system interconnect (for example, cache state = O and MTag state = gS). A retry takes place in LPA.

5.12 Miscellaneous Counters

5.12.1 System Interface Event Counters

System interface statistics are collected through the counters listed in TABLE 5-15. The counters with a _sh suffix are shared by both cores, and so, the count in both cores is the same for the shared counters.

TABLE 5-15 Counters for System Interface Statistics

Counter	Description
SI_snoop_sh [PICL]	Number of snoops from other processors on the system (due to foreign RTS, RTSR, RTO, RTOR,RS, RTSM, WS, RTSU, RTOU, UGM).
SI_ciq_flow_sh [PICL]	Number of system cycles with flow control (PauseIn) observed by the local processor.
SI_owned_sh [PICU]	Number of times owned_in is asserted on bus requests from the local processor. This corresponds to the number of requests from this processor that will be satisfied either by the local processor's cache (in case of RTO_nodata), or by the cache of another processor on the system, but not by the memory.
SI_RTS_src_data [PICL]	Number of local RTS transactions due to I-cache, D-cache, or P-cache requests from this core where data is from the cache of another processor on the system, not from memory. The count does not include re-issued local RTS (i.e., RTSR) transactions.
SI_RTO_src_data [PICU]	Number of local RTO transactions due to W-cache or P-cache requests from this core where data is from the cache of another processor on the system, not from memory. The count does not include local RTO_nodata and re-issued local RTO (i.e., RTOR) transactions.

5.12.2 Software Events

Software statistics are collected through the counter listed in TABLE 5-16. This is a private counter.

TABLE 5-16 Counters for Software Statistics

Counter	Description
SW_count_NOP [PICL,PICU]	Number of retired, non-annulled special software NOP instructions (which is equivalent to `sethi %hi(0xfc000),%g0` instruction).

5.12.3 Floating-Point Operation Events

Floating-point operation statistics are collected through the counters listed in TABLE 5-17. These are private counters

TABLE 5-17 Counters for Floating-Point Operation Statistics

Counter	Description
FA_pipe_completion [PICL]	Number of retired instructions that complete execution on the Floating-Point/ Graphics ALU pipeline.
FM_pipe_completion [PICU]	Number of retired instructions that complete execution on the Floating-Point/ Graphics Multiply pipeline.

5.13 PCR.SL and PCR.SU Encoding

TABLE 5-18 lists PCR.SL selection bit field encoding for the PICL counters as well as PCR.SU encoding for the PICU counters.

TABLE 5-18 PCR.SU and PCR.SL Selection Bit Field Encoding ***(1 of 2)***

PCR.SU Value	PICU Selection	PCR.SL Value	PICL Selection
000000	Cycle_cnt	000000	Cycle_cnt
000001	Instr_cnt	000001	Instr_cnt
000010	Dispatch0_other	000010	Dispatch0_IC_miss
000011	DC_wr	000011	IU_stat_jmp_correct_pred
000100	Re_DC_missovhd	000100	Dispatch0_2nd_br
000101	Re_FPU_bypass	000101	Rstall_storeQ
000110	L3_write_hit_RTO	000110	Rstall_IU_use
000111	L2L3_snoop_inv_sh	000111	IU_stat_ret_correct_pred
001000	IC_L2_req	001000	IC_ref
001001	DC_rd_miss	001001	DC_rd
001010	L2_hit_I_state_sh	001010	Rstall_FP_use
001011	L3_write_miss_RTO	001011	SW_pf_instr
001100	L2_miss	001100	L2_ref
001101	SI_owned_sh	001101	L2_write_hit_RTO
001110	SI_RTO_src_data	001110	L2_snoop_inv_sh
001111	SW_pf_duplicate	001111	L2_rd_miss
010000	IU_stat_jmp_mispred	010000	PC_rd
010001	ITLB_miss	010001	SI_snoop_sh
010010	DTLB_miss	010010	SI_ciq_flow_sh
010011	WC_miss	010011	Re_DC_miss
010100	IC_fill	010100	SW_count_NOP

TABLE 5-18 PCR.SU and PCR.SL Selection Bit Field Encoding *(2 of 2)*

PCR.SU Value	PICU Selection	PCR.SL Value	PICL Selection
010101	IU_stat_ret_mispred	010101	IU_stat_br_miss_taken
010110	Re_L3_miss	010110	IU_stat_br_count_untaken
010111	Re_PFQ_full	010111	HW_pf_exec
011000	PC_soft_hit	011000	FA_pipe_completion
011001	PC_inv	011001	SSM_L3_wb_remote
011010	PC_hard_hit	011010	SSM_L3_miss_local
011011	IC_pf	011011	SSM_L3_miss_mtag_remote
011100	SW_count_NOP	011100	SW_pf_str_trapped
011101	IU_stat_br_miss_untaken	011101	SW_pf_PC_installed
011110	IU_stat_br_count_taken	011110	IPB_to_IC_fill
011111	PC_miss	011111	L2_write_miss
100000	MC_writes_0_sh	100000	MC_reads_0_sh
100001	MC_writes_1_sh	100001	MC_reads_1_sh
100010	MC_writes_2_sh	100010	MC_reads_2_sh
100011	MC_writes_3_sh	100011	MC_reads_3_sh
100100	MC_stalls_1_sh	100100	MC_stalls_0_sh
100101	MC_stalls_3_sh	100101	MC_stalls_2_sh
100110	Re_RAW_miss	100110	L2_hit_other_half
100111	FM_pipe_completion	100111	*Reserved*
101000	SSM_L3_miss_mtag_remote	101000	L3_rd_miss
101001	SSM_L3_miss_remote	101001	Re_L2_miss
101010	SW_pf_exec	101010	IC_miss_cancelled
101011	SW_pf_str_exec	101011	DC_wr_miss
101100	SW_pf_dropped	101100	L3_hit_I_state_sh
101101	SW_pf_L2_installed	101101	SI_RTS_src_data
101110	*Reserved*	101110	L2_IC_miss
101111	L2_HW_pf_miss	101111	SSM_new_transaction_sh
110000	*Reserved*	110000	L2_SW_pf_miss
110001	L3_miss	110001	L2_wb
110010	L3_IC_miss	110010	L2_wb_sh
110011	L3_SW_pf_miss	110011	L2_snoop_cb_sh
110100	L3_hit_other_half	110100	*Reserved*
110101	L3_wb	110101	*Reserved*
110110	L3_wb_sh	110110	*Reserved*
110111	L2L3_snoop_cb_sh	110111	*Reserved*
111000-111111	*Reserved*	110000-111111	*Reserved*

6

IEEE 754-1985 Standard

The implementation of the floating-point unit for standard and non-standard operating modes is described in this chapter. Debug and diagnostic support are defined in these sections:

Chapter Topics

6.1 Floating-Point Operations

Floating-point operations (FPops) include the algebraic operations and usually do not include the specially treated floating-point load/store, FBfcc, or the VIS instructions. The `FABS`, `FNEG`, and `FMOV` instructions are also treated separately from the algebraic operations.

6.1.1 Rounding Mode

The rounding mode of the floating-point unit is determined either by the FSR.RD bit when in standard rounding mode or by the GSR.IRND bit when in interval arithmetic rounding mode. The rounding direction affects the result after any under or overflow condition is detected. Underflow is detected before rounding.

TABLE 6-1 Rounding Direction

FSR.RD	Round Toward
0	Nearest (even, if tie)
1	0
2	+ ∞
3	– ∞

6.1.2 Non-standard Floating-Point Operating Mode

The processor supports a non-standard floating-point mode to facilitate the handling of subnormals by the hardware, thus avoiding a software trap to supervisor software. The floating-point operating mode is controlled by the FSR.NS bit.

- When FSR.NS = 1, non-standard mode is selected.
- When GSR.IM = 1, interval arithmetic rounding mode is selected. In that case, the processor will be in standard mode regardless of the FSR.NS bit.

6.1.3 Memory and Register Data Images

Floating-point values are represented in the floating-point (f) registers in the same way that they are represented in memory. Any conversions for ALU operations are completed within the floating-point execution unit. Load and store operations do not modify the register value.

VIS™ instructions (logical and move/copy operations) can be used with values generated by the floating-point unit.

6.1.4 Subnormal Operations

Subnormal operations include operations with subnormal number operands and operations without subnormal number operands that generate a subnormal number result. The floating-point unit response to subnormal numbers is described in *Subnormal Operations* on page 138.

6.1.5 FSR.CEXC and FSR.AEXC Updates

The current exception (CEXC) and accrued exception (AEXC) fields in the FSR register are described in *IEEE Traps* on page 133. FPops update these fields in the following situations:

- **CEXC** - Only floating-point operations will update CEXC and only when an exceptional condition is detected. All other instructions will leave CEXC unchanged.
- **AEXC** - When an exception is detected and the trap is masked, the FPop will update the appropriate AEXC field of the FSR register.

6.1.6 Prediction Logic

Prediction of overflow, underflow, and inexact traps is used in the hardware. Prediction provides correct results when possible and generates an exception when it is not possible.

Prediction of an inexact value never occurs if one of the operands is a zero, NaN (Not a Number), or infinity. When an inexact prediction occurs and the exception is enabled, system software will properly handle these cases and resume program execution. If the exception is not enabled, the result status is used to update the FSR.AEXC and FSR.CEXC fields of the FSR register.

6.2 Floating-Point Numbers

Floating-point number types and their abbreviations are shown in TABLE 6-2. In general the IEEE 754-1985 Standard reserves exponent field values of all 0s and all 1s to represent special values in the standard's floating-point scheme.

TABLE 6-2 Floating-Point Numbers

Number Type	Abbreviation	Data Representation		
		Sign	Exponent	Fraction
Zero	0	0 or 1	000...000	000...000
Subnormal	SbN	0 or 1	000...000	000...001 to 111...111
Normal	Normal	0 or 1	000...001 to 111...110	000...000 to 111...111
Infinity	Infinity	0 or 1	111...111	000...000
Signaling NaN	SNaN	0 or 1	111...111	0xx...xxx
Quiet NaN	QNaN	0 or 1	111...111	1xx...xxx

6.2.1 Zero

Zero is not directly representable if the straight format is followed. This limitation is due to the assumption of a leading 1. To allow the number zero to yield a value of zero, the fraction (or mantissa) must be exactly zero. Therefore the number zero is a special case with exponent and fraction fields of zero. Note that -0 and +0 are considered to be distinct values, even though they both compare as equal.

6.2.2 Subnormal

If the exponent field is all 0's and the fraction field is non-zero, the value is a subnormal (denormalized) number. These numbers do not have an assumed leading 1 before the binary point. For single precision, these numbers are represented as $(-1)^s \times 0.f \times 2^{-126}$. In double precision, the representation is $(-1)^s \times 0.f \times 2^{-1022}$. In both cases, s is the sign bit and f is the fraction. Exponent and fraction fields of all 0s are the special representation of the number zero. From this point of view, the number zero can be considered a subnormal.

6.2.3 Infinity

The values -infinity and +infinity are represented with an exponent field of all 1s and a fraction field of all 0s. The sign bit distinguishes between positive and negative infinities. The infinity representation is important because it allows operations to continue past overflow. Operations dealing with infinities are well defined by the IEEE 754-1985 Standard.

6.2.4 Not a Number (NaN)

The value NaN (Not a Number) is used to represent values that do not represent real numbers. The NaN exponent field is all 1s and the fraction field is non-zero. There are two categories of NaN:

- **QNaN (quiet NaN)** is a NaN with the most significant fraction field bit set. QNaN is allowed to freely propagate through most arithmetic operations. QNaN tends to appear when an operation produced mathematically undefined results.
- **SNaN (signaling NaN)** is a NaN with the most significant fraction field bit clear. SNaN is used to signal an exception when it appears out of an operation being executed.

Semantically, QNaN denotes indeterminate operations, while SNaN indicates invalid operations.

6.2.5 Floating-Point Number Line

The floating-point number line in FIGURE 6-1 represents the floating-point numbers used in the processor.

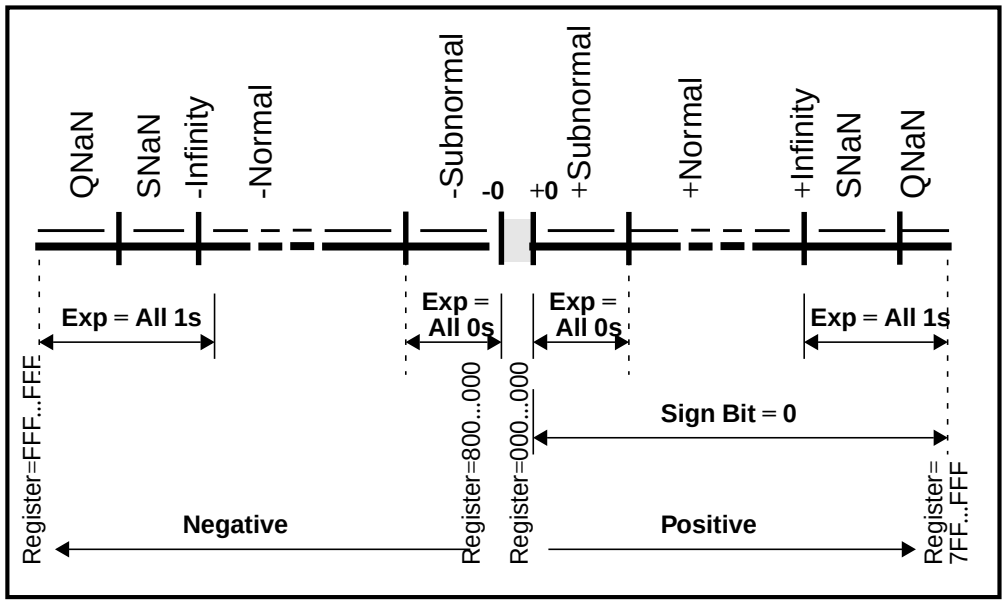

FIGURE 6-1 Floating-Point Number Line

6.3 IEEE Operations

The response of each operation to operands with 0, normal, infinite, and NaN numbers are described in this section. The response to subnormal numbers is described in *Subnormal Operations* on page 138.

The result of each operation is concluded by one of the following:

- A number is written to the destination f register (rd).
- A number is written to the destination register and an IEEE flag is set.
- An IEEE flag is set and an IEEE trap is generated (rd is unchanged).

Each instruction is defined with one or more operands. Most instructions generate a result. The FCMP{E} instruction does not generate a result; it sets the fccN bits instead.

6.3.1 Addition

TABLE 6-3 Floating-Point Addition

ADDITION Instruction **FADD *rs1, rs2 [rs2, rs1] → rd***	**Result from the operation includes one or more of the following:** • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/overflow can occur.			
	Masked Exception, TEM = 0		**Enabled Exception, TEM = 1**	
	Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
+0, +0	+0	None set.	+0	None set.
+0, -0	+0 (FSR.RD = 0,1,2) -0 (FSR.RD = 3)	None set.	+0 (FSR.RD = 0,1,2) -0 (FSR.RD = 3)	None set.
-0, -0	–0	None set.	-0	None set.
±0, +Normal	+Normal	None set.	+Normal	None set.
±0, -Normal	–Normal	None set.	-Normal	None set.
±0, +Infinity	+Infinity	None set.	+Infinity	None set.
±0, -Infinity	–Infinity	None set.	-Infinity	None set.
±Normal, +Infinity	+Infinity	Asserts ofc, ofa, nvc, nva.	No	Asserts ofc, nvc. IEEE trap[1] enabled.
±Normal, -Infinity	–Infinity	Asserts ofc, ofa, nvc, nva.	No	Asserts ofc, nvc. IEEE trap enabled.
+Normal, +Normal	Can overflow. See 6.5.3.		Can overflow. See 6.5.3.	
+Normal, -Normal	±Normal		Normal	
–Normal, +Normal	±Normal		Normal	
–Normal, -Normal	Can underflow. See 6.5.4.		Can underflow. See 6.5.4.	
+Infinity, +Infinity	+Infinity	None set.	+Infinity	None set.
+Infinity, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
–Infinity, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
–Infinity, -Infinity	–Infinity	None set.	-Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754.*

6.3.2 Subtraction

TABLE 6-4 Floating-Point Subtraction

SUBTRACTION Instruction rs_1 - rs_2 **FSUB** rs_1, $rs_2 \rightarrow rd$	**Result from the operation includes one or more of the following:** • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/overflow can occur.			
	Masked Exception, TEM = 0		**Enabled Exception, TEM = 1**	
	Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
+0, +0	+0	None set.	+0	None set.
+0, -0	-0	None set.	-0	None set.
-0, +0	-0	None set.	-0	None set.
-0, -0	+0	None set.	+0	None set.
±0, +Normal	-Normal	None set.	-Normal	None set.
±0, -Normal	+Normal	None set.	+Normal	None set.
±0, +Infinity	-Infinity	None set.	-Infinity	None set.
±0, -Infinity	+Infinity	None set.	+Infinity	None set.
±Normal, +Infinity	-Infinity	Asserts ufc, nvc, ufa, nva.	No	Asserts ufc, nvc. IEEE trap[1] enabled.
±Normal, -Infinity	+Infinity	Asserts ufc, nvc, ufa, nva.	No	Asserts ofc, nvc. IEEE trap enabled.
+Normal, -Normal	Can overflow. See 6.5.3.		Can overflow. See 6.5.3.	
+Normal, +Normal	±Normal	None set.	±Normal	None set.
-Normal,+Normal	Can underflow. See 6.5.4.		Can underflow. See 6.5.4.	
-Normal,–Normal	Can underflow. See 6.5.4.		Can underflow. See 6.5.4.	
+Infinity, [±0, ±Normal]	+Infinity	None set.	+Infinity	None set.
-Infinity, [±0, ±Normal]	-Infinity	None set.	-Infinity	None set.
+Infinity, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
+Infinity, -Infinity	+Infinity	None set.	+Infinity	None set.
-Infinity, +Infinity	-Infinity	None set.	-Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754.*

6.3.3 Multiplication

TABLE 6-5 Floating-Point Multiplication

MULTIPLICATION Instruction FMUL rs_1, rs_2 [rs_2, rs_1] → rd	Result from the operation includes one or more of the following: • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/overflow can occur.			
	Masked Exception, TEM = 0		Enabled Exception, TEM = 1	
	Destination Register Written (rd)	Flag(s)	Destination Register Written (rd)	Flag(s), Trap
+0, [+0|+Normal]	+0	None set.	+0	None set.
+0, [-0|-Normal]	-0	None set.	-0	None set.
-0, [+0|+Normal]	-0	None set.	-0	None set.
-0, [-0|-Normal]	+0	None set.	+0	None set.
+0, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap[1] enabled.
+0, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-0, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-0, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
±Normal, ±Normal	Can underflow/ overflow. See 6.5.		Can underflow/ overflow. See 6.5.	
[+Normal|+Infinity], +Infinity	+Infinity	None set.	+Infinity	None set.
[+Normal|+Infinity], -Infinity	-Infinity	None set.	-Infinity	None set.
[-Normal|-Infinity], +Infinity	-Infinity	None set.	-Infinity	None set.
[-Normal|-Infinity], -Infinity	+Infinity	None set.	+Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754*.

6.3.4 Division

TABLE 6-6 Floating-Point Division

DIVISION Instruction rs_1 rs_2 FDIV rs_1, rs_2 → rd	Result from the operation includes one or more of the following: • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/overflow can occur.			
	Masked Exception, TEM = 0		Enabled Exception, TEM = 1	
	Destination Register Written (rd)	Flag(s)	Destination Register Written (rd)	Flag(s), Trap
±0, ±0	sign=0, expo=111...111, frac=111...111 (QNaN)	Asserts nvc, nva.	No	Asserts nvc. IEEE trap[1] enabled.
±0, ±Normal	±0	None set.	±0	None set.
±0, ±Infinity	±0	None set.	±0	None set.
+Normal, +0	+Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
+Normal, -0	-Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
-Normal, +0	-Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
-Normal, -0	+Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
±Normal, ±Normal	Can underflow/overflow. See 6.5.		Can underflow/overflow. See 6.5.	
±Infinity, ±Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
+Infinity, +Normal	+Infinity	None set.	+Infinity	None set.
+Infinity, -Normal	-Infinity	None set.	-Infinity	None set.
-Infinity, +Normal	-Infinity	None set.	-Infinity	None set.
-Infinity, -Normal	+Infinity	None set.	+Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754*.

6.3.5 Square Root

TABLE 6-7 Floating-Point Square Root

SQUARE ROOT Instruction **sq root of rs_2** **FSQRT $rs_2 \rightarrow rd$**	**Result from the operation includes one or more of the following:** • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/Overflow can occur.			
	Masked Exception, TEM = 0		**Enabled Exception, TEM = 1**	
	Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
+0	+0	None set.	+0	None set.
-0	-0	Asserts nvc, nva.	No	Asserts nvc. IEEE trap[1] enabled.
+Normal	Can underflow/overflow. See 6.5.		Can underflow/overflow. See 6.5.	
[-Normal|-Infinity]	QNaN (sign=0, expo=111...111, frac=111...111)	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
+Infinity	+ Infinity	None set.	+ Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754*.

6.3.6 Compare

Two f registers are compared. The result of the compare is reflected in the fccN bits of the FSR registers. The FCMPE version of the instruction relates to subnormal operations. See TABLE 6-16 on page 137.

TABLE 6-8 Number Compare

Floating-Point NUMBER COMPARE Instruction **FCMP{E} rs_1, rs_2**	**Result from the operation includes one or more of the following:** • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • The fcc bit set.			
	Masked Exception, TEM = 0		**Enabled Exception, TEM = 1**	
	Condition Code Setting (*fccN*)	**Flag(s)**	**Condition Code Setting (*fccN*)**	**Flag(s), Trap**
+0, +0	fcc=0 ($rs_1 = rs_2$)	None set.	fcc=0 ($rs_1 = rs_2$)	None set.
-0, -0	fcc=0 ($rs_1 = rs_2$)	None set.	fcc=0 ($rs_1 = rs_2$)	None set.
+0, [+Normal|+Infinity]	fcc=1 ($rs_1 < rs_2$)	None set.	fcc=1 ($rs_1 < rs_2$)	None set.
-0, [-Normal|-Infinity]	fcc=0 ($rs_1 = rs_2$)	None set.	fcc=0 ($rs_1 = rs_2$)	None set.
-0, [+0|+Normal|+Infinity]	fcc=1 ($rs_1 < rs_2$)	None set.	fcc=1 ($rs_1 < rs_2$)	None set.
+0, [-0|-Normal|-Infinity]	fcc=2 ($rs_1 > rs_2$)	None set.	fcc=2 ($rs_1 > rs_2$)	None set.
±Normal, ±Normal	=, >, or <	None set.	=, >, or <	None set.

6.3.7 Precision Conversion

TABLE 6-9 Precision Conversion

PRECISION CONVERSION Operations **single operand** **FsTOd $rs_2 \rightarrow rd$** **FdTOs $rs_2 \rightarrow rd$**	**Result from the operation includes one or more of the following:** • Number in `f` register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/Overflow can occur.			
	Masked Exception, TEM = 0		**Enabled Exception, TEM = 1**	
	Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
FsTOd ±0 FdTOs ±0	±0	None set.	±0	None set.
FsTOd ±Normal	Normal	None set.	±Normal	None set.
FdTOs ±Normal	Can underflow/overflow. See 6.4.		Can underflow/overflow. See 6.4.	
FsTOd ±Infinity FdTOs ±Infinity	±Infinity	None set.	±Infinity	None set.

Examples:

- `FsTOd` (7FD1.0000) = 7FFA.2000.0000.0000
- `FsTOd` (FDD1.0000) = FFFA.2000.0000.0000
- `FdTOs` (7FFA.2000.0000.0000) = 7FD1.0000
- `FdTOs` (FFFA.2000.0000.0000) = FFD1.0000

6.3.8 Floating-Point to Integer Number Conversion

TABLE 6-10 Floating-Point to Integer Number Conversion

Floating-Point to Integer NUMBER CONVERSION Instruction **single operand** **FsTOi** $rs_2 \rightarrow rd$ **FsTOx** $rs_2 \rightarrow rd$ **FdTOi** $rs_2 \rightarrow rd$ **FdTOx** $rs_2 \rightarrow rd$		**Result from the operation includes one or more of the following:** • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/Overflow can occur.			
		Masked Exception, TEM.NVM = 0		**Enabled Exception, TEM.NVM = 1**	
		Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
SP/DP Int	+0	000...000	None set.	000...000	None set.
	-0	111...111	None set.	111...111	None set.
	+Infinity	011...111	None set.	No	Asserts nvc. IEEE trap[1] enabled.
	-Infinity	100...000	None set.	No	Asserts nvc. IEEE trap enabled.
SP Int	+Normal < 2^{31}	Integer representation of the normal number	None set.	Integer representation of the normal number	None set.
	+Normal ≥ 2^{31}	011...111	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
	-Normal > – [2^{31} + 1]	Integer representation of the normal number	None set.	Integer representation of the normal number	None set.
	-Normal ≤ – [2^{31} + 1]	100...000	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
DP Int	+Normal < 2^{63}	Integer representation of the normal number	None set.	Integer representation of the normal number	None set.
	+Normal ≥ 2^{63}	011...111	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
	-Normal > – [2^{63} + 1]	Integer representation of the normal number	None set.	Integer representation of the normal number	None set.
	-Normal ≤ – [2^{63} + 1]	100...000	None set.	100...000	None set.

1.IEEE trap means *fp_exception_IEEE_754.*

6.3.9 Integer to Floating-Point Number Conversion

TABLE 6-11 Integer to Floating-Point Number Conversion

Integer to Floating-Point NUMBER CONVERSION Instruction **single operand** **FiTOs $rs_2 \rightarrow rd$** **FiTOd $rs_2 \rightarrow rd$** **FxTOs $rs_2 \rightarrow rd$** **FxTOd $rs_2 \rightarrow rd$**		**Result from the operation includes one or more of the following:** • Number in f register. See *Trap Event* on page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/Overflow may occur.			
		Masked Exception, TEM.NXM = 0		**Enabled Exception, TEM.NXM = 1**	
		Destination Register Written (rd)	**Flag(s)**	**Destination Register Written (rd)**	**Flag(s), Trap**
SP/DP	0	0	None set.	0	None set.
SP	+Integer $< 2^{23}$	+Normal	None set.	+Normal	None set.
	+Integer $\geq 2^{23}$	Integer is rounded to 23 MSB and converted.	Asserts nvc, nxc.	No	Asserts nvc. IEEE trap[1] enabled.
	-Integer $> -[2^{23} + 1]$	+Normal	None set.	+Normal	None set.
	-Integer $\leq -[2^{23} + 1]$	Integer is rounded to 23 MSB and converted.	Asserts nvc, nxc.	No	Asserts nvc. IEEE trap enabled.
DP	+Integer $< 2^{52}$	+Normal	None set.	+Normal	None set.
	+Integer $\geq 2^{52}$	Integer is rounded to 52 MSB and converted.	Asserts nvc, nxc.	No	Asserts nvc. IEEE trap enabled.
	-Integer $> -[2^{52} + 1]$	+Normal	None set.	+Normal	None set.
	-Integer $\leq -[2^{52} + 1]$	Integer is rounded to 52 MSB and converted.	Asserts nvc, nxc.	No	Asserts nvc. IEEE trap enabled.

1.IEEE trap means *fp_exception_IEEE_754.*

6.3.10 Copy/Move Operations

Floating-point numbers are not modified by the copy and move instructions: `FMOV`, `FABS`, and `FNEG`. The copy/move instructions will not generate an *unfinished_FPop* or *unimplemented_FPop* exception, but they will generate the *fp_disabled* exception if the floating-point unit is disabled.

The processor performs the appropriate sign bit transformation but will not cause an invalid exception and will not perform a QNaN to SNaN transformation.

The following single-operand instructions use the rs_2 register as the source operand.

6.3.10.1 The `FMOV` Instruction

The `FMOV` instruction:

- Performs f register to f register moves.
- Does not change any bit, regardless of register content.
- Is useful with VIS instructions.

6.3.10.2 The FABS Instruction

The FABS instruction:

- Changes the floating-point/integer sign bit to positive, if needed.
- Does not change any other bit, regardless of register content.

6.3.10.3 The FNEG Instruction

The FNEG instruction:

- Toggles the floating-point/integer sign bit. If 0, changes it to 1; if 1, changes it to 0.
- Does not change any other bit, regardless of register content.

6.3.11 f Register Load/Store Operations

Load/store operations for the f register include the following:

- **Load single floating-point (LDF)** instruction writes to a 32-bit register. This value must be converted to a 64-bit value (FsTOd) for use with double-precision instructions.
- **Load double floating-point (LDDF)** instruction can write to a pair of adjacent 32-bit f registers aligned to an even boundary. LDDF can also write to a 64-bit register. The value must be converted to a 32-bit value (FdTOs) for use with single precision instructions.
- **Two** LDF instructions can be used to load a 64-bit value when the memory address alignment to 64 bits is not guaranteed.
- **Two** STF instructions can be used to store a 64-bit value when the memory address alignment to 64 bits is not guaranteed.

6.3.12 VIS Operations

The floating-point unit must be enabled to execute VIS instructions. VIS instructions do not generate interrupts unless the floating-point unit is disabled. VIS instructions are unaffected by floating-point models.

6.4 Traps and Exceptions

Three trap vectors are defined for floating-point operations:

- *fp_disabled*
- *fp_exception_IEEE_754* (See *IEEE Traps* on page 133.)
- *fp_exception_other*

6.4.1 *fp_disabled* Trap

The floating-point unit can be enabled and disabled.

6.4.2 *fp_exception_other* Trap

The *fp_exception_other* trap occurs when a floating-point operation cannot be completed by the processor (*unfinished_FPop*) or an operation is requested that is not implemented by the processor (*unimplemented_FPop*).

6.4.3 Summary of Exceptions

TABLE 6-12 Floating-Point Unit Exceptions

Trap Description	IEEE Flag	Trap Status	Fault Trap Type	Exception/Trap Vector
Floating-Point unit disabled	None set.	No traps enabled.	None	fp_disabled (020_{16})
Floating-Point operation invalid (IEEE)	nv	IEEE trap enabled.	IEEE_745_exception (FSR.FTT = 1)	fp_exception_IEEE_754 (021_{16})
Floating-Point operation overflow (IEEE)	of			
Floating-Point operation underflow (IEEE)	uf			
Floating-Point operation division by zero (IEEE)	dz			
Floating-Point operation inexact (IEEE)	nx			

6.4.4 Trap Event

When a floating-point exception causes a trap, the trap is precise. The traps that are affected are checked in TABLE 6-13.

TABLE 6-13 Response to Traps

Exception Event → Resulting Action ↓	fp_disabled	fp_exception_other		fp_exception_IEEE_754
		unimplemented_FPop	unfinished_FPop	
Address of instruction that caused the trap is put in the PC and pushed onto the trap stack.	✓	✓	✓	✓
The destination f register (rd) is unchanged from its state prior to the execution of the instruction that caused the trap.	✓	✓	✓	✓
The floating-point condition codes (fccN) are unchanged.	✓	✓	✓	✓
The FSR.AEXC field is unchanged.	✓	✓	✓	✓
The FSR.CEXC field is unchanged.	✓	✓	✓	Appropriate bit is set to 1.
The FSR.FTT field is set to:	No change	3	2	1

6.4.5 Trap Priority

Traps generated by floating-point exceptions (*fp_disabled*, *fp_exception_IEEE_754*, and *fp_exception_other*) are prioritized.

6.5 IEEE Traps

Underflow, overflow, inexact, division-by-zero, and invalid IEEE traps are supported in standard and non-standard modes. These traps are listed in TABLE 6-12 and operate according to the IEEE 754-1985 Standard.

6.5.1 IEEE Trap Enable Mask (TEM)

Individual IEEE traps (*nv*, *of*, *uf*, *dz*, and *nx*) are masked by the FSR.TEM bits. When a trap is masked and an exception is detected, the appropriate FSR.CEXC bit(s) are set and the destination register is written with data shown in TABLE 6-3, TABLE 6-4, TABLE 6-5, TABLE 6-6, TABLE 6-7, TABLE 6-8, and TABLE 6-9.

6.5.2 IEEE Invalid (*nv*) Trap

The IEEE invalid exception (*nv*) is generated when either the source operand to a mathematical operation is a NaN (signaling or quiet) or the result of a mathematical operation does not fit in the integer format. The *nv* trap for an invalid case can be masked using the FSR register.

6.5.3 IEEE Overflow (*of*) Trap

When an overflow occurs, the inexact flag is also set.

- If an overflow occurs *and* the IEEE overflow (*of*) and invalid (*nv*) traps are enabled (FSR.TEM.NVM = 1), an *fp_exception_IEEE_754* is generated.
- If the overflow trap is masked and the operation is valid, the destination register (rd) receives infinity.

The overflow trap is caused when the result of an arithmetic operation exceeds the range supported by the floating-point or integer number precision. This condition can happen in many different cases as listed in the tables of this section.

6.5.4 IEEE Underflow (*uf*) Trap

When a normal number underflows, the inexact flag is also set. Underflow is detected before rounding. The underflow condition leads to a subnormal result unless gross underflow is detected. In that case, the result is 0 and the inexact flag is asserted. Underflow is discussed in detail in *Underflow Operation* on page 134.

6.5.5 IEEE Divide-by-Zero (*dz*) Trap

When a number is divided by zero, the divide-by-zero flag is asserted and an *IEEE_exception* is generated, if enabled. The *dz* flag and trap can only be generated by the `FDIV` instruction.

6.5.6 IEEE Inexact (*nx*) Trap

When an inexact condition occurs, the processor sets the FSR.AEXC.NXA and/or the FSR.CEXC.NXC bits whenever the rounded result of an operation differs from the precise result.

- The inexact flag is asserted for most overflow or underflow conditions.
- The inexact trap is caused when the ideal result cannot fit into the destination format. This occurs for:
  - Most square root operations
  - Some add, subtract, multiply, and divide operations
  - Some number and precision conversion operations

TABLE 6-14 Floating-Point ↔ Integer Conversions That Generate Inexact Exceptions

Instruction	Conversion Description	Unmasked Exception, TEM = 0	Masked Exception, TEM = 1
FsTOi FdTOi	Floating-Point to 32-bit integer when the source operand is not between $-(2^{31} - 1)$ and 2^{31}, then the result is inexact.	Integer number, nx	nx IEEE trap
FsTOx FdTOx	Floating-Point to 64-bit integer when the source operand is not between $-(2^{63} - 1)$ and 2^{63}, then the result is inexact.	Integer number, nx	nx IEEE trap
FiTOs	Integer to Floating-Point when the 32-bit integer source operand magnitude is not exactly representable in single precision (23-bit fraction).[1]	Single Precision Normal, nx	nx IEEE trap
FxTOs	Integer to Floating-Point when the 64-bit integer source operand magnitude is not exactly representable in single precision (23-bit fraction).[1]	Single Precision Normal, nx	nx IEEE trap
FxTOd	Integer to Floating-Point when the 64-bit integer source operand magnitude is not exactly representable in double precision (52-bit fraction).[2]	Double Precision Normal, nx	nx IEEE trap

1.Even if the operand is $> 2^{24} - 1$, if enough of its trailing bits are zeros, it may still be exactly representable.

2.Even if the operand is $> 2^{53} - 1$, if enough of its trailing bits are zeros, it may still be exactly representable.

6.6 Underflow Operation

Underflow occurs when the result of an operation (before rounding) is less than that representable by a normal number.

After rounding, the tiny number (underflow) is usually represented by a subnormal number, but may equal the smallest normal number if the unrounded result is just below the range of normal numbers and the rounding mode (specified in FSR.RD) moves the value into the normal number range. The underflow result will be zero, subnormal, or the smallest normal value.

Note – The floating-point unit does not support exponent wrapping for underflow or overflow.

6.6.1 Trapped Underflow

The floating-point unit will trap on underflow if the FSR.TEM.UFM bit is set to 1. Because tininess is detected before rounding, trapped underflow occurs when the exact unrounded result has a magnitude between zero and the smallest representable normal number in the precision of the destination format. When underflow is trapped, the destination and other registers are left unchanged. See *Trap Event* on page 132.

6.6.2 Untrapped Underflow

If the FSR.TEM.UFM bit is set to 0, the floating-point unit will not generate an underflow trap when an underflow occurs.

If the result causes an underflow and the result after rounding is exact, the floating-point unit will not generate an inexact trap.

Tininess detection before rounding is summarized in TABLE 6-15 using the following terms:

- ***u*** is the unrounded (exact) value of the result.
- ***r*** is the rounded value of ***u*** which occurs when there is no trap generated.
- Underflow is when: $0 < |\boldsymbol{u}| <$ smallest normal number.

TABLE 6-15 Underflow Exception Summary

Underflow:		**enabled** (UFM = 1)	**masked** (UFM = 0)	**masked** (UFM = 0)
Inexact:		**don't care** (NXM = x)	**enabled** (NXM = 1)	**masked** (NXM = 0)
u = *r* exact result	***r*** is minimum normal	None	None	None
	r is subnormal	Asserts ufc. IEEE trap[1] enabled.	None	None
	r is zero	None	None	None
u r inexact result	***r*** *is minimum normal*	Asserts ufc. IEEE trap enabled.	Asserts nxc. IEEE trap enabled.	Asserts ufc, ufa.
	r is subnormal	Asserts ufc. IEEE trap enabled.	Asserts nxc. IEEE trap enabled.	Asserts ufc, ufa.
	r is zero	Asserts ufc. IEEE trap enabled.	Asserts nxc. IEEE trap enabled.	Asserts ufc, ufa.

1.IEEE trap means *fp_exception_IEEE_754.*

6.7 IEEE NaN Operations

When a NaN operand appears or a NaN result is generated and the invalid (*nv*) trap is enabled (FSR.TEM.NVM = 1), the *fp_exception_IEEE_754* occurs.

If the invalid (*nv*) trap is masked (FSR.TEM.NVM = 0), a signaling NaN operand is transformed into a quiet NaN. A quiet NaN operand will propagate to the destination register. Subnormal operations are described in TABLE 6-16.

Whenever a NaN is created from non-NaN operands, the *nv* flag is set.

6.7.1 Signaling and Quiet NaNs

SNaN and QNaN numbers are unsigned. The sign bit is an extension of the NaN's fraction field.

SNaN operands propagate to the destination register as a QNaN result when the *nv* exception is masked. All operations with NaN operands keep the sign bit unchanged including an `FSQRT` operation.

NaNs are generated for the conditions shown in *NaN Results From Operands Without NaNs* on page 137.

6.7.2 SNaN to QNaN Transformation

The signaling to quiet NaN transformation causes the following events:

- The most significant bits of the operand fraction are copied to the most significant bits of the result's fraction.
  - In conversion to a narrower format, excess low-order bits of the operand fraction are discarded.
  - In conversion to a wider format, unwritten low-order bits of the result fraction are set to 0.
- The quiet bit (the most significant bit of the result fraction) is set to 1. The NaN transformation produces a QNaN.
- The sign bit is copied from the operand to the result without modification.

6.7.3 Operations With NaN Operands

Operations with NaN operands may assert the IEEE invalid trap flag (*nv*). These operations are listed in TABLE 6-16.

If the invalid trap is enabled (FSR.TEM.NVM = 1), a trap event occurs as described in *Trap Event* on page 132.

TABLE 6-16 Results From NaN Operands

Operation		Result from the operation includes one or more of the following: • Number in f register. See *Trap Event* note, page 132. • Exception bit set. See TABLE 6-12. • Trap occurs. See abbreviations in TABLE 6-12. • Underflow/Overflow may occur.			
		Masked Exception, TEM.NVM = 0		**Enabled Exception, TEM.NVM = 1**	
		rd or fcc register written	**flag set**	**rd or fcc register written**	**flag set**
One Operand $rs_2 \rightarrow rd$					
Any	QNaN	QNaN See note[1]	None set.	QNaN See note.[1]	None set.
Any	SNaN	SNaN → QNaN See note.[1]	Asserts nvc, nva.	No	Asserts nvc. IEEE trap[2] enabled.
Two Operand rs_1, rs_2 $[rs_2, rs_1] \rightarrow rd$					
FADD, FSUB, FMUL, FDIV	QNaN, QNaN	$QNaN_{rs2}$	None set.	$QNaN_{rs2}$	None set.
	QNaN, anything except SNaN and QNan	QNaN	None set.	QNaN	None set.
	SNaN, SNaN	$SNaN_{rs2}$ → QNaN See note.[1]	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
	SNaN, anything except SNaN	SNaN → QNaN See note.[1]	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
FCMPEs,d	[SNaN or QNaN], anything	fcc=3 (unordered)	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
FCMPs,d	SNaN, anything	fcc=3 (unordered)	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
FCMPs,d	QNaN, anything except SNaN	fcc=3 (unordered)	None set.	fcc=3 (unordered)	None set.

1.For the `Fs,dTOs,d` and other instructions, see *SNaN to QNaN Transformation* on page 136.

2.IEEE trap means *fp_exception_IEEE_754.*

Note – Notice from TABLE 6-16 that the *compare and cause exception if unordered* instruction (`FCMPEs,d`) will cause an invalid (*nv*) exception if either operand is a quiet or signaling NaN. The `FCMP` instruction causes an exception for signaling NaNs only.

6.7.4 NaN Results From Operands Without NaNs

The following operations generate NaNs. See *IEEE Operations* on page 122 for details.

- `FSQRT` [-Normal, or -0]
- `FDIV` ±0

6.8 Subnormal Operations

The handling of subnormals is different for standard and non-standard floating-point modes. The handling of operands and results are described separately in the following sections.

6.8.1 Response to Subnormal Operands

The floating-point unit responds to subnormal operands and results by either handling the result in the hardware or by generating an *fp_exception_other (*with FSR.FTT = 2*, unfinished_FPop*).

The response of the floating-point unit depends on its operating mode, which is controlled by the FSR.NS bit.

6.8.1.1 Standard Mode

In standard mode, the floating-point unit in most cases traps when a subnormal operand is detected or a subnormal result is generated. In this situation, the system software must perform or complete the operation.

The floating-point unit supports the following in standard mode:

- Some cases of subnormal operands are handled in hardware.
- Gross underflow results are supported in hardware for `FdTOs, FMULs,d,` and `FDIVs,d` instructions.

6.8.1.2 Non-standard Mode

In non-standard mode, the floating-point unit in most cases flushes subnormal operands to 0 (with the same sign as the subnormal number) then uses the value in the operation. Subnormal results (results that would otherwise cause an *unfinished_FPop*) are also flushed to 0 in non-standard mode.

If the higher priority invalid operation or a divide-by-zero condition occurs, the corresponding bits are asserted in the FSR.CEXC register field.

- If the trap is enabled (FSR.TEM), an *fp_exception_IEEE_754* trap occurs.
- If the trap is disabled, the corresponding bits are also flagged in the FSR.AEXC register field.

If neither the invalid nor divide-by-zero condition occurs, an inexact condition plus any other detected floating-point exception conditions are flagged in the FSR.CEXC register field.

- If an IEEE trap is enabled (FSR.TEM), an *fp_exception_IEEE_754* trap occurs.
- If the trap is disabled, the corresponding condition(s) are also flagged in the FSR.AEXC register field.

6.8.2 Subnormal Number Generation

Handling of the FMULs, FMULd, FDIVs, FDIVd, and FdTOs instructions requires further explanation using the following terms:

- $Sign_r$ = sign of result,
- RT_{Eff} = round nearest effective truncate or round truncate
- RP = round to +infinity
- RM = round to -infinity
- RND = FSR.RD
- Er = biased exponent result
- E_{rb} = the biased exponent result before rounding
- $E(rs_1)$ = biased exponent of rs_1 operand
- P_rs_1 = precision of the rs_1 operand

The value of the constants depends on precision type as shown in TABLE 6-17.

TABLE 6-17 Subnormal Handling Constants per Destination Register Precision

Destination Register Precision (P)	Number of Bits in Exponent Field	Exponent Bias (E_{BIAS})	Exponent Max (E_{MAX})	Exponent Gross Underflow (E_{GUF})
Single	8	127	255	-24
Double	11	1023	2047	-53

- For FMULs and FMULd: $E_r = E(rs_1) + E(rs_2) - E_{BIAS}$
- For FDIVs and FDIVd: $E_r = E(rs_1) - E(rs_2) + E_{BIAS} - 1$

When two normal operands of FMULs,d and FDIVs,d generate a subnormal result, the E_{rb} is calculated using the algorithm shown below.

```
If (fraction_msb overflows)    // i.e., fraction_msb >= 1'd2
{
        Erb = Er + 1
}
ELSE
{
        Erb = Er
}
```

- For FdTOs, $E_r = E(rs_2) - E_{BIAS}(P\_rs_2) + E_{BIAS}(P\_rd)$, where P_rs_2 is the larger precision of the source and P_rd is the smaller precision of the destination.

Even though $0 \le [E(rs_1)$ or $E(rs_2)] \le 255$ for each single precision biased operand exponent, the computed biased exponent result (E_r) can be $0 \le E_r \le 255$ or can even be negative. For example, for the FMULs instruction:

- If $E(rs_1) = E(rs_2) = +127$, then $E_r = +127$ (127 + 127 - 127)
- If $E(rs_1) = E(rs_2) = 0$, then $E_r = -127$ (0 + 0 – 127)

6.8.2.1 Overflow Result

If the appropriate trap enable masks are not set (FSR.OFM = 0 *and* FSR.NXM = 0), set the FSR.AEXC and FSR.CEXC overflow and inexact flags as follows: FSR.OFA = 1, FSR.NXA = 1, FSR.OFC = 1, and FSR.NXC = 1. No trap is generated.

If any or both of the appropriate trap enable masks are set (FSR.OFM = 1 *or* FSR.NXM = 1), only an IEEE overflow trap is generated: FSR.FTT = 1. The FSR.CEXC bit that is set follows the SPARC V9 architecture:

- If FSR.OFM = 0 *and* FSR.NXM = 1, then FSR.NXC = 1.
- If FSR.OFM = 1 (independent of FSR.NXM), then FSR.OFC = 1 *and* FSR.NXC = 0.

6.8.2.2 Gross Underflow Zero Result

Result = 0 (with correct sign).

If the appropriate trap enable masks are not set (FSR.UFM = 0 *and* FSR.NXM = 0), set the FSR.AEXC and FSR.CEXC underflow and inexact flags: FSR.UFA = 1, FSR.NXA = 1, FSR.UFC = 1, and FSR.NXC = 1. A trap is not generated.

If either or both of the appropriate trap enable masks are set (FSR.UFM = 1 *or* FSR.NXM = 1), only an IEEE underflow trap is generated: FSR.FTT = 1 *and* FSR.CEXC.UF = 1. The FSR.CEXC bit that is set diverges from previous UltraSPARC implementations to follow the SPARC V9 architecture:

- If FSR.UFM = 0 *and* FSR.NXM = 1, then FSR.NXC = 1.
- If FSR.UFM = 1, independent of FSR.NXM, then FSR.UFC = 1 *and* FSR.NXC = 0.

6.8.2.3 Subnormal Handling Override

Result is a QNaN or SNaN.

- Subnormal + SNaN = QNaN, invalid exception generated
  - Standard mode: No *unfinished_FPop*
  - Non-standard mode: No FSR.NX
- Subnormal + QNaN = QNaN, no exception generated
  - Standard mode: No *unfinished_FPop*
  - Non-standard mode: No FSR.NX

Result already generates an exception (divide-by-zero or invalid operation).

- `FSQRT`(number less than zero) = invalid

Result is infinity.

- Subnormal + Infinity = Infinity, no exception generated
  - Standard mode: No *unfinished_FPop*
  - Non-standard mode: No FSR.NX
- Standard mode: Subnormal × Infinity = Infinity

Non-standard mode: Subnormal × Infinity = QNaN with *nv* exception
(Subnormal is flushed to zero.)

Result is zero.

- Subnormal × 0 = 0, no exception generated
  - Standard mode: No *unfinished_FPop*
  - Non-standard mode: No FSR.NX

7

Error Handling

This chapter describes the behavior of the UltraSPARC IV+ processor as viewed by a programmer writing operating systems software, service processor diagnosis or recovery code for the UltraSPARC IV+ processor.

Errors are checked in data arriving at or passing through the processor from the L2-cache, the L3-cache[1], the system bus, and in the MTags arriving from the system bus. In addition, certain cache arrays, protocols and internal logic are also checked for errors.

Error information is logged in the Asynchronous Fault Address Register (AFAR) and Asynchronous Fault Status Register (AFSR). Errors are logged even if their corresponding traps are disabled. First error information is captured in the secondary fault registers.

Chapter Topics

7.1 Error Classes

The three main classes or types of errors that occur are:

1. **Hardware correctable errors:** These errors are corrected automatically by the hardware. For some hardware correctable errors, a trap is optionally generated to log the error condition.
2. **Software correctable errors:** These errors are not corrected automatically by the hardware, but are correctable by the software.
3. **Uncorrectable errors:** These errors are not correctable by either the software or the hardware.

1.This user's manual refers to the level one instruction cache and data cache as I-cache and D-cache, the prefetch cache as P-cache, the write cache as W-cache, the level two cache as L2-cache, and the level three external cache as L3-cache.

These three main classes of errors are handled by normal recovery mechanisms and result in the following types of traps:

- **Precise traps:** These errors are either signaled as software correctable errors or are a form of an uncorrectable error that requires system intervention before normal processor execution can continue.
- **Deferred traps:** These errors are signaled as uncorrectable errors requiring immediate attention, but do not require a system reset.
- **Disrupting traps:** These errors are signaled as requiring logging and clearing, but which do not otherwise affect processor execution.
- **Fatal errors:** These errors are normally handled by a processor reset before continuing.

7.2 Memory Errors

Memory errors include:

- I-cache errors
- D-cache errors
- I-TLB parity error
- D-TLB parity error
- IPB data parity error
- P-cache data parity error
- L2-cache errors
- L3-cache errors
- Errors on the system bus

7.2.1 I-cache Errors

Parity error protection is provided for the I-cache physical and snoop tag arrays, I-cache data array and the instruction prefetch buffer (IPB) data array. The I-cache/IPB is clean, meaning that the value in the I-cache/IPB for a given physical address is always the same as that in some other store in the system. This means that recovery from single bit errors in the I-cache/IPB needs only parity error detection, not full error correcting code. The basic concept is that, when an error is observed, the I-cache/IPB can be invalidated, and the access retried. Both hardware and software methods are used in the UltraSPARC IV+ processor.

Parity error checking in physical tags, snoop tags and I-cache/IPB data is enabled by DCR.IPE in the dispatch control register. When this bit is 0, parity will still be correctly generated and installed in the I-cache/IPB arrays, but will not be checked. Parity error checking is also enabled separately for each line by the line's valid bit. If a line is not valid in the I-cache/IPB, then tag and data parity for that line is not checked.

I-cache physical tag or I-cache/IPB data parity errors are not checked for non-cacheable accesses, or when DCUCR.IC is 0. Although I-cache/IPB errors are not logged in AFSR, trap software can log I-cache/IPB errors.

7.2.1.1 I-cache Physical Tag Errors

I-cache physical tag errors can only occur as the result of an instruction fetch. I-cache entries that have been invalidated by bus coherence traffic do not use the physical tag array.

For each instruction fetch, the microtags select the correct bank of physical tag array. The resulting physical tag data value is then checked for parity errors and can be compared in parallel with the fetched physical address for determining I-cache hit/miss. If there is a parity error in the selected bank of physical tag data, a parity error is reported (regardless whether the result was an I-cache hit or miss). When the trap is taken, TPC will point to the beginning of the fetch group.

An I-cache physical tag error is only reported for instructions that are actually executed. Errors associated with instructions which are fetched, but later discarded without being executed, are ignored. Such an event will leave no trace, but may cause an out-of-sync event in a lockstep system.

7.2.1.2 I-cache Snoop Tag Errors

Snoop cycles from the system bus and from store operations executed in this processor may need to invalidate entries in the I-cache. To discover if the referenced line is present in the I-cache, the physical address from the snoop access is compared, in parallel, with the snoop tags for each of the ways of the I-cache.

In the event that any of the addressed valid snoop tags contains a parity error, the processor will safely report snoop hits. There is just one valid bit for each line, used in both the I-cache physical tags and snoop tags. Clearing this bit will make the next instruction fetch to this line miss in the I-cache and refill the physical and snoop tags and the I-cache entry. Hardware in the I-cache snoop logic does not differentiate between conventional invalidations (where the snoop tag matches) and error invalidations (where there is a snoop tag parity error, hence the hardware cannot tell if the snoop tag matched or not). This is done to ensure that the ordering and timing requirements necessary for the coherence policy are met.

This operation of automatically clearing the valid bits on an error is neither logged in the AFSR nor reported as a trap. Therefore, it is possible to cause undiagnosable out-of-sync events in lockstep systems. In non-lockstep systems, if a snoop tag array memory bit became stuck because of a fault, it would not be detected in normal operation, but would slow down the processor. Depending on the fault, it might be detected in power-on self test code.

I-cache snooping and invalidation continues to operate even if DCUCR.IC is 0. In this case, when DCR.IPE is 1 and a parity error occurs the silent snoop fix-up will apply.

7.2.1.3 I-cache Data Errors

I-cache data parity errors can only be the result of an instruction fetch. I-cache data is not checked as part of a snoop invalidation operation. An I-cache data parity error is determined based on per instruction fetch group granularity[1]. Any instruction (no matter whether it is executed or not) in the fetch group which has parity error will cause an *icache_parity_error* trap on that fetch group. When the trap is taken, TPC will point to the beginning instruction of the fetch group.

If an instruction fetch hits in one way of the I-cache, an *icache_parity_error* trap will be generated for data errors only in the way of the I-cache which hits at this address. Meanwhile, data parity errors in the other ways will be ignored.

1.A fetch group is the collection of instructions which appear on the I-cache outputs at the end of the F Stage. A fetch group is four or fewer consecutive instructions contained within a 32-byte I-cache line

If an instruction fetch misses in the I-cache, then an *icache_parity_error* trap may still be generated if there is a parity error in one of the ways of the set of the I-cache indexed for the instruction fetch. Only one arbitrary way of the I-cache set will be checked for parity errors in the case of an instruction fetch miss. If there are parity errors in one of the other ways, then that is not checked it will not be detected.

An I-cache data parity error for an instruction fetched and later discarded can cause an undiagnosable out-of-sync event in a lockstep system.

7.2.1.4 I-cache Error Recovery Actions

An I-cache physical tag or I-cache/IPB data parity error results in an *icache_parity_error* precise trap. The processor logs nothing about the error in hardware for this event. The trap routine can log the index of the I-cache/IPB error based on the virtual address stored in the TPC.

The trap routine may attempt to discover which associative way caused an error, and whether the physical tags or the data is in error, and sometimes even the bit that is in error, by accessing the I-cache/IPB in its diagnostic address space. However, this attempt depends on the I-cache continuing to return an error for the same access, and the entry not being displaced by coherence traffic. This means that attempting to pinpoint the erroneous bit may often fail.

When the processor takes an *icache_parity_error* trap, both the I-cache and the D-cache are automatically disabled by setting the DCUCR.DC and the DCUCR.IC bits to 0. This prevents recursion in the *icache_parity_error* trap routine when it takes an I-cache miss for an I-cache index which contains an error. It also prevents a D-cache error causing a trap to the *dcache_parity_error* routine from within the *icache_parity_error* routine. When the primary caches are disabled in this way, program accesses can still hit in the L2-cache, so throughput will not be excessively degraded.

Note – In the extremely rare case of an I-cache parity error occurring immediately after a write to the DCU register enabling either the D-cache or I-cache, the DCU register update may take effect after the hardware has automatically disabled caches, resulting in the caches being enabled after the processor vectors to the trap handler for the parity error.

Actions required of the *icache_parity_error* trap routine:

- Log the event
- Clear the I-cache by setting the valid bit for every way at every line to 0 with a write to `ASI_ICACHE_TAG`.
- Clear the IPB by setting the valid bit for every entry of the IPB tag array to 0 with a write to `ASI_IPB_TAG`.
- Clear the D-cache by setting the valid bit for every way at every line to 0 with a write to `ASI_DCACHE_TAG`. Also initialize the D-cache by setting distinct values to `ASI_DCACHE_UTAG` and writing 0 to `ASI_DCACHE_TAG` (both data and parity) of all D-cache tag indexes and ways.
- Clear the P-cache by setting the valid bit for every way at every line to 0 with a write to `ASI_PCACHE_TAG`.
- Re-enable I-cache and D-cache by setting DCUCR.DC and DCUCR.IC to 1.
- Execute `RETRY` to return to the originally faulted instruction.

It is necessary to invalidate the D-cache because any stores the *icache_parity_error* trap routine makes on pending stores in the store queue, while DCUCR.DC is 0, will not write to the D-cache. The D-cache will later contain stale data.

7.2.1.5 I-cache Error Detection Details

I-cache/IPB diagnostic accesses described in *I-cache Data Errors* on page 145 and *Instruction Prefetch Buffer (IPB) Data Errors* on page 156 can be used for diagnostic purposes or testing/development of the I-cache parity error trap handling code. For SRAM manufacturing test, since the parity bits are part of existing SRAM, it is automatically included in the SRAM loop chain of `ASI_SRAM_FAST_INIT` (ASI 0x40).

To allow code to be written to check the operation of the parity detection hardware, the following equations specify which storage bits are covered by which parity bits.

IC_tag_parity = xor(IC_tag[28:0]) [This is equal to xor(PA[41:13])].

IC_snoop_tag_parity = xor(IC_snoop_tag[28:0]) [This is equal to xor(PA[41:13])].

For a non-PC-relative instruction # [IC_predecode[7] == 0]

IC_parity = xor(IC_instr[31:0], IC_predecode[9:7, 5:0])

For a PC-relative instruction # [IC_predecode[7] == 1]

IC_parity = xor(IC_instr[31:11], IC_predecode[9:7, 4:0])

For a non-PC-relative instruction # [IPB_predecode[7] == 0]

IPB_parity = xor(IPB_instr[31:0], IPB_predecode[9:7, 5:0])

For a PC-relative instruction # [IPB_predecode[7] == 1]

IPB_parity = xor(IPB_instr[31:11], IPB_predecode[9:7, 4:0])

The PC-relative instructions are, in SPARC V9 terms, BPcc, Bicc, BPr, CALL, FBfcc and FBPfcc where IC_predecode[7]/IPB_predecode[7] = 1.

To test hardware and software for I-cache parity error recovery, programs can cause an instruction to be loaded into the I-cache by executing it, then use the I-cache diagnostic accesses to flip the parity bit (using `ASI_ICACHE_INSTR`, ASI 0x66), or modify the tags (using `ASI_ICACHE_TAG`, ASI 0x67) or data (using `ASI_ICACHE_DATA`). Upon re-executing the modified instruction, a precise trap should be generated. If no trap is generated, the program should check the I-cache using diagnostic accesses to see whether the instruction has been repaired. This would be a sign that the broken instruction had been displaced from the I-cache before it had been re-executed. Iterating this test can check that each covered bit of the I-cache physical tags and data is actually connected to its parity generator.

Instructions need to be executed in order to discover the value of the predecode bits synthesized by the processor. After the I-cache fill, the predecode bits can be read via `ASI_ICACHE_INSTR`. Instructions can, if needed, then be placed directly in the cache without executing them, by diagnostic accesses.

To test hardware and software for IPB parity error recovery, programs need to load an instruction in the IPB. Depending on the I-cache prefetch stride, the instruction block at the 64-, 128-, or 192-byte offset of the current instruction block will be prefetched into the IPB. Once the instruction block is loaded into the IPB, use the `ASI_IPB_DATA` (ASI 0x69) diagnostic access to flip the parity bit of the instruction in the IPB entry. Upon executing the modified instruction, a precise

trap should be generated. If no trap is generated, the program should check the IPB using ASI_IPB_DATA to see whether the instruction has been repaired. This would be a sign that the broken instruction had been displaced from the IPB before it had been executed. Iterating this test for each entry of the IPB can check that each covered bit of the IPB data is actually connected to its parity generator.

Testing the I-cache snoop tag error recovery hardware is harder, but one example is:

- First, execute multiple instructions that map to the same I-cache tag index, so all the ways of the I-cache are filled with valid instructions.
- Then insert a parity error in one of the I-cache snoop tag entries for this index, using ASI_ICACHE_SNOOP_TAG (ASI 0x68) diagnostic access.
- Then perform a write to an address which maps to the same I-cache tag index, but does not match any of the entries in the I-cache. Diagnostic accesses to the I-cache should confirm that all the ways of the I-cache at this index have either been invalidated or displaced by instructions from other addresses that happened to map to the same I-cache tag index. Again, this can be iterated for all the covered I-cache snoop tag bits.

The parity is computed and stored independently for each instruction in the I-cache. An I-cache parity error is determined based on per instruction fetch group granularity. Unused or annulled instruction(s) are not masked during the parity check. It means that they can still cause an *icache_parity_error* trap.

Note – In the event of the simultaneous detection of an I-cache parity error and an I-TLB miss, the *icache_parity_error* trap is taken. When this trap routine executes RETRY to return to the original code, a *fast_instruction_access_MMU_miss* trap will be generated.

Because the *icache_parity_error* trap routine uses the alternate global register set, recovery from an I-cache parity error is unlikely to be possible if the error occurs in a trap routine which is already using these registers. This is expected to be a low probability event. The *icache_parity_error* trap routine can check to see if it has been entered at other than TL = 1, and reboot the domain if it has.

7.2.1.6 Notes on the Checking Sequence of *icache_parity_error* Trap

TABLE 7-1 and TABLE 7-2 highlight the checking sequence for I-cache tag/data parity errors and IPB data parity error, respectively.

TABLE 7-1 I-cache Tag/Data Parity Errors

Page Cacheable	microtag Hit	Cache Line Valid	Physical Tag Parity Error	Fetch Group Executed	Fetch Group I-cache Data Parity Error	DCR.IPE Bit	Signal a Trap
Yes	No	X	X	X	X	x	No
Yes	Yes	No	X	X	X	x	No
Yes	Yes	Yes	Yes	No	X	x	No
Yes	Yes	Yes	Yes	Yes	X	0	No
Yes	Yes	Yes	Yes	Yes	X	1	Yes
Yes	Yes	Yes	No	No	X	x	No
Yes	Yes	Yes	No	Yes	No	x	No
Yes	Yes	Yes	No	Yes	Yes	0	No
Yes	Yes	Yes	No	Yes	Yes	1	Yes

TABLE 7-2 I-cache Data Parity Error

Page Cacheable	I-cache Miss	IPB Tag Hit	IPB Entry Valid	Fetch Group Executed	Fetch Group IPB Data Parity Error	DCR.IPE Bit	Signal a Trap
Yes	No	X	X	X	X	x	No
Yes	Yes	No	X	X	X	x	No
Yes	Yes	Yes	No	X	X	x	No
Yes	Yes	Yes	Yes	No	X	x	No
Yes	Yes	Yes	Yes	Yes	X	0	No
Yes	Yes	Yes	Yes	Yes	No	1	No
Yes	Yes	Yes	Yes	Yes	Yes	1	Yes

7.2.2 D-cache Errors

Parity error protection is provided for the D-cache physical tag array, snoop tag array and data array. The D-cache is clean, meaning that the value in the D-cache for a given physical address is always the same as that in some other store in the system. This means that recovery from single bit errors in D-cache need only parity error detection, not full error correcting code. The basic concept is that when an error is observed, the D-cache is invalidated, and the access retried. Both hardware and software methods are used in the UltraSPARC IV+ processor.

Parity error checking in physical tags, snoop tags and D-cache data is enabled by DCR.DPE in the dispatch control register. When this bit is 0, parity will still be correctly generated and installed in the D-cache arrays, but will not be checked.

Note – D-cache physical tag or data parity errors are not checked for non-cacheable accesses, or when DCUCR.DC is 0. D-cache errors are not logged in AFSR. Trap software can log D-cache errors.

7.2.2.1 D-cache Physical Tag Errors

D-cache physical tag parity errors can only occur as the result of a load, store, or atomic instruction. Invalidation of D-cache entries by bus coherence traffic does not use the physical tag array.

The physical address of each datum being fetched as a result of a data access instruction (including speculative loads) is compared in parallel with the physical tags in all four ways of the D-cache. A parity error on the physical tag of any of the four ways will result in a *dcache_parity_error* trap when it hits valid and micro tag array.

The D-cache actually only supplies data for load-like operations from the processor. However, physical tag parity is checked for store-like operations as well, because if the D-cache has an entry for the relevant line, it must be updated.

Note – A D-cache physical tag error is only reported for data which is actually used. Errors associated with data which is speculatively fetched, but later discarded without being used, are ignored for the moment. D-cache physical tag parity errors on executed instructions are reported by a precise *dcache_parity_error* trap. See *D-cache Error Recovery Actions* on page 151.

7.2.2.2 D-cache Snoop Tag Errors

Snoop cycles from the system bus may need to invalidate entries in the D-cache. To discover if the referenced line is present in the D-cache, the physical address from the snoop access is compared in parallel with the snoop tags for each of the ways of the D-cache.

In the event that any of the addressed valid snoop tags contains a parity error, the processor hardware will automatically clear the valid bits for all the ways of the D-cache at that tag index. This applies whether the snoop hits in the D-cache or not. There is just one valid bit for each line, used in both the D-cache physical tags and snoop tags. Clearing this bit will make the next data fetch to this line miss in the D-cache and refill the physical and snoop tags and the D-cache entry. Hardware in the D-cache snoop logic ensures that both a conventional invalidation (where the snoop tag matches) and an error invalidation (where there is a snoop tag parity error, so the hardware cannot tell if the snoop tag matched or not) meet the ordering and timing requirements necessary for the coherence policy.

Note – This operation of automatically clearing the valid bits on an error is not logged in the AFSR or reported as a trap. Therefore, it may cause undiagnosable out-of-sync events in lockstep systems. In non-lockstep systems, if a snoop tag array memory bit became stuck because of a fault, it would not be detected in normal operation, but would slow down the processor. Depending on the fault, it might be detected in power-on self test code. D-cache snooping and invalidation continues to operate even if DCUCR.DC is 0. In this case, if DCR.DPE is 1, the fix-up with silent snoop will still apply in the case of a parity error.

7.2.2.3 D-cache Data Errors

D-cache data parity errors can only be the result of a load-like instruction accessing cacheable data. D-cache data is not checked as part of a snoop invalidation.

A D-cache data parity error can only be reported for instructions which are actually executed, not for those speculatively fetched and later discarded. The error will result in a *dcache_parity_error* precise trap.

If a load has a microtag hit for a particular way in the D-cache and there is a D-cache data array parity error in that way, then irrespective of the valid bit, a *dcache_parity_error* trap will be generated. D-cache data array errors in any other way will be ignored.

7.2.2.4 D-cache Error Recovery Actions

A D-cache physical tag or data parity error results in a *dcache_parity_error* precise trap. The processor logs nothing about the error in hardware for this event. To log the index of the D-cache which produced the error, the trap routine can disassemble the instruction pointed to by TPC.

The trap routine may attempt to discover the way in error, and whether the physical tag or the data is in error, and maybe even the bit in error, by accessing the D-cache in its diagnostic address space. However, this attempt would depend on the D-cache continuing to return an error for the same access, and the entry not being displaced by coherence traffic. This means that attempting to pinpoint the erroneous bit may often fail.

When the processor takes a *dcache_parity_error* trap, it automatically disables both the I-cache and D-cache by setting DCUCR.DC and DCUCR.IC to 0. This prevents recursion in the *dcache_parity_error* trap routine when it takes a D-cache miss for a D-cache index which contains an error. It also prevents an I-cache error causing a trap to the *icache_parity_error* routine from within the *dcache_parity_error* routine. When the primary caches are disabled in this way, program accesses can still hit in the L2-cache, so throughput will not degrade excessively.

Actions required of the *dcache_parity_error* trap routine:

- Log the event.
- Clear the I-cache by writing the valid bit for every way at every line to 0 with a write to `ASI_ICACHE_TAG`.
- Clear the IPB by writing the valid bit for every entry of the IPB tag array to 0 with a write to `ASI_IPB_TAG`.
- Clear the D-cache by writing the valid bit for every way at every line to 0 with a write to `ASI_DCACHE_TAG`. Also initialize the D-cache by writing distinct values to `ASI_DCACHE_UTAG` and writing 0 to `ASI_DCACHE_DATA` (both data and parity) of all D-cache tag indexes and ways.
- Clear the P-cache by writing the valid bit for every way at every line to 0 with a write to `ASI_Pcache_TAG`.
- Re-enable I-cache and D-cache by writing 1 to DCUCR.DC and DCUCR.IC.
- Execute `RETRY` to return to the originally faulted instruction.

D-cache entries must be invalidated because cacheable stores performed by the *dcache_parity_error* trap routine and any pending stores in the store queue will not update any old data in the D-cache while DCUCR.DC is set to 0. If the D-cache was not invalidated, some data could be stale when the D-cache was re-enabled. Snoop invalidates, however, still happen normally while DCUCR.DC is 0.

The D-cache entries must be initialized, and the correct data parity installed, because D-cache data parity is still checked even when the cache line is marked as invalid. It is possible to write to just one cache line, that which returned the error, but this would require disassembling the instruction pointed to by the TPC for this precise trap. Disassembling the instruction would provide the D-cache tag index from the target address.

D-cache physical tag parity is checked only when DC_valid bit is 1. It is not necessary to write to ASI_DCACHE_SNOOP_TAG. Zero can be written to each tag at ASI_DCACHE_TAG. The fact that the physical tag is the same for every entry, and is different from the snoop tag for every entry, will not be significant, because every entry will be invalid. Snoop tag and physical tag will be reloaded next time the line is used.

7.2.2.5 D-cache Error Detection Details

D-cache diagnostic accesses described in *D-cache Errors* on page 149 can be used for diagnostic purposes or testing/development of the D-cache parity error trap handling code.

To allow code to be written to check the operation of the parity detection hardware, the following equations specify which storage bits are covered by which parity bits.

DC_tag_parity = xor(DC_tag[28:0]) - This is equal to xor(PA[41:13]).

DC_snoop_tag_parity = xor(DC_snoop_tag[28:0]) - This is equal to xor(PA[41:13]).

TABLE 7-3 D-cache Parity Generation for Load Miss Fill and Store Update ***(1 of 2)***

Parity Bit	D-cache Load Miss Fill	D-cache Store Update
DC_data_parity[0]	xor(data[7:0])	xor(data[7:0])
DC_data_parity[1]	xor(data[15:8])	xor(data[15:8])
DC_data_parity[2]	xor(data[23:16])	xor(data[23:16])
DC_data_parity[3]	xor(data[31:24])	xor(data[31:24])
DC_data_parity[4]	xor(data[39:32])	xor(data[39:32])
DC_data_parity[5]	xor(data[47:40])	xor(data[47:40])
DC_data_parity[6]	xor(data[55:48])	xor(data[55:48])
DC_data_parity[7]	xor(data[63:56])	xor(data[63:56])
DC_data_parity[8]	xor(data[71:64])	xor(data[7:0])
DC_data_parity[9]	xor(data[79:72])	xor(data[15:8])
DC_data_parity[10]	xor(data[87:80])	xor(data[23:16])
DC_data_parity[11]	xor(data[95:88])	xor(data[31:24])
DC_data_parity[12]	xor(data[103:96])	xor(data[39:32])
DC_data_parity[13]	xor(data[111:104])	xor(data[47:40])
DC_data_parity[14]	xor(data[119:112])	xor(data[55:48])
DC_data_parity[15]	xor(data[127:120])	xor(data[63:56])
DC_data_parity[16]	xor(data[135:128])	xor(data[7:0])
DC_data_parity[17]	xor(data[143:136])	xor(data[15:8])
DC_data_parity[18]	xor(data[151:144])	xor(data[23:16])
DC_data_parity[19]	xor(data159:152])	xor(data[31:24])
DC_data_parity[20]	xor(data[167:160])	xor(data[39:32])

TABLE 7-3 D-cache Parity Generation for Load Miss Fill and Store Update *(2 of 2)*

Parity Bit	D-cache Load Miss Fill	D-cache Store Update
DC_data_parity[21]	xor(data[175:168])	xor(data[47:40])
DC_data_parity[22]	xor(data[183:176])	xor(data[55:48])
DC_data_parity[23]	xor(data[191:184])	xor(data[63:56])
DC_data_parity[24]	xor(data[199:192])	xor(data[7:0])
DC_data_parity[25]	xor(data[207:200])	xor(data[15:8])
DC_data_parity[26]	xor(data[215:208])	xor(data[23:16])
DC_data_parity[27]	xor(data[223:216])	xor(data[31:24])
DC_data_parity[28]	xor(data[231:224])	xor(data[39:32])
DC_data_parity[29]	xor(data[239:232])	xor(data[47:40])
DC_data_parity[30]	xor(data[247:240])	xor(data[55:48])
DC_data_parity[31]	xor(data[255:248])	xor(data[63:56])

Note – The D-cache data parity check granularity is 8 bits. This is the same for line fill and store update. (Fill data size = data[255:0], store data size = data[63:0].)

To test the hardware or software for D-cache parity error recovery, programs can cause data to be loaded into the D-cache by reading it, then use the D-cache diagnostic accesses to modify the tags (using `ASI_DCACHE_TAG`, ASI 0x47) or data (using `ASI_DCACHE_DATA`, ASI 0x46). Alternatively, the data can be synthesized completely using diagnostic accesses. Upon executing an instruction to read the modified line into a register, a trap should be generated. If no trap is generated, the program should check the D-cache using diagnostic accesses to see whether the entry has been repaired. This would be a sign that the broken entry had been displaced from the D-cache before it had been re-accessed. Iterating this test can check that each covered bit of the D-cache physical tags and data is actually connected to its parity generator.

Testing the D-cache snoop tag error recovery hardware is more extensive. First, load multiple lines of data that map to the same D-cache tag index, so all the ways of the D-cache are filled with valid data. Then insert a parity error in one of the D-cache snoop tag entries for this index, using `ASI_DCACHE_SNOOP_TAG` (ASI 0x44) diagnostic access. Then have another processor perform a write to an address which maps to the same D-cache tag index, but does not match any of the entries in the D-cache. Diagnostic accesses to the D-cache should confirm that all the ways of the D-cache at this index have either been invalidated or displaced by data from other addresses that happened to map to the same D-cache tag index. Again, this can be iterated for all the covered D-cache snoop tag bits.

If an instruction is not executed, whether this is because it is annulled, because a trap prior to the instruction switched the program flow, or because a branch diverted control, then a D-cache data parity error on that instruction cannot cause a *dcache_parity_error* trap.

D-cache data parity is not checked for store-like instructions. D-cache physical tag parity is checked for all store instructions except for `STD` and `STDA`.

Because the primary D-cache has a high ratio of reads to writes, and also because the majority of D-cache writes do overwrite entire 32 bit words, the effect of this reduction in coverage is small.

D-cache data parity is not checked for block stores, which overwrite a number of 64-bit words and recompute parity. Any original parity error is overwritten.

D-cache data parity is not checked for atomic operations. Atomic operation data comes from the L2-cache, not the D-cache.

D-cache data parity is not checked for block load / quad load operations. Data for block load/ quad load never comes from the D-cache.

For load-like instructions, D-cache data and physical tag parity is always checked, except for integer LDD and LDDA. For these instructions, which access two or four 32-bit words, data and physical tag parity is only checked for the first 4-byte (32 bit) word retrieved. Thus, the parity error detected in the second access of integer LDD/LDDA, in which helper is used, is suppressed.

In the event of the simultaneous detection of a D-cache parity error and a D-TLB miss, the *dcache_parity_error* trap is taken. When this trap routine executes RETRY to return to the original code, a *fast_data_access_MMU_miss* trap will be generated.

7.2.2.6 Notes on D-cache Data Parity Error Traps

D-cache Data Parity will not be taken under the following conditions:

- When D-Cache is disabled (DCR's DPE = 0).
- Parity errors detected for any type of Stores.
- Parity errors detected for Block loads.
- Parity errors detected for Atomic.
- Parity errors detected for Quad loads.
- Parity errors detected for integer LDD (second access only or helper access).
- Parity errors detected for Internal ASI loads.

D-Cache Data parity Traps will not be suppressed on Data Parity Error that occurs for load that misses D-Cache or causes other types of recirculates.

7.2.2.7 Notes on D-cache Physical Tag Parity Error Traps

D-cache Physical Tag Parity will not be taken under the following conditions:

- When D-Cache is disabled (DCR's DPE = 0).
- Parity errors detected for a invalid line.
- Parity errors detected with microtag miss.
- Parity errors detected for Block loads.
- Parity errors detected for Quad loads.
- Parity errors detected for integer LDD (second access only or helper access).
- Parity errors detected for Internal ASI loads.

D-cache Data parity Traps will NOT be suppressed on D-cache Tag Parity Errors that occurs for Load that misses D-cache or causes other types of recirculates.

For loads, D-cache Physical Tag Parity error Trap is taken if the following expression is true:

```
((pe[0] & microtag[0] & vtag[0] ) |
 (pe[1] & microtag[1] & vtag[1] ) |
 (pe[2] & microtag[2] & vtag[2] ) |
 (pe[3] & microtag[3] & vtag[3] ))
```

For Stores and Atomics, D-cache Tag Parity error Trap is taken if the following expression is true:

```
((pe[0] & vtag[0] ) |
 (pe[1] & vtag[1] ) |
 (pe[2] & vtag[2] ) |
 (pe[3] & vtag[3] ))
```

The values of the pe's, microtag's and vtag's used in the above expressions are determined as follows:

pe[0] =1 if Parity is detected in Way 0 else it is equal to 0
pe[1] =1 if Parity is detected in Way 1 else it is equal to 0
pe[2] =1 if Parity is detected in Way 2 else it is equal to 0
pe[3] =1 if Parity is detected in Way 3 else it is equal to 0

utag[0] =1 if microtags hits in Way 0 else it is equal to 0
utag[1] =1 if microtags hits in Way 1 else it is equal to 0
utag[2] =1 if microtags hits in Way 2 else it is equal to 0
utag[3] =1 if microtags hits in Way 3 else it is equal to 0

vtag[0] =1 if the line is valid in Way 0 else it is equal to 0
vtag[1] =1 if the line is valid in Way 1 else it is equal to 0
vtag[2] =1 if the line is valid in Way 2 else it is equal to 0
vtag[3] =1 if the line is valid in Way 3 else it is equal to 0

TABLE 7-4 highlights the checking sequence for D-cache tag/data parity errors.

TABLE 7-4 D-cache Tag/Data Parity Errors

Page Cacheable	microtag Hit	Cache Line Valid	Physical tag Hit	D-Cache Tag Parity Error	D-Cache Data Parity Error	DCR's DPE bit	Signal a Trap
Yes	x	x	x	x	x	0	No
Yes	1	1	x	1	x	1	Yes
Yes	1	x	x	x	1	1	Yes

7.2.3 I-TLB Parity Errors

The I-TLB is composed of two structures: the 2-way associative T512 array and the fully associative T16 array. The T512 array has parity protection for both tags and data, while the T16 array is not parity protected.

7.2.3.1 I-TLB Parity Error Detection

Please refer to *I-TLB Parity Protection* on page 304 for details about I-TLB parity error detection.

7.2.3.2 I-TLB parity Error Recovery Actions

When all parity error reporting conditions are met: I-MMU enabled, I-TLB parity enabled, and no translation hit in the T16, a parity error detected on a translation will generate an *instruction_access_exception* and the I-SFSR.FT will be set to 20_{16}. Thus, the *instruction_access_exception* trap handler must check the I-SFSR to determine if the cause of the trap was an I-TLB parity error.

When an I-TLB parity error trap is detected, software must invalidate the corresponding T512 TTE by either executing either a demap-context or a demap-all, and write a new entry with the correct parity.

7.2.4 D-TLB Parity Errors

The D-TLB is composed of three structures: the two 2-way associative TLB array (T512_0 and T512_1) and the fully associative TLB array (T16). The T512 array has parity protection for both tag and data, while T16 array is not parity protected.

7.2.4.1 D-TLB Parity Error Detection

Please refer to *D-TLB Parity Protection* on page 322 for details about D-TLB parity error detection.

7.2.4.2 D-TLB parity Error Recovery Actions

When all parity error reporting conditions are met: D-MMU enabled, D-TLB parity enabled, and no translation hit in T16, a parity error detected on a translation will generate an *data_access_exception* and the SFSR will be set to Fault Type 20_{16}. Thus, the *data_access_exception* trap handler must check the SFSR to determine if the cause of the trap was an D-TLB parity error.

When a D-TLB parity error trap is detected, software must invalidate the corresponding T512 TTE by either a demap-page, demap-context, or demap-all, and write a new entry with correct parity.

7.2.5 Instruction Prefetch Buffer (IPB) Data Errors

Similar to those of the I-cache, IPB data parity errors are only caused by an I-fetch from IPB. IPB data is not checked as part of a snoop invalidation operation.

An IPB data parity error is determined based on per instruction fetch group granularity. If a fetch group is used in the pipe from IPB and if any of the instructions in the fetch group has a parity error, an *icache_parity_error* trap will be taken on that fetch group, irrespective of whether the instruction is executed or not. When the trap is taken, TPC will point to the beginning instruction of the fetch group.

If an I-fetch misses the IPB, an *icache_parity_error* trap will not be generated.

7.2.6 P-cache Data Parity Errors

Parity error protection is provided for the P-cache data array only. P-cache snoop tag array, virtual tag array, and status array are not parity error protected.

P-cache Data Parity trap is taken only when a floating point load hits P-cache and there is a parity error associated with that extended word. Software or hardware prefetch instruction do not generate P-cache Data parity trap.

Parity error checking in P-cache data array is enabled by DCR.PPE in the dispatch control register. When this bit is 0, parity will still be correctly generated and installed in the P-cache data array, but will not be checked.

Note – P-cache data parity errors are not checked while DCUCR.PE is 0.

7.2.6.1 P-cache Error Recovery Actions

P-cache data parity error is reported the same way (same trap type and precise-trap timing) as in D-cache Data Array Parity error (see *D-cache Error Recovery Actions* on page 151).

7.2.6.2 P-cache Error Detection Details

Use `ASI_PCACHE_STATUS_DATA` (ASI 0x30) diagnostic access to access the P-cache data parity bits stored in the P-cache Status Array (see *P-cache Data Parity Errors* on page 157 for details).

To test hardware and software for P-cache data parity error recovery, programs can cause data to be loaded into the P-cache by issuing prefetches, then use the P-cache diagnostic access to modify the parity bits. Alternatively, the data can be synthesized completely using diagnostic accesses. Upon executing an instruction to read the modified line into a register, a trap should be generated. If no trap is generated, the program should check the P-cache using diagnostic accesses to see whether the entry has been repaired. This would be a sign that the broken entry had been displaced from the P-cache before it had been re-accessed. Iterating this test can check that each covered bit of the P-cache data is actually connected to its parity generator.

7.2.7 L2-cache Errors

Both the L2-cache tags and data are internal to the processor and covered by ECC. L2-cache errors can be recovered by hardware measures. Information on errors detected is logged in the AFSR and AFAR.

7.2.7.1 L2-cache Tag ECC Errors

The types of L2-cache tag ECC errors are:

- Hardware corrected (HW_corrected) L2-cache tag ECC errors - single bit ECC errors that are corrected by hardware.
- Uncorrectable L2-cache tag ECC errors - multi-bit ECC errors that are not correctable by hardware or software.

Each L2-cache tag entry is covered by ECC. The tag includes the MOESI state of the line, which implies that tag ECC is checked whether or not the line is valid. Tag ECC must be correct even if the line is not present in the L2-cache.

L2-cache tag ECC checking is enabled by the L2_TAG_ECC_en bit in the L2-cache control register. The processor always generates correct ECC when writing L2-cache tag entries, except when programs use diagnostic ECC tag accesses.

7.2.7.2 Hardware Corrected L2-cache Tag ECC Errors

Hardware corrected errors occur on single bit errors, in tag value or tag ECC, detected as the result of the following transactions:

- Cacheable I-fetches
- Cacheable load-like operations
- Atomic operations
- W-cache exclusive request to the L2-cache to obtain data and ownership of the line in W-cache
- Reads of the L2-cache by the processor in order to perform a writeback to L3-cache, or a copyout to the system bus
- Reads of the L2-cache by the processor to perform an operation placed in the prefetch queue by an explicit software PREFETCH instruction or a P-cache hardware PREFETCH operation
- Reads of the L2-cache tag while performing snoop read, displacement flush, L3-cache data fill, block store

Hardware corrected errors optionally produce an *ECC_error* disrupting trap, enabled by the CEEN bit in the Error Enable Register, to carry out error logging.

Hardware corrected L2-cache tag errors set AFSR.THCE and log the access physical address in AFAR. In contrast to L2-cache data ECC errors, AFSR.E_SYND is not captured.

For hardware corrected L2-cache tag errors, the processor actually writes the corrected entry back to the L2-cache tag array, then retries the original request except for a snoop request. For a snoop request, the Hardware corrected L2-cache state will be returned to the system interface unit, and the processor writes the corrected entry back to L2-cache tag array. For most cases, future accesses to this same tag should see no error. This is in contrast to Hardware corrected L2-cache data ECC errors, for which the processor corrects the data, but does not write it back to the L2-cache. This rewrite correction activity by the processor is to maintain the full required snoop latency and also obey the coherence ordering rules.

7.2.7.3 Uncorrectable L2-cache Tag ECC Errors

An uncorrectable L2-cache tag ECC error may be detected as the result of any operation which reads the L2-cache tags.

All uncorrectable L2-cache tag ECC errors are fatal errors, indicated by the processor asserting its ERROR output pin. The event will be logged in AFSR.TUE or AFSR.TUE_SH and AFAR.

All uncorrectable L2-cache tag ECC errors for tag update or copyout due to foreign bus transactions or snoop request will set AFSR.TUE_SH. All uncorrectable L2-cache tag ECC errors for fill, tag update due to local bus transactions, writeback will set AFSR.TUE.

The response of the system to assertion of the ERROR output pin depends on the system, but is expected to result in a reboot of the affected domain. Error status can be read from the AFSR after a system reset event.

7.2.7.4 L2-cache Data ECC Errors

- Hardware corrected (HW_corrected) L2-cache data ECC errors - single bit ECC errors that are corrected by hardware.
- Software correctable (SW_correctable) L2-cache data ECC errors - L2-cache ECC errors that require software intervention.
- Uncorrectable L2-cache data ECC errors - multi-bit ECC errors that are not correctable by hardware or software.

Depending on the operation accessing the L2-cache the full 64-byte line may be checked for data ECC errors or only a 32-byte portion representing either the lower or upper half of the line may be checked. The cases where only 32 bytes are checked correspond to some reads from the L1-caches that only load 32 bytes.

7.2.7.5 Hardware Corrected L2-cache Data ECC Errors

Hardware corrected ECC errors occur on single bit errors detected as the result of the following transactions:

- W-cache exclusive request accesses to the L2-cache to obtain the line and ownership from L2-cache (AFSR.EDC)
- Reads of the L2-cache by the processor in order to perform a writeback to L3-cache or copyout to the system bus (AFSR.WDC, AFSR.CPC)
- Reads of the L2-cache by the processor to perform an operation placed in the prefetch queue by an explicit software `PREFETCH` instruction (AFSR.EDC) or a P-cache hardware `PREFETCH` operation
- Reads of the L2-cache by the processor to perform an operation block load instruction (AFSR.EDC)

Hardware corrected errors optionally produce a disrupting *ECC_error* trap, enabled by the CEEN bit in the Error Enable Register, to carry out error logging.

Note – For hardware corrected L2-cache data errors, the hardware does not actually write the corrected data back to the L2-cache data array. However, the L2-cache sends corrected data to the W-cache. Therefore, the instruction that creates the single bit error transaction can be completed without correcting the L2-cache data. The L2-cache data may later be corrected in the disrupting *ECC_error* trap.

7.2.7.6 Software Correctable L2-cache Data ECC Errors

Software correctable errors occur on single-bit data errors detected as the result of the following transactions:

- Reads of data from the L2-cache to fill the D-cache.
- Reads of data from the L2-cache to fill the I-cache.

- Performing an atomic instruction.

All these events cause the processor to set AFSR.UCC. A SW_correctable error will generate a precise *fast_ECC_error* trap if the UCEEN bit is set in the Error Enable Register. See *Uncorrectable L2-cache Data ECC Errors* on page 162.

7.2.7.7 Software Correctable L2-cache ECC Error Recovery Actions

The *fast_ECC_error* trap handler should carry out the following sequence of actions to correct an L2-cache tag or data ECC error:

1. Read the address of the correctable error from the AFAR register.
2. Invalidate the entire D-cache using writes to `ASI_DCACHE_TAG` to zero out the valid bits in the tags.
3. Evict the L2-cache line that contained the error. This requires four `LDXA ASI_L2CACHE_TAG` operations using the L2-cache disp_flush addressing mode to evict all four ways of the L2-cache.

A single bit data error will be corrected when the processor reads the data from the L2-cache and writes it to L3-cache to perform the writeback. This operation may set the AFSR.WDC or AFSR.WDU bits. If the offending line was in O, Os, or M MOESI state, and another processor happened to read the line while the trap handler was executing, the AFSR.CPC or AFSR.CPU bit could be set.

A single bit tag error will be corrected when the processor reads the line tag to update it to I state. This operation may set AFSR.THCE.

4. Evict the L3-cache line that contained the error. This requires four `LDXA ASI_L3CACHE_TAG` operations using the L3-cache disp_flush addressing mode to evict all four ways of the L3-cache to memory.

When the line is displacement flushed from L2-cache, if the line is not in I state, then it will be writeback to L3-cache.

5. Log the error.
6. Clear AFSR.UCC, UCU, WDC, WDU, CPC, CPU, THCE, and AFSR_EXT.L3_WDU, L3_CPU.
7. Clear AFSR2 and AFSR2_EXT.
8. Displacement flush any cacheable *fast_ECC_error* exception vector or cacheable *fast_ECC_error* trap handler code or data from the L2-cache to L3-cache.
9. Displacement flush any cacheable *fast_ECC_error* exception vector or cacheable *fast_ECC_error* trap handler code or data from the L3-cache to memory.
10. Re-execute the instruction that caused the error using `RETRY`.

Corrupt data is never stored in the I-cache.

Data in error is stored in the D-cache. If the data was read from the L2-cache as the result of a load instruction, or an atomic instruction, corrupt data will be stored in D-cache. However, if the data was read as the result of a block load instruction, corrupt data will not be stored in D-cache. Store-like instructions never cause *fast_ECC_error* traps directly, just load-like and atomic instructions. Store-like instructions never result in corrupt data being loaded into the D-cache.

The entire D-cache is invalidated because there are circumstances when the AFAR used by the trap routine does not point to the line in error in the D-cache. This can happen when multiple errors are reported, and when an instruction fetch (not a prefetch queue operation) has logged an L2-cache error in AFSR and AFAR but not generated a trap due to a misfetch. Displacing the wrong line by mistake from the D-cache would give a data correctness problem. Displacing the wrong line by mistake from the L2-cache and L3-cache will only lead to the same error being reported twice. The second time the error is reported, the AFAR is likely to be correct. Corrupt data is never stored in the D-cache without a trap being generated to allow it to be cleared out.

Note – While the above code description appears only to be appropriate for correctable L2-cache data and tag errors, it is actually effective for uncorrectable L2-cache data errors as well. In the event that it is handling an uncorrectable error, the victimize at step 3, “Evict the L2-cache line that contained the error”, will write it out to L3-cache.

If the L2-cache still returns an uncorrectable data ECC error when the processor reads it to perform a L2-cache writeback, the WDU bit will be set in the AFSR during this trap handler, which would generate a disrupting trap later if it was not cleared somewhere in this handler.
In this case, the processor will write deliberately bad signalling ECC back to L3-cache. In the event that it is handling an uncorrectable error, the victimize at step 4, “Evict the L3-cache line that contained the error”, will either invalidate the line in error or will, if it is in M, O, or Os state, write it out to memory.

If the L3-cache still returns an uncorrectable data ECC error when the processor reads it to perform a L3-cache writeback, the L3_WDU bit will be set in the AFSR_EXT during this trap handler, which would generate a disrupting trap later if it were not cleared somewhere in this handler. In this case, the processor will write deliberately bad signalling ECC back to memory.
When the *fast_ECC_error* trap handler exits and retries the offending instruction, the previously faulty line will be re-fetched from main memory. It will either be correct, so the program will continue correctly, or still contain an uncorrectable data ECC error, in which case the processor will take a deferred *instruction_access_error* or *data_access_error* trap.

It is the responsibility of these later traps to perform the proper clean-up for the uncorrectable error. The *fast_ECC_error* trap routine does not need to execute any complex cleanup operations.

Encountering a software correctable error while executing the software correctable trap routine is unlikely to be recoverable. To avoid this, three approaches are known.

1. The software correctable exception handler code can be written normally, in cacheable space. If a single bit error exists in exception handler code in the L2-cache, other single bit L2-cache data or tag errors will be unrecoverable. To reduce the probability of this, the software correctable exception handler code can be flushed from the L2-cache and L3-cache at the end of execution. This solution does not cover cases where the L2-cache or L3-cache has a hard fault on a tag or data bit, giving a software correctable error on every access.
2. All the exception handler code can be placed on a non-cacheable page. This solution does cover hard faults on data bits in the L2-cache, at least adequately to report a diagnosis or to remove the processor from a running domain, provided that the actual exception vector for the *fast_ECC_error* trap is not in the L2-cache. Exception vectors are normally in cacheable memory. To avoid fetching the exception vector from the L2-cache, flush it from the L2-cache and L3-cache in the *fast_ECC_error* trap routine.

3. Exception handler code may be placed in cacheable memory, but only in the first 32 bytes of each 64-byte L2-cache sub-block. At the end of the 32 bytes, the code has to branch to the beginning of another L2-cache sub-block. The first 32 bytes of each L2-cache sub-block fetched from system bus are sent directly to the instruction unit without being fetched from the L2-cache. None of the I-cache or D-cache lines fetched may be in the L2-cache or L3-cache. This does cover hard faults on data bits in the L2-cache or L3-cache, for systems that do not have non-cacheable memory from which the trap routine can be run. It does not cover hard faults on tag bits. The exception vector and the trap routine must all be flushed from the L2-cache and L3-cache after the trap routine has executed.

Note – If, by some means, the processor does encounter a software correctable L2-cache ECC error while executing the *fast_ECC_error* trap handler, the processor may recurse into RED_state, and not make any record in the AFSR of the event, leading to difficult diagnosis. The processor will set the AFSR.ME bit for multiple software correctable events, but this is expected to occur routinely, when an AFAR and AFSR is captured for an instruction which is prefetched automatically by the instruction fetcher, then discarded.

The *fast_ECC_error* trap uses the alternate global registers. If a software correctable L2-cache error occurs while the processor is running some other trap which uses alternate global registers (such as spill and fill traps) there may be no practical way to recover the system state. The *fast_ECC_error* routine should note this condition and, if necessary, reset the domain rather than recover from the software correctable event. One way to look for the condition is to check whether the TL of the *fast_ECC_error* trap handler is greater than 1.

7.2.7.8 Uncorrectable L2-cache Data ECC Errors

Uncorrectable L2-cache data ECC errors occur on multi-bit data errors detected as the result of the following transactions:

- Reads of data from the L2-cache to fill the D-cache
- Reads of data from the L2-cache to fill the I-cache
- Performing an atomic instruction

These events set AFSR.UCU. An uncorrectable L2-cache data ECC error which is the result of an I-fetch, or a data read caused by any instruction other than a block load, causes a *fast_ECC_error* trap. As described in *Software Correctable L2-cache ECC Error Recovery Actions* on page 160, these errors will be recoverable by the trap handler if the line at fault was in the E or S MOESI state.

Uncorrectable L2-cache data ECC errors can also occur on multi-bit data errors detected as the result of the following transactions:

- Reads of data from an O, Os, or M state line to respond to an incoming snoop request (copyout).
- Reads of data from an E, S, O, or Os, or M state line to write it back to L3-cache (writeback).
- Reads of data from the L2-cache to merge with bytes being written by the processor (W-cache exclusive request).
- Reads of data from the L2-cache to perform an operation placed in the prefetch queue by an explicit software `PREFETCH` instruction or a P-cache hardware `PREFETCH` operation.
- Reads of the L2-cache by the processor to perform an operation block load instruction.

For copyout, the processor reading the uncorrectable error from its L2-cache sets its AFSR.CPU bit. In this case, deliberately bad signalling ECC is sent with the data to the processor issuing the snoop request. If the device issuing the snoop request is a processor, it takes an *instruction_access_error* or *data_access_error* trap. If the device issuing the snoop request is an I/O device, it will have some device-specific error reporting mechanism which the device driver must handle.

The processor being snooped logs error information in AFSR. For copyout, the processor reading the uncorrectable error from its L2-cache sets its AFSR.CPU bit.

For writeback, the processor reading the uncorrectable error from its L2-cache sets its AFSR.WDU bit, and the uncorrectable writeback data is written into L3-cache.

If an uncorrectable L2-cache Data ECC error occurs as the result of a copyout, deliberately bad signalling ECC is sent with the data to the system bus. Correct system bus ECC for the uncorrectably corrupted data is computed and transmitted on the system bus, and data bits [127:126] are inverted as the corrupt 128 bit word is transmitted on the system bus. This signals to other devices that the word is corrupt and should not be used. The error information is logged in the AFSR and an optional disrupting *ECC_error* trap is generated if the NCEEN bit is set in the Error Enable Register. Software should log the copyout error so that a subsequent uncorrectable system bus data ECC error can be correlated back to the L2-cache data ECC error.

For an uncorrectable L2-cache data ECC error as the result of a exclusive request from the store queue or W-cache, or as the result of an operation placed in the prefetch queue by an explicit software `PREFETCH` instruction, the processor sets AFSR.EDU. Data can be read from L2-cache by the W-cache exclusive request. On these W-cache exclusive request, if an uncorrectable error occurs in the requested line, a disrupting *ECC_error* trap is generated if the NCEEN bit is set in the Error Enable Register, and L2-cache sends 64-byte data to W-cache associated with the uncorrectable data error information. W-cache stores the uncorrectable data error information in W-cache, so that deliberately bad signalling ECC is scrubbed back to the L2-cache. Correct ECC is computed for the corrupt merged data, then ECC check bits 0 and 1 are inverted in the check word scrubbed to L2-cache.

The W-cache sends out data for an L2-cache writeback or copyout event if it has the latest modified data, rather than eviction from the W-cache and update of the L2-cache. For writeback or copyout, AFSR.WDU or AFSR.CPU is set, and not AFSR.EDU, which only occurs on W-cache exclusive requests and prefetch queue operations.

A copyout operation which happens to hit in the processor writeback buffer sets AFSR.WDU, not AFSR.CPU.

7.2.8 L3-cache Errors

L3-cache tags are on-chip, and data held in external RAMs, are covered by ECC. L3-cache errors can be recovered by software or hardware measures. Information on errors detected is logged in the AFSR_EXT and AFAR. L3-cache address parity is also detected and reported as a L3-cache uncorrectable or correctable data error with AFSR_EXT.L3_MECC is set.

7.2.8.1 L3-cache Tag ECC errors

The types of L3-cache tag ECC errors are:

- Hardware corrected (HW_corrected) L3-cache tag ECC errors - single bit ECC errors that are corrected by hardware.
- Uncorrectable L3-cache tag ECC errors - multi-bit ECC errors that are not correctable by hardware or software.

Each L3-cache tag entry is covered by ECC. The tag includes the MOESI state of the line, which implies that tag ECC is checked whether or not the line is valid. Tag ECC must be correct even if the line is not present in the L3-cache.

L3-cache tag ECC checking is enabled by the ET_ECC_en bit in the L3-cache control register. The processor always generates correct ECC when writing L3-cache tag entries, except when programs use diagnostic ECC tag accesses.

7.2.8.2 Hardware Corrected L3-cache Tag ECC Errors

Hardware corrected errors occur on single bit errors, in tag value or tag ECC, detected as the result of these transactions:

- Cacheable I-fetches
- Cacheable load pperations
- Atomic operations
- W-cache exclusive request to the L3-cache to obtain data and ownership of the line in W-cache
- Reads of the L3-cache by the processor in order to perform a writeback or copyout to the system bus
- Reads of the L3-cache by the processor to perform an operation placed in the prefetch queue by an explicit software `PREFETCH` instruction or a P-cache hardware `PREFETCH` operation
- Reads of the L3-cache Tag while performing snoop read, local writeback, displacement flush, L3-cache data fill

Hardware corrected errors optionally produce an *ECC_error* disrupting trap, enabled by the CEEN bit in the Error Enable Register, to carry out error logging.

Hardware corrected L3-cache tag errors set AFSR_EXT.L3_THCE and log the access physical address in AFAR. In contrast to L3-cache data ECC errors, AFSR.E_SYND is not captured.

For hardware corrected L3-cache tag errors, the processor actually writes the corrected entry back to the L3-cache tag array. Future accesses to this same tag should see no error. This is in contrast to Hardware corrected L3-cache data ECC errors, for which the processor corrects the data, but does not write it back to the L3-cache. This rewrite correction activity by the processor manages still to maintain the full required snoop latency and also obey the coherence ordering rules.

7.2.8.3 Uncorrectable L3-cache Tag ECC Errors

An uncorrectable L3-cache tag ECC error may be detected as the result of any operation which reads the L3-cache tags.

All uncorrectable L3-cache tag ECC errors are fatal errors. The processor will assert its ERROR output pin. The event will be logged in AFSR_EXT.L3_TUE or AFSR_EXT.TUE_SH and AFAR.

All uncorrectable L3-cache tag ECC errors for tag update or copyout due to foreign bus transactions or snoop request will set AFSR_EXT.L3_TUE_SH. All uncorrectable L3-cache tag ECC errors for fill, tag update due to local bus transactions, writeback will set AFSR_EXT.L3_TUE.

The response of the system to assertion of the ERROR output pin depends on the system, but is expected to result in a reboot of the affected domain. Error status can be read from the AFSR_EXT after a system reset event.

For L3_TUE or L3_TUE_SH, the request will be dropped including copyback request.

7.2.8.4 L3-cache Data ECC Errors

The types of L3-cache data ECC errors are:

- Hardware corrected (HW_corrected) L3-cache data ECC errors - single bit ECC errors that are corrected by hardware.
- Software correctable (SW_correctable) L3-cache data ECC errors - L2-cache ECC errors that require software intervention.
- Uncorrectable L3-cache data ECC errors - multi-bit ECC errors that are not correctable by hardware or software.

7.2.8.5 Hardware Corrected L3-cache Data ECC Errors

Hardware corrected ECC errors occur on single bit errors detected as the result of these transactions:

- W-cache read accesses to L3-cache (AFSR_EXT.L3_EDC)
- Reads of the L3-cache by the processor in order to perform a writeback or copyout to the system bus (AFSR_EXT.L3_WDC, AFSR_EXT.L3_CPC)
- Reads of the L3-cache by the processor to perform an operation placed in the prefetch queue by an explicit software `PREFETCH` instruction (AFSR_EXT.L3_EDC)
- Reads of L3-cache by the processor to perform an operation block load instruction (AFSR_EXT.L3_EDC)

Hardware corrected errors optionally produce an *ECC_error* disrupting trap, enabled by the CEEN bit in the Error Enable Register, to carry out error logging.

Note – For HW_corrected L3-cache data errors, the hardware does not actually write the corrected data back to the L3-cache data SRAM. However, the L3-cache sends corrected data to the P-cache, the W-cache, and the system bus. Therefore, the instruction that creates the single bit error transaction can be completed without correcting the L3-cache data SRAM. The L3-cache data SRAM may later be corrected in the disrupting *ECC_error* trap.

7.2.8.6 Software Correctable L3-cache Data ECC Errors

Software correctable errors occur on single-bit data errors detected as the result of the following transactions:

- Reads of data from the L3-cache to fill the D-cache.

- Reads of data from the L3-cache to fill the I-cache.
- Performing an atomic instruction.

All these events cause the processor to set AFSR_EXT.L3_UCC. A software correctable error will generate a precise *fast_ECC_error* trap if the UCEEN bit is set in the Error Enable Register. See *Software Correctable L3-cache ECC Error Recovery Actions* on page 166.

7.2.8.7 Software Correctable L3-cache ECC Error Recovery Actions

The *fast_ECC_error* trap handler should carry out the following sequence of actions to correct an L3-cache tag or data ECC error:

1. Read the address of the correctable error from the AFAR register.
2. Invalidate the entire D-cache using writes to `ASI_DCACHE_TAG` to zero the valid bits in the tags.
3. Evict the L2-cache line that contained the error. This requires four `LDXA ASI_L2CACHE_TAG` operations using the disp_flush addressing mode to evict all four ways of the L2-cache.

L2-cache and L3-cache are mutually exclusive. When a read miss from primary caches hit in L3-cache, the line will be moved from L3-cache to L2-cache. If the line contains the error, it is still moved to L2-cache without any change. Therefore, the recovery actions for software correctable L3-cache ECC error must displacement flush the line in L2-cache.

A single bit data error will be corrected when the processor reads the data from the L2-cache and writes it to the L3-cache to perform the writeback. This operation may set the AFSR.WDC or AFSR.WDU bits. If the offending line was in O, Os, or M MOESI state, and another processor happened to read the line while the trap handler was executing, the AFSR.CPC or AFSR.CPU bits could be set.

A single bit tag error will be corrected when the processor reads the line tag to update it to I state. This operation may set AFSR.THCE.

4. Evict the L3-cache line that contained the error. This requires four `LDXA ASI_L3CACHE_TAG` operations using the L3-cache disp_flush addressing mode to evict all four ways of the L3-cache to memory.

When the line is displacement flushed from L2-cache, if the line is not in I state, then it will be writeback to L3-cache.

5. Log the error.
6. Clear AFSR.WDC, WDU, CPC, CPU, THCE, AFSR_EXT.L3_UCC, L3_UCU, L3_WDC, L3_WDU, L3_CPC, L3_CPU, and L3_THCE.
7. Clear AFSR2 and AFSR2_EXT
8. Displacement flush any cacheable *fast_ECC_error* exception vector or cacheable *fast_ECC_error* trap handler code or data from the L2-cache to L3-cache.
9. Displacement flush any cacheable *fast_ECC_error* exception vector or cacheable *fast_ECC_error* trap handler code or data from the L3-cache.
10. Re-execute the instruction that caused the error using `RETRY`.

Data in error *is* stored in the D-cache. If the data was read from the L3-cache as the result of a load instruction, or an atomic instruction, corrupt data will be stored in D-cache. However, if the data was read as the result of a block load instruction, corrupt data will *not* be stored in D-cache. Store instructions never cause *fast_ECC_error* traps directly, just load and atomic instructions. Store-like instructions never result in corrupt data being loaded into the D-cache.

The entire D-cache is invalidated because there are circumstances when the AFAR used by the trap routine does not point to the line in error in the D-cache. This can happen when multiple errors are reported, and when an instruction fetch (not a prefetch queue operation) has logged an L3-cache error in AFSR_EXT and AFAR but not generated a trap due to a misfetch. Displacing the wrong line by mistake from the D-cache would give a data correctness problem. Displacing the wrong line by mistake from the L2-cache and L3-cache will only lead to the same error being reported twice. The second time the error is reported, the AFAR is likely to be correct. Corrupt data is never stored in the D-cache without a trap being generated to allow it to be cleared out.

Note – While the above code description appears only to be appropriate for correctable L3-cache data and tag errors, it is actually effective for uncorrectable L3-cache data errors as well. In the event that it is handling an uncorrectable error, the victimize at step 3, “Evict the L2-cache line that contained the error”, will write it out to L3-cache.

If the L2-cache still returns an uncorrectable data ECC error when the processor reads it to perform a L2-cache writeback, the WDU bit will be set in the AFSR during this trap handler, which would generate a disrupting trap later if it was not cleared somewhere in this handler.
In this case, the processor will write deliberately bad signalling ECC back to L3-cache. In the event that it is handling an uncorrectable error, the victimize at step 4, “Evict the L3-cache line that contained the error”, will either invalidate the line in error or will, if it is in M, O, or Os state, write it out to memory.

If the L3-cache still returns an uncorrectable data ECC error when the processor reads it to perform a L3-cache writeback, the L3_WDU bit will be set in the AFSR_EXT during this trap handler, which would generate a disrupting trap later if it were not cleared somewhere in this handler.
In this case, the processor will write deliberately bad signalling ECC back to memory. When the *fast_ECC_error* trap handler exits and retries the offending instruction, the previously faulty line will be re-fetched from main memory. It will either be correct, so the program will continue correctly, or still contain an uncorrectable data ECC error, in which case the processor will take a deferred *instruction_access_error* or *data_access_error* trap.

It is the responsibility of these later traps to perform the proper cleanup for the uncorrectable error. The *fast_ECC_error* trap routine does not need to execute any complex cleanup operations.

Encountering a software correctable error while executing the software correctable trap routine is unlikely to be recoverable. To avoid this, three approaches are known.

1. The software correctable exception handler code can be written normally, in cacheable space. If a single bit error exists in exception handler code in the L3-cache, other single bit L3-cache data or tag errors will be unrecoverable. To reduce the probability of this, the software correctable exception handler code can be flushed from the L3-cache at the end of execution. This solution does not cover cases where the L3-cache has a hard fault on a tag or data bit, giving a software correctable error on every access.
2. All the exception handler code can be placed on a non-cacheable page. This solution does cover hard faults on data bits in the L3-cache, at least adequately to report a diagnosis or to remove the processor from a running domain, provided that the actual exception vector for the

fast_ECC_error trap is not in the L3-cache. Exception vectors are normally in cacheable memory. To avoid fetching the exception vector from the L3-cache, flush it from the L3-cache in the *fast_ECC_error* trap routine.

3. Exception handler code may be placed in cacheable memory, but only in the first 32 bytes of each 64-byte L2-cache sub-block. At the end of the 32 bytes, the code has to branch to the beginning of another L2-cache sub-block. The first 32 bytes of each L2-cache sub-block fetched from system bus are sent directly to the instruction unit without being fetched from the L2-cache or L3-cache. None of the I-cache or D-cache lines fetched may be in the L2-cache or L3-cache. This does cover hard faults on data bits in the L2-cache or L3-cache, for systems that do not have non-cacheable memory from which the trap routine can be run. It does not cover hard faults on tag bits. The exception vector and the trap routine must all be flushed from the L2-cache and L3-cache after the trap routine has executed.

Note – If, by some means, the processor does encounter a software correctable L3-cache ECC error while executing the *fast_ECC_error* trap handler, the processor may recurse into RED_state, and not make any record in the AFSR of the event, leading to difficult diagnosis. The processor will set the AFSR.ME bit for multiple software correctable events, but this is expected to occur routinely, when an AFAR and AFSR is captured for an instruction which is prefetched automatically by the instruction fetcher, then discarded.

The *fast_ECC_error* trap uses the alternate global registers. If a software correctable L3-cache error occurs while the processor is running some other trap which uses alternate global registers (such as spill and fill traps) there may be no practical way to recover the system state. The *fast_ECC_error* routine should note this condition and, if necessary, reset the domain rather than recover from the software correctable event. One way to look for the condition is to check whether the TL of the *fast_ECC_error* trap handler is greater than 1.

7.2.8.8 Uncorrectable L3-cache Data ECC Errors

Uncorrectable L3-cache data ECC errors occur on multi-bit data errors detected as the result of the following transactions:

- Reads of data from the L3-cache to fill the D-cache.
- Reads of data from the L3-cache to fill the I-cache.
- Performing an atomic instruction.

These events set AFSR_EXT.L3_UCU. An uncorrectable L3-cache data ECC error which is the result of an I-fetch, or a data read caused by any instruction other than a block load, causes a *fast_ECC_error* trap. As described in *Software Correctable L3-cache Data ECC Errors* on page 165, these errors will be recoverable by the trap handler if the line at fault was in the E or S MOESI state.

Uncorrectable L3-cache data ECC errors can also occur on multi-bit data errors detected as the result of the following transactions:

- Reads of data from an O, Os, or M state line to respond to an incoming snoop request (copyout).
- Reads of data from an O, Os, or M state line to write it back to memory (writeback).
- Reads of data from the L3-cache to merge with bytes being written by the processor (W-cache exclusive request).

- Reads of data from the L3-cache to perform an operation placed in the prefetch queue by an explicit software PREFETCH instruction or a P-cache hardware PREFETCH operation.
- Reads of the L2-cache by the processor to perform an operation block load instruction.

For copyout, the processor reading the uncorrectable error from its L3-cache sets its AFSR_EXT.L3_CPU bit. In this case, deliberately bad signalling ECC is sent with the data to the processor issuing the snoop request. If the processor issuing the snoop request is a processor, it takes an *instruction_access_error* or *data_access_error* trap. If the procssor issuing the snoop request is an I/O device, it will have some device-specific error reporting mechanism which the device driver must handle.

The processor being snooped logs error information in AFSR_EXT. For copyout, the processor reading the uncorrectable error from its L3-cache sets its AFSR_EXT.L3_CPU bit.

For writeback, the processor reading the uncorrectable error from its L3-cache sets its AFSR_EXT.L3_WDU bit.

If an uncorrectable L3-cache Data ECC error occurs as the result of a writeback or a copyout, deliberately bad signalling ECC is sent with the data to the system bus. Correct system bus ECC for the uncorrectably corrupted data is computed and transmitted on the system bus, and data bits [127:126] are inverted as the corrupt 128-bit word is transmitted on the system bus. This signals to other devices that the word is corrupt and should not be used. The error information is logged in the AFSR_EXT and an optional disrupting *ECC_error* trap is generated if the NCEEN bit is set in the Error Enable Register. Software should log the writeback error or the copyout error so that a subsequent uncorrectable system bus data ECC error, reported by this processor or any other processor, can be correlated back to the L3-cache data ECC error.

For an uncorrectable L3-cache data ECC error as the result of a exclusive request from the store queue or W-cache, or as the result of an operation placed in the prefetch queue by an explicit software PREFETCH instruction, the processor sets AFSR_EXT.L3_EDU. If the W-cache is turned off for some reason, the store buffer causes this to happen on every store-like and atomic instruction for cacheable data. On these W-cache exclusive request, if an uncorrectable error occurs in the requested line, a disrupting *ECC_error* trap is generated if the NCEEN bit is set in the Error Enable Register, and L3-cache sends 64-byte data to W-cache associated with the uncorrectable data error information. W-cache stores the uncorrectable data error information in W-cache, so that deliberately bad signalling ECC is scrubbed back to the L2-cache during W-cache eviction. Correct ECC is computed for the corrupt evicted W-cache data, then ECC check bits 0 and 1 are inverted in the check word scrubbed to L2-cache.

A copyout operation which happens to hit in the processor writeback buffer sets AFSR_EXT.L3_WDU, not AFSR_EXT.L3_CPU.

L3-cache address parity errors

When an L3-cache address parity error is detected, it is reported and treated as an uncorrectable L3-cache data error, which are described in previous subsections, with AFSR_EXT.L3_MECC set. When an L3-cache address parity error occurs, AFSR_EXT.L3_MECC is set and based on the request source, the corresponding L3_UCU, or L3_EDU, or L3_CPU, or L3_WDU is also set. The recovery actions for L3_UCU, L3_EDU, L3_CPU, L3_WDU are also applied to L3-cache address parity errors.

In very rare cases, when an L3-cache address parity error is detected, AFSR_EXT.L3_MECC is set and based on the request source, the corresponding L3_UCC, L3_EDC, L3_CPC, or L3_WDC is also set. However, the recovery action should treat this as an uncorrectable L3-cache data error.

Note – There is no real parity check (parity pin) for the L3-cache data address bus since the parity check requires a motherboard respin. To avoid motherboard respin and to protect SRAM data address bus, 9-bit ECC for the first DIMM is stored in the second DIMM, and vice versa. If an error occurs on the address bus, there is high possibility that ECC violation is detected on both DIMMs. Thus, we assume that address bus errors occur when both DIMMs get ECC errors.

7.2.9 Errors on the System Bus

Errors on the system bus are detected as the result of:

- Cacheable read accesses to the system bus
- Non-cacheable read accesses to the system bus

Data ECC, microtag ECC, system bus error with Dstat = 2 or 3, and unmapped responses are always checked on these accesses.

- Fetching interrupt vector data by the processor on the system bus

Data ECC, microtag ECC and system bus error with Dstat = 2 or 3 responses are always checked on interrupt vector fetches.

- Cacheable write accesses to the system bus
- Non-cacheable write accesses to the system bus
- Transmitting interrupts by the processor on the system bus

Unmapped response is checked on these accesses above.

7.2.9.1 HW_corrected System Bus Data and Microtag ECC Errors

ECC is checked for data and microtags arriving at the processor from the system bus. Single bit errors in data and microtags are fixed in hardware. A single bit data error as the result of a system bus read from memory or I/O sets the AFSR.CE bit. A single bit data error as the result of an interrupt vector fetch sets AFSR.IVC. A single bit microtag error as the result of a system bus read from memory or I/O or sets AFSR.EMC. A single bit microtag error as the result of an interrupt vector fetch sets AFSR.IMC.

HW_corrected system bus errors cause an *ECC_error* disrupting trap if the CEEN bit is set in the Error Enable Register. The HW_corrected error is corrected by the system bus interface hardware at the processor, and the processor uses the corrected data automatically.

The microtag ECC correctness is checked whether or not the processor is configured in SSM mode by setting the SSM bit in the Fireplane configuration register.

All four microtag values associated with a 64-byte system bus read are checked for ECC correctness.

7.2.9.2 Uncorrectable System Bus Data ECC Errors

An uncorrectable system bus data ECC error, as the result of a system bus read from memory or I/O, caused by an instruction fetch, load-like, block load or atomic operation, sets the AFSR.UE bit. The ECC syndrome is captured in E_SYND. The AFSR.PRIV bit is set if PSTATE.PRIV was set at the time the error was detected.

If the same ECC error is caused by a read triggered by a prefetch queue operation, or caused by the read-to-own required to obtain permission to complete a store-like operation for which data is held in the store queue, then the behavior is different. An uncorrectable system bus data ECC error will set AFSR.DUE and will capture the ECC syndrome in E_SYND. AFSR.PRIV will not be captured.

Uncorrectable system bus data ECC errors on read accesses to a cacheable space will install the bad ECC from the system bus directly in the L2-cache. This prevents using the bad data, or having the bad data written back to memory with good ECC bits. Uncorrectable ECC errors from the system bus on cache fills will be reported for any ECC error in the 64 byte line, not just the referenced word. The error information is logged in the AFSR. An *instruction_access_error* or *data_access_error* deferred trap (if AFSR.UE) or an *ECC_error* disrupting trap (if AFSR.DUE) is generated provided that the NCEEN bit is set in the Error Enable Register. If NCEEN were clear, the processor would operate incorrectly on the corrupt data.

An uncorrectable error as the result of a system bus read for an instruction fetch causes an *instruction_access_error* deferred trap. An uncorrectable error as the result of a load-like, block load or atomic operation, causes a *data_access_error* deferred trap. An uncorrectable error as the result of a prefetch queue or store queue system bus read causes a disrupting *ECC_error* trap. See *Multiple Errors and Nested Traps* on page 199, for the behavior in the event of multiple errors being detected simultaneously.

When an uncorrectable error is present in a 64-byte line read from the system bus in order to complete a load-like or atomic instruction, corrupt data will be installed in the D-cache. The deferred trap handler should invalidate the D-cache during recovery.

Corrupt data is never stored in the I-cache or P-cache.

An uncorrectable system bus data ECC error on a read to a non-cacheable space is handled in the same way as cacheable accesses, except that the error cannot be stored in the processor caches so there is no need to flush them. An uncorrectable error cannot occur as the result of a store-like operation to uncached space.

An uncorrectable system bus data ECC error as the result of an interrupt vector fetch sets AFSR.IVU in the processor fetching the vector. The error is not reported to the processor which generates the interrupt. When the uncorrectable interrupt vector data is read by the interrupt vector fetch hardware of the processor receiving the interrupt, a disrupting *ECC_error* exception is generated. No *interrupt_vector* trap is generated. The processor will store the uncorrected interrupt vector data in the internal interrupt registers unmodified, as it is received from the system bus.

7.2.9.3 Uncorrectable System Bus Microtag Errors

An uncorrectable microtag ECC error as the result of a system bus read of memory or I/O sets AFSR.EMU.

Whether or not the processor is configured in SSM mode by setting the SSM bit in the Sun Fireplane Interconnect Configuration register, system bus microtag ECC is checked.

All four microtag values associated with a 64-byte system bus read are checked for ECC correctness.

Uncorrectable errors in microtags arriving at the processor from the system bus are not normally recoverable. When the processor detects one of these errors, it will assert its ERROR output pin. The response of the system to the assertion of the ERROR output pin is system-dependent, but will usually result in the reset of all the chips in the affected coherence domain.

In addition to asserting its ERROR output pin, the processor will take a trap, if the NCEEN bit is set in the Error Enable Register. This will be an *instruction_access_error* disrupting trap if the error is the result of an instruction fetch, or a *data_access_error* disrupting trap for any other operation. Whether the trap taken has any effect or meaning will depend on the system's response to the processor ERROR output pin.

The effect of an uncorrectable microtag ECC error on the L2-cache state is undefined.

System bus microtag ECC is checked on interrupt vector fetch operations and on read accesses to uncacheable space, even though the microtag has little meaning for these. An uncorrectable error in microtag will still result in the ERROR output pin being asserted, AFSR.IMU being set, M_SYND and AFAR being captured, and disrupting trap being taken.

7.2.9.4 System Bus "DSTAT = 2 or 3" Errors

A DSTAT = 2 or 3 may be returned in response to a system bus read operation. In this case, the processor handles the event in the same way as specified in the section titled *Uncorrectable System Bus Data ECC Errors* on page 171, except for the following differences:

1. For a system bus read from memory or I/O, caused by an instruction fetch, load, block load or atomic operation, the AFSR.BERR bit is set (instead of AFSR.UE).
2. For a system bus read from memory or I/O caused by prefetch queue, or a system bus read from memory causes by read-to-own store queue operation, the AFSR.DBERR bit is set (instead of AFSR.DUE).
3. The BERR or DBERR AFSR and AFAR overwrite priorities are used rather than the UE or DUE priorities.
4. Data bits [1:0] of each of the four 128 bit correction words written to the L2-cache are inverted to create signalling ECC, if the access is cacheable.
5. For AFSR.BERR, a deferred *instruction_access_error* or *data_access_error* trap is generated.
6. For AFSR.DBERR, a disrupting *ECC_error* trap is generated.

The processor treats both Sun Fireplane Interconnect termination code DSTAT = 2 ("time-out error"), and DSTAT = 3 ("bus error"), as the same event. Both will cause AFSR.BERR or AFSR.DBERR to be set and cause the same signalling ECC to be sent to the L2-cache. These conditions are checked on both cacheable and non cacheable reads. Neither sets AFSR.TO nor AFSR.DTO.

7.2.9.5 System Bus Unmapped Errors

The AFSR.TO or AFSR.DTO bit is set when no device responds with a MAPPED status as the result of the system bus address phase. This is not a hardware time-out operation, which causes an AFSR.PERR event. It is also different from a DSTAT = 2 "time-out" response for a Sun Fireplane Interconnect transaction, which actually sets AFSR.BERR or AFSR.DBERR.

A TO or DTO may be returned in response to a system bus read or write operation. In this case, the processor handles the event in the same way as specified above at *Uncorrectable System Bus Data ECC Errors* on page 171, except for the following differences:

1. The AFSR.DTO is set (instead of AFSR.DUE) for a system bus read caused by prefetch queue or read-to-own store queue operation.
2. The AFSR.TO is set (instead AFSR.UE) for a system bus read caused by an instruction fetch, load-like, block load, or atomic operation.
3. The AFSR.TO is also set for all system bus unmapped errors caused by write access transfer. This includes block store to memory (WS), store to I/O (WIO), block store to I/O (WBIO), or writeback from L3-cache operation (WB).
4. The AFSR.TO is also set for all system bus unmapped errors caused by issuing an interrupt to undefined target device or disabled logical processor (INT).
5. The TO or DTO AFSR and AFAR overwrite priorities are used rather than the UE or DUE priorities.
6. If the access is a cacheable read transfer, the data value from the system bus, and the ECC present on the system bus, are not written to the L2-cache.
7. For AFSR.TO, a deferred *instruction_access_error* or *data_access_error* trap is generated. The latter applies even if the event was a writeback from L2-cache, not directly related to instruction processing.
8. For AFSR.DTO, a disrupting *ECC_error* trap is generated.

It is possible that no destination asserts MAPPED when the processor attempts to send an interrupt. This event also causes AFSR.TO to be set, and PRIV and AFAR to be captured, although the exact meaning of the captured information is not clear. A deferred *data_access_error* trap will be taken.

Copyout events from L2-cache or L3-cache cannot see an AFSR.TO event, because they are responses to Sun Fireplane Interconnect transactions, not initiating Sun Fireplane Interconnect transactions.

7.2.9.6 System Bus Hardware Time-outs

The AFSR.TO bit is set when no device responds with a MAPPED status as the result of the system bus address phase. This is not a hardware time-out operation.

In addition to the AFSR.TO functionality, there are hardware time-outs that detect that an ASIC is taking too long to complete a system bus operation. This time-out might come into effect if, for example, a target device developed a fault during an access to it.

Hardware time-outs are reported as AFSR.PERR fatal error events, no matter what bus activity was taking place. No other status is logged in the AFSR. The AFAR will log the address causing time-out.

- When a transaction stays at the top of CPQ more than the time period specified in the TOL field of Sun Fireplane Interconnect Configuration Register, the CPQ_TO bit will be set in the EESR, and the PA[42:4] will be logged into AFAR.
- When a transaction stays at the top of NCPQ more than the time period specified in the TOL field of Sun Fireplane Interconnect Configuration Register, the NCPQ_TO bit will be set in the EESR, and the PA[42:4] will be logged into AFAR.

7.2.10 Memory Errors and Prefetch

7.2.10.1 Memory Errors and Prefetch by the Instruction Fetcher and I-cache

The instruction fetcher sends requests for instructions to the I-cache before it is certain that the instructions will ever be executed. This occurs, for example, when a branch is mispredicted. These appear as perfectly normal operations to the rest of the processor and the rest of the system, which cannot tell that they are prefetch requests.

One of these requests from the instruction fetcher to the I-cache can miss in the I-cache and cause a fetch from the L2-cache. It can also miss in the L2-cache. A miss in the L3-cache can cause a fetch from the system bus. In addition, any instruction fetch by an I-cache miss causes an automatic read by the I-cache of the next I-cache line from the L2-cache/L3-cache.

In the event of an error from the L2-cache or L3-cache for one of these fetches, a *fast_ECC_error* trap is generated, provided that the fetched instruction is actually executed. If the instruction marked as encountering an error is discarded without being executed, no trap is generated. However, AFSR, AFSR_EXT, and AFAR will still be updated with the L2-cache or L3-cache error status.

In the event of an error from the system bus for an instruction fetch, the processor works exactly as normal, with the AFSR and AFAR being set, and a deferred *instruction_access_error* trap being taken, despite the fact that the faulty line has not yet been used in the committed instruction stream, and may in fact never be used.

The above applies to speculative fetches well beyond a branch and also to annulled instructions in the delay slot of a branch.

Corrupt data is never stored in the I-cache.

The execution unit can issue speculative requests for data because of load-like instructions (but not block load, store, or atomic operations). These can miss in the D-cache and go to the L2-cache. However, in all circumstances, if the data is not to be used, the execution unit cancels the fetch before the L2-cache can detect any errors. The AFSR and AFAR are not updated, the D-cache is not loaded with corrupt data, and no trap is taken.

Speculative data fetches which are later discarded never cause system bus reads. Speculation around store-like instructions never cause system bus reads for stores that will not be executed.

7.2.10.2 Memory Errors and Hardware PREFETCH

The P-cache can autonomously read data from the L2-cache into the internal P-cache. This is called “hardware prefetch”. This never generates system bus activity.

In the UltraSPARC IV+ processor, errors detected as the result of hardware prefetch operations are treated exactly the same as errors detected as the result of explicit software PREFETCH.

7.2.10.3 Memory Errors and Software PREFETCH

Programs can execute explicit software PREFETCH operations. The effect of an explicit software PREFETCH instruction is to write a command to the prefetch queue, an autonomous hardware queue outside of the execution unit. The prefetch queue hardware works very much like the store queue, L3-cache accesses and system bus accesses being handled by the hardware, completely decoupled from the flow of instructions through the execution unit.

In the UltraSPARC IV+ processor, errors as the result of operations from the prefetch queue are handled the same as errors as the result of operations from the store queue.

A prefetch queue operation first searches the L2-cache and L3-cache. If it misses in both the L2-cache and L3-cache, it will generate a system bus read operation. After the prefetch queue operation completes, the prefetched data will be in the P-cache and maybe in the L2-cache depending on the type of prefetch it is.

If the prefetch queue operation to the L2-cache detects a single bit L2-cache data error, AFSR.EDC will be set, AFAR captured, and a disrupting *ECC_error* trap generated to log the event. The P-cache line being fetched will not be installed. The L2-cache data is unchanged.

If the prefetch queue operation to the L2-cache detects an uncorrectable L2-cache data error, AFSR.EDU will be set, AFAR captured, and a disrupting *ECC_error* trap generated to log the event. The P-cache line being fetched will not be installed.

If the prefetch queue operation that misses in L2-cache and hits in L3-cache detects a single bit L3-cache data error, AFSR_EXT.L3_EDC will be set, AFAR captured, and a disrupting *ECC_error* trap generated to log the event. The P-cache line being fetched will not be installed. The L3-cache data will be moved from L3-cache to L2-cache without any correction.

If the prefetch queue operation that misses in L2-cache and hits in L3-cache detects an uncorrectable L2-cache data error, AFSR.EDU will be set, AFAR captured, and a disrupting *ECC_error* trap generated to log the event. The P-cache line being fetched will not be installed. The L3-cache data will be moved from L3-cache to L2-cache without any change.

If the prefetch queue operation causes a system bus read operation, correctable data ECC, uncorrectable data ECC, correctable microtag ECC, uncorrectable microtag ECC, unmapped, or DSTAT = 2 or 3 could be returned. All of these are handled in the same way as a system bus read-to-own operation triggered by a store queue entry.

If a single bit data ECC error or single bit microtag ECC error is returned from the system bus for a prefetch queue operation, the event is logged in AFSR.CE, or AFSR.EMC and AFAR. Hardware will correct the error and install the prefetched data in the P-cache or L2-cache or both.

Uncorrectable system bus data ECC as the result of a prefetch queue operation will set AFSR.DUE and generate an *ECC_error* disrupting trap. If the prefetch queue operation causes an RTO, or an RTSR in an SSM system, the unmodified uncorrectable error will be installed in the L2-cache. Otherwise, the L2-cache line remains invalid.

Corrupt data is never stored in P-cache.

Uncorrectable system bus microtag ECC as the result of an operation from the prefetch queue will set AFSR.EMU and cause the processor to assert its ERROR output pin and take a *data_access_error* trap. If the prefetch instruction causes an RTO, or an RTSR in an SSM system, the unmodified uncorrectable error will be installed in the L2-cache. Otherwise, the L2-cache line remains invalid.

If a bus error or unmapped error is returned from the system bus for a prefetch queue operation, the processor sets AFSR.DBERR or DTO and takes a disrupting *data_access_error*.

The behavior on errors for prefetch queue operations does not depend on the privilege state of the original PREFETCH instruction.

In the UltraSPARC IV+ processor, any error returned as the result of a prefetch queue operation is correctly reported to the operating system for logging.

7.2.11 Memory Errors and Interrupt Transmission

System bus data ECC errors for an interrupt vector fetch operation are treated in a special manner. HW_corrected interrupt vector data ECC errors set AFSR.IVC (not AFSR.CE) and correct the error in hardware before writing the vector data into the interrupt receive registers and generating an *interrupt_vector* trap. Uncorrectable interrupt vector data ECC errors set AFSR.IVU (not AFSR.UE or DUE), write the received vector data unchanged into the interrupt receive registers, and do not generate an *interrupt_vector* disrupting trap. AFSR.E_SYND will be captured. AFAR will not be captured. AFSR.PRIV will be updated with the state that happens to be in PSTATE.PRIV at the time the event is detected.

System bus microtag ECC errors for interrupt vector fetches, whether in an SSM system or not, are treated exactly as though the bus cycle was a read access to I/O or memory. AFSR.IMC or IMU is set, M_SYND and AFAR will be captured. The value captured in AFAR is not meaningful. AFSR.PRIV will be updated with the state that happens to be in PSTATE.PRIV at the time the event is detected. For AFSR.IMU events, the processor will assert its ERROR output pin and take a disrupting *instruction_access_error* trap or *data_access_error* trap. For AFSR.IMC events, an *ECC_error* disrupting trap will be taken. Both of these events also generate an *interrupt_vector* trap.

A Sun Fireplane Interconnect DSTAT = 2 or DSTAT = 3 response from the interrupting device to an interrupt vector fetch operation will set AFSR.BERR at the processor which is fetching the interrupt vector. The interrupt vector data received in this transfer is written into the interrupt receive registers, and an *interrupt_vector* exception is generated, even though the data may be incorrect. A deferred *data_access_error* trap is also generated.

A processor transmitting an interrupt may receive no MAPPED response to its Sun Fireplane Interconnect address cycle. This is treated exactly as though the bus cycle was a read access to I/O or memory. AFSR.TO will be set, AFAR will be captured (although its meaning is uncertain) and AFSR.PRIV will be updated with the state that happens to be in PSTATE.PRIV at the time the event is detected. A deferred *data_access_error* trap will be generated.

7.2.12 Cache Flushing in the Event of Multiple Errors

If a software trap handler needs to flush a line from any processor cache in order to ensure correct operation as part of recovery from an error, and multiple uncorrectable errors are reported in the AFSR, either through multiple sticky bits or through AFSR.ME, then the value stored in AFAR may not be the only line needing to be flushed. In this case, the trap handler should flush all D-cache contents from the processor to be sure of flushing all the required lines.

7.3 Error Registers

7.3.1 Shared Error Reporting

Shared errors are more complicated than private errors. When a shared error occurs, it must be recorded; one of the logical processor within the CMT must be trapped to deal with the error. By definition, shared errors are asynchronous errors (if they could be identified with an instruction they could be identified with a logical processor) that occur in shared resources. Even within this set, as many errors as possible are attributed to a specific core and reported as a private errors. Where to record the error and which logical processor to trap is addressed in the following subsections.

Note – `MEMBAR #Sync` is generally needed after stores to error ASI Registers.

7.3.1.1 CMT Error Steering Register (`ASI_CMP_ERROR_STEERING`)

The CMT Error Steering register, is a shared register accessible from all logical processors as well as being accessible from the system controller. This register is used to control and identify which logical processor is responsible for handling all shared errors. The specified logical processor has the error recorded in its asynchronous error reporting mechanism and takes an appropriate trap, to resolve the error.

Whenever an error occurs that cannot be identified with any particular logical processor, that error is recorded in, and a trap is sent to the logical processor, identified by the CMT Error Steering register.

Name: `ASI_CMP_ERROR_STEERING` Register
ASI 0x41, VA[63:0]==0x40,
Read-Write, Privileged Access

TABLE 7-5 CMT Error Steering Register

Bit	Field	Description
[63:1]	*Reserved*	Reserved for future implementation.
[0]	Target_ID	The Target_ID field has only one 1-bit field that encodes the LP_ID of the logical processor that will be informed of shared errors. The Target_ID indicates the TTE that has an LP_ID equal in value to that of the Target_ID.

Software is responsible for ensuring that the CMT Error Steering register identifies an appropriate logical processor. Particularly the case of assigning the LP_ID of non-enabled logical processor to the CMT Error Steering register must be avoided. If the CMT Error Steering register identifies a logical processor that is parked, the shared error is reported to that logical processor and the logical processor will take the appropriate trap but not until after it is unparked.

The timing of the update to the CMT Error Steering register is not defined. If the store to the CMT Error Steering register is followed by a MEMBAR synchronization barrier, the completion of the barrier guarantees the completion of the update. During the update of the CMT Error Steering

register, the hardware guarantees that the error reporting and trap are both delivered to the same logical processor, either the logical processor specified by the old or new value of the Steering register.

7.3.1.2 Recording Shared Errors

Before a trap can be generated for a shared error, the error must be recorded. Shared errors are recorded in the asynchronous error reporting mechanism of the logical processor specified by the `ASI_CMP_ERROR_STEERING` register. The asynchronous error reporting mechanism that is used for reporting private errors is also used in this case.

7.3.2 Error Enable Register

Refer to TABLE 7-6 for the state of this register after reset.

ASI== $4B_{16}$, VA[63:0]==0x0, Shared

Name: `ASI_ESTATE_ERROR_EN_REG`

TABLE 7-6 Error Enable Register After Reset

Bits	Field	Description	RW
[22]	FPPE	Force Cport data parity error on data parity bit on both incoming and outgoing data	RW
[21]	FDPE	Force Cport data parity error on data LSB bit on both incoming and outgoing data	RW
[20]	FISAPE	Force Sun Fireplane Interconnect address parity error on parity bit	RW
[19]	FSDAPE	Force SDRAM address parity error on parity bit	RW
[18]	FMT	Force error on the outgoing system microtag ECC	RW
[17:14]	FMECC	Forced error on the outgoing system microtag ECC vector	RW
[13]	FMD	Force error on the outgoing system Data ECC	RW
[12:4]	FDECC	Forced error on the outgoing system Data ECC vector	RW
[3]	UCEEN	Enable fast_ECC_error trap on SW_correctable and uncorrectable L3-cache error	RW
[2]	Reserved	Reserved for future implementation.	RW
[1]	NCEEN	Enable instruction_access_error, data_access_error or ECC_error trap on uncorrectable ECC errors and system errors	RW
[0]	CEEN	Enable ECC_error trap on HW_corrected ECC errors	RW

FPPE: When this bit is 1, force Cport (processor data port) data parity error on data parity bit.

FDPE: When this bit is 1, force Cport data parity error on data LSB bit.

FISAPE: When this bit is 1, force Sun Fireplane Interconnect address parity error on parity bit.

FSDAPE: When this bit is 1, force SDRAM address parity error on parity bit during memory write access.

FMT: When this bit is 1, the contents of the FMECC field are transmitted as the system bus microtag ECC bits, for all data sent to the system bus by this processor. This includes writeback, copyout, interrupt vector and non-cacheable store-like operation data.

FMECC: 4 bit ECC vector to transmit as the system bus microtag ECC bits.

FMD: When this bit is 1, the contents of the FDECC field are transmitted as the system bus data ECC bits, for all data sent to the system bus by this processor. This includes writeback, copyout, interrupt vector and non-cacheable store-like operation data.

FDECC: 9 bit ECC vector to transmit as the system bus data ECC bits.

The FMT and FMD fields allow test code to confirm correct operation of system bus error detection hardware and software. To check L3-cache error detection, test programs should use the L3-cache diagnostic access operations.

UCEEN: If set a SW_correctable or uncorrectable L2-cache ECC error or uncorrectable L3-cache ECC error will generate a precise *fast_ECC_error* trap. This event can only occur on reads of the L2-cache or L3-cache by this processor for I-fetches, data loads and atomic operations, and not on merge, writeback and copyout operations. This bit enables the traps associated with the AFSR.UCC and UCU, and AFSR_EXT. L3_UCC and L3_UCU.

NCEEN: If set:

- An uncorrectable system bus data or microtag ECC error, system bus error with unmapped, or DSTAT = 2 or 3 response as the result of an instruction fetch causes a deferred *instruction_access_error* trap*,* and as the result of a load-like, atomic or block load instruction causes a deferred *data_access_error* trap.
- An uncorrectable L2-cache/L3-cache data error as the result of a store queue exclusive request, a prefetch queue operation, or for a writeback or copyout, will generate a disrupting *ECC_error* trap.
- An uncorrectable L2-cache/L3-cache tag error will cause a disrupting *ECC_error* trap.
- An uncorrectable system bus data ECC error as the result of an interrupt vector fetch will generate a disrupting *ECC_error* trap.
- An uncorrectable system bus microtag ECC error or system bus DSTAT = 2 or 3 will cause a deferred *data_access_error* trap.
- If NCEEN is clear, the error is logged in the AFSR and no trap is generated. This bit enables the traps associated with the AFSR.EMU, EDU, WDU, CPU, IVU, UE, DUE, BERR, DBERR, TO, DTO, and IMU bits and AFSR_EXT.L3_WDU, L3_CPU, L3_EDU bits.

Note – Executing code with NCEEN clear can lead to the processor executing instructions with uncorrectable errors spuriously, because it will not take traps on uncorrectable errors.

CEEN: If set:

- A HW_corrected data or microtag ECC error detected as the result of a system bus read causes an *ECC_error* disrupting trap.
- A HW_corrected L2-cache/L3-cache data error as the result of a store queue exclusive request, or for a writeback or copyout, will generate a disrupting *ECC_error* trap.
- A HW_corrected L2-cache/L3-cache tag error will cause a disrupting *ECC_error* trap.
- If CEEN is clear, the error is logged in the AFSR and no trap is generated. This bit enables the errors associated with the AFSR.EDC, WDC, CPC, IVC, CE, EMC, and IMC bits and AFSR_EXT.L3_WDC, L3_CPC, L3_EDC bits.

Note – FSAPE in the UltraSPARC IV processor is renamed to FISAPE in the UltraSPARC IV+ processor to avoid the naming confusion between forcing Sun Fireplane Interconnect address parity error and forcing SDRAM address parity error.

7.3.3 AFSR Register and AFSR_EXT Register

The Asynchronous Fault Status Register (AFSR) is presented as two separate registers, the primary AFSR and the secondary AFSR. These registers have the same bit specifications throughout, but have different mechanisms for writing and clearing. The new primary Asynchronous Fault Status Extension register (AFSR_EXT) is added to log the L3-cache tag or L3-cache data ECC errors. The secondary AFSR2_EXT is added to extend the AFSR_2 register. The Asynchronous Fault Status Extension Register (AFSR_EXT) is presented as two separate registers, the primary AFSR_EXT and the secondary AFSR_EXT. These registers have the same bit specifications throughout, but have different mechanisms for writing and clearing.

Note – AFSR, AFSR_EXT, AFSR2, AFSR2_EXT, AFAR, AFAR2 are private ASI registers.

7.3.3.1 Primary AFSR and AFSR_EXT

The primary AFSR accumulates all errors from system bus and L2-cache that have occurred since its fields were last cleared. An extension of AFSR register, AFSR_EXT, accumulates all errors from L3-cache that have occurred since its fields were last cleared. The AFSR and AFSR_EXT are updated according to the policy described in Table 7-16, *Error Reporting Summary,* on page 192.

The primary AFSR is represented by the label AFSR1 in this document, where it is necessary to distinguish the registers. A reference to AFSR should be taken to mean “either primary or secondary AFSR”.

The primary AFSR_EXT is represented by the label AFSR1_EXT in this document, where it is necessary to distinguish the registers. A reference to AFSR_EXT should be taken to mean “either primary or secondary AFSR_EXT”.

7.3.3.2 Secondary AFSR

The secondary AFSR and secondary AFSR_EXT are intended to capture the first event that the processor sees among a closely connected series of errors. The secondary AFSR and secondary AFSR_EXT captures the first event that sets one of bits [62:33] of the primary AFSR and bits [11:0] of the primary AFSR_EXT. In the case there are multiple “first” errors arriving at exactly the same cycle, multiple error bits will be captured at secondary AFSR/AFSR_EXT.

The secondary AFSR and AFSR_EXT are unfrozen/enabled to capture a new event when bits [62:54] and [51:33] of the primary AFSR and bits [11:0] of the primary AFSR_EXT are 0. Note that AFSR1.PRIV and AFSR1.ME do not have to be 0 in order to unlock the secondary AFSR.

The secondary AFSR and AFSR_EXT never accumulates, nor does any overwrite policy apply. To clear the secondary AFSR bits, software should clear bits [62:33] of the primary AFSR and bits [11:0] of the primary AFSR_EXT.

The secondary AFSR and AFSR_EXT enable diagnosis software to determine the source of an error. If the processor reads a uncorrectable data ECC error from the system bus into the L2-cache, and then a writeback event copies the same error out to the system bus again before the diagnosis software executes, the primary AFSR/AFSR_EXT cannot show whether the original event came from the system bus or the L2-cache. The secondary AFSR, in this case, would show that it came from the system bus.

The secondary AFSR is represented by the label AFSR2 in this manual, where it is necessary to distinguish primary and secondary registers. The secondary AFSR_EXT enables diagnosis software to determine the source of an error. The secondary AFSR_EXT is represented by the label AFSR2_EXT in this document, where it is necessary to distinguish primary and secondary registers.

7.3.3.3 AFSR Fields

Bit [53], the accumulating multiple-error (ME) bit, is set in AFSR1 or AFSR1_EXT when an uncorrectable error occurs, or a SW_correctable error occurs, and the AFSR1 or AFSR1_EXT status bit to report that error is already set to 1. Multiple errors of different types are indicated by setting more than one of the AFSR1 or AFSR1_EXT status bits.

Note – AFSR1.ME is not set if multiple HW_corrected errors with the same status bit occur: only uncorrectable and SW_correctable. AFSR2.ME can never be set, because if any bit is already set in AFSR1, AFSR2 is already frozen.

AFSR1.ME is not set by multiple ECC errors which occur within a single 64-byte system bus transaction. The first ECC error in a 16 byte correction word will be logged. Further errors of the same type in following 16 byte words from the same 64 byte transaction are ignored.

Bit [52], the accumulating privilege-error (PRIV), is set when an error is detected at a time when PSTATE.PRIV = 1. If this bit is set for an uncorrectable error, system state has been corrupted. (The corruption may be limited and may be recoverable if this occurs as the result of code as described in *Special Access Sequence for Recovering Deferred Traps* on page 255.)

PRIV accumulates the privilege state of the processor at the time errors are detected, until software clears PRIV.

PRIV accumulates the state of the PSTATE.PRIV bit at the time the event is detected, rather than the PSTATE.PRIV value associated with the instruction which caused the access which returns the error.

Note – `MEMBAR #Sync` is required before an ASI store which changes the PSTATE.PRIV bit to act as an error barrier between previous transactions that were launched with a different PRIV state. This ensures that privileged operations which fault will be recorded as privileged in AFSR.PRIV.

AFSR1.PRIV accumulates as specified in TABLE 7-16. AFSR2.PRIV captures privilege state as specified in this table, but only for the first error encountered.

Bits [51:50], PERR and IERR, indicate that either an internal inconsistency has occurred in the system interface logic or that a protocol error has occurred on the system bus. If either of these conditions occurs the processor will assert its ERROR output pin. The AFSR may be read after a reset event used to recover from the error condition to discover the cause.

The IERR status bit indicates that an event has been detected which is likely to have its source inside the processor reporting the problem. The PERR status bit indicates that the error source may well be elsewhere in the system, not in the processor reporting the problem. However, this differentiation cannot be perfect. These are merely likely outcomes. Further error recording elsewhere in the system is desirable for accurate diagnosis.

Bits [62:54, 49:33] are sticky error bits that record the most recently detected errors. Each sticky bit in AFSR1 accumulates errors that have been detected since the last write to clear the bit. Unless two errors in AFSR or AFSR_EXT are detected in the same clock cycle, at most one of these bits can be set in AFSR2 and AFSR2_EXT.

Bits [19:16] contain the data microtag ECC syndrome captured on a system bus microtag ECC error. The syndrome field captures the status of the first occurrence of the highest priority error according to the M_SYND overwrite policy. After the AFSR1 sticky bit, corresponding to the error for which the M_SYND is reported, is cleared, the contents of the M_SYND field will be cleared.

Bits[8:0] contain the data ECC syndrome. The syndrome field captures the status of the first occurrence of the highest priority error according to the E_SYND overwrite policy. After the AFSR sticky bit, corresponding to the error for which the E_SYND is reported, is cleared, the contents of the E_SYND field will be cleared. E_SYND applies only to system bus, L2-cache, and L3-cache data ECC errors. It is not updated for L2-cache tag ECC errors or L3-cache tag ECC errors.

7.3.3.4 AFSR_EXT Fields

Bits [11:0] are sticky error bits that record the most recently detected errors. Each sticky bit in AFSR1_EXT accumulates errors that have been detected since the last write to clear the bit. Unless two errors are detected in the same clock cycle, at most one of these bits can be set in AFSR2_EXT.

7.3.3.5 Clearing the AFSR and AFSR_EXT

AFSR1/AFSR1_EXT must be explicitly cleared by software; it is not cleared automatically by a read. Writes to the AFSR1 RW1C bits ([62:33]) with particular bits set will clear the corresponding bits in both AFSR1 and AFSR2. Writes to the AFSR1_EXT RW1C bits ([11:0]) with particular bits set will clear the corresponding bits in both AFSR1_EXT and AFSR2_EXT. Bits associated with disrupting traps must be cleared, before re-enabling interrupts by setting PSTATE.IE, to prevent multiple traps for the same error. Writes to AFSR1 bits with particular bits clear will not affect the corresponding bits in either AFSR. The syndrome fields are read only and writes to these fields are ignored.

Each of the AFSR2 bits[62:33] is cleared automatically when software clears the corresponding AFSR1 bit. Each of the AFSR2_EXT bits [11:0] is cleared automatically when software clears the corresponding AFSR1_EXT bit. AFSR2/AFSR2_EXT is read-only. AFSR2 only becomes available (*unfrozen*) to capture a new hardware error when bits [62:54, 51:33] of AFSR1 and bits [11:0] of AFSR1_EXT are zero.

Note – Software should clear both the error bit and the PRIV bit in the AFSR register at the same time.

If software attempts to clear error bits at the same time as an error occurs, one of two events will occur:

1. The clear will appear to happen before the error occurs. The state of AFSR1/AFSR1_EXT syndrome, ME, PRIV and sticky bits, and the state of AFAR1, will all be consistent with the clear having happened before the error occurs. If the clear zeroed all bits[62:54, 51:33] of AFSR1 and bits [11:0] AFSR1_EXT, then AFSR2 and AFSR2_EXT and AFAR2 will capture the new error.
2. The clear will appear to happen after the error occurs. The state of AFSR1/AFSR_EXT syndrome, ME, PRIV and sticky bits, and the state of AFAR1, will all be consistent with the clear having happened after the error occurs. AFSR2 and AFSR2_EXT and AFAR2 will not have been updated with the new error information.

The PERR and IERR bits must be cleared by software by writing a “1” to the corresponding bit positions.

When multiple events have been logged by the various bits in AFSR1 or AFSR1_EXT, at most one of these events will have its status captured in AFAR1. AFAR1 will be unlocked and available to capture the address of another event as soon as the one bit is cleared in AFSR1 or AFSR1_EXT which corresponds to the event logged in AFAR1. For example, if AFSR1.CE is detected, then AFSR1.UE (which overwrites AFAR1), and AFSR1.UE is cleared but not AFSR1.CE, then AFAR1 will be unlocked and ready for another event, even though AFSR1.CE is still set.

This same argument also applies to primary AFSR1.M_SYND and AFSR1.E_SYND fields. Each field will be unlocked and available for further error capture when the specific AFSR1 status bit is cleared, associated with the event logged in the field.

AFAR2 captures the address associated with the error stored in AFSR2/AFSR2_EXT. AFSR2/AFSR2_EXT and AFAR2 are frozen with the status captured on the first error which sets one of bits [62:33] of AFSR1 and bits [11:0] AFSR1_EXT. No overwrites occur in AFSR2/AFSR2_EXT and AFAR2.

AFSR1: ASI== $4C_{16}$, VA[63:0]==0x0, private

Name: `ASI_ASYNC_FAULT_STATUS`

AFSR2: ASI== $4C_{16}$, VA[63:0]==0x8, private

Name: ASI_ASYNC_FAULT_STATUS2

TABLE 7-7 Asynchronous Fault Status Register ***(1 of 2)***

Bits	Field	Description	RW
[63]	*Reserved*	Reserved for future implementation.	R
[62]	TUE_SH	Uncorrectable L2-cache tag UE due to copyback, or tag update from foreign Sun Fireplane device or snoop request.	RW1C
[61]	IMC	Single bit ECC error on system bus microtag for interrupt vector.	RW1C
[60]	IMU	Multi-bit microtag ECC error on system bus microtag for interrupt vector.	RW1C
[59]	DTO	Unmapped error from system bus for prefetch queue or store queue read operation.	RW1C
[58]	DBERR	Bus error from system bus for prefetch queue or store queue read operation.	RW1C
[57]	THCE	HW_corrected L2-cache tag ECC error.	RW1C
[56]	*Reserved*	Reserved for future implementation.	R
[55]	TUE	Uncorrectable L2-cache tag ECC error due to logical processor specific tag access.	RW1C
[54]	DUE	Uncorrectable system bus data ECC for prefetch queue or store queue read operation.	RW1C
[53]	ME	Multiple error of same type occurred.	RW1C
[52]	PRIV	Privileged state error has occurred.	RW1C
[51]	PERR	System interface protocol error.	RW1C
[50]	IERR	Internal processor error.	RW1C
[49]	ISAP	System request parity error on incoming address.	RW1C
[48]	EMC	HW_corrected system bus microtag ECC error.	RW1C
[47]	EMU	Uncorrectable system bus microtag ECC error.	RW1C
[46]	IVC	HW_corrected system bus data ECC error for read of interrupt vector.	RW1C
[45]	IVU	Uncorrectable system bus data ECC error for read of interrupt vector.	RW1C
[44]	TO	Unmapped error from system bus.	RW1C
[43]	BERR	Bus error response from system bus.	RW1C
[42]	UCC	SW_correctable L2-cache ECC error for instruction fetch, load-like or atomic instruction.	RW1C
[41]	UCU	Uncorrectable L2-cache data ECC error for instruction fetch, load-like or atomic instruction.	RW1C
[40]	CPC	HW_corrected L2-cache data ECC error for copyout Note: This bit is not set if the copyout hits in the writeback buffer. Instead, the WDC bit is set.	RW1C
[39]	CPU	Uncorrectable L2-cache data ECC error for copyout Note: This bit is not set if the copyout hits in the writeback buffer. Instead, the WDU bit is set.	RW1C

TABLE 7-7 Asynchronous Fault Status Register *(2 of 2)*

Bits	Field	Description	RW
[38]	WDC	HW_corrected L2-cache data ECC error for writeback.	RW1C
[37]	WDU	Uncorrectable L2-cache data ECC error for writeback.	RW1C
[36]	EDC	HW_corrected L2-cache data ECC error for store or block load or prefetch queue operation.	RW1C
[35]	EDU	Uncorrectable L2-cache data ECC error for store or block load or prefetch queue operation.	RW1C
[34]	UE	Uncorrectable system bus data ECC error for read of memory or I/O for instruction fetch, load-like, block load or atomic operations.	RW1C
[33]	CE	Correctable system bus data ECC error for any read of memory or I/O.	RW1C
[20:32]	*Reserved*	Reserved for future implementation.	R
[19:16]	M_SYND	System bus microtag ECC syndrome	R
[15:9]	*Reserved*	Reserved for future implementation.	R
[8:0]	E_SYND	System bus or L2-cache or L3-cache data ECC syndrome.	R

Note – For system bus read access error due to prefetch queue or store queue read operation, a DUE, DTO, or DBERR is set instead of UE, TO, or BERR, respectively.

TABLE 7-7 describes AFSR1. AFSR2 is identical except that all bits are read-only, and AFSR2.ME is always 0.

AFSR1_EXT: ASI== $4C_{16}$, VA[63:0]==0x10, private

Name: `ASI_ASYNC_FAULT_STATUS_EXT`

AFSR2_EXT: ASI== $4C_{16}$, VA[63:0]==0x18, private

Name: `ASI_ASYNC_FAULT_STATUS_EXT`

TABLE 7-8 Asynchronous Fault Status Extension Register *(1 of 2)*

Bits	Field	Description	RW
[63:14]	*Reserved*	Reserved for future implementation.	R
[13]	RED_ERR	e-Fuse error for I-cache/D-Cache/DTLBs/I-TLB SRAM redundancy, or shared L2-cache/L3-cache tag /L2-cache data SRAM redundancy.	RW1C
[12]	EFA_PAR_ERR	e-Fuse parity error.	RW1C
[11]	L3_MECC	Both 16-byte data of L3-cache data access have ECC error (either correctable or uncorrectable ECC error).	RW1C
[10]	L3_THCE	Single bit ECC error on L3-cache tag access.	RW1C

TABLE 7-8 Asynchronous Fault Status Extension Register *(2 of 2)*

Bits	Field	Description	RW
[9]	L3_TUE_SH	Multiple-bit ECC error on L3-cache tag access due to copyback, or tag update from foreign Sun Fireplane device, snoop request.	RW1C
[8]	L3_TUE	Multiple-bit ECC error on L3-cache tag access due to private tag access.	RW1C
[7]	L3_EDC	Single bit ECC error on L3-cache data access for P-cache and W-cache request.	RW1C
[6]	L3_EDU	Multiple-bit ECC error on L3-cache data access for P-cache and W-cache request.	RW1C
[5]	L3_UCC	Single bit ECC error on L3-cache data access for I-cache and D-cache request.	RW1C
[4]	L3_UCU	Multiple-bit ECC error on L3-cache data access for I-cache and D-cache request.	RW1C
[3]	L3_CPC	Single bit ECC error on L3-cache data access for copyout.	RW1C
[2]	L3_CPU	Multiple-bit ECC error on L3-cache data access for copyout.	RW1C
[1]	L3_WDC	Single bit ECC error on L3-cache data access for writeback.	RW1C
[0]	L3_WDU	Multiple-bit ECC error on L3-cache data access for writeback.	RW1C

TABLE 7-8 describes AFSR1_EXT. AFSR2_EXT is identical except that all bits are read-only.

7.3.4 ECC Syndromes

ECC syndromes are captured on system bus data and microtag ECC errors, and on L2-cache data ECC errors, and L3-cache data ECC errors. Syndromes are not captured for L2-cache tag ECC errors and L3-cache tag ECC errors. The syndrome tables for system bus data, L2-cache data, L3-cache data and system bus microtags are given here.

7.3.4.1 E_SYND

The AFSR.E_SYND field contains a 9 bit value that indicates which data bit of a 128-bit quad-word contains a single bit error. This field is used to report the ECC syndrome for system bus, L2-cache tag and data, and L3-cache tag and data ECC errors of all types: HW_corrected, SW_correctable and uncorrectable.

TABLE 7-10 shows the 9 bit ECC syndromes that correspond to a single bit error for each of the 128 data bits. To locate a syndrome in the table use the low order 3 bits of the data bit number to find the column and the high order 4 bits of the data bit number to find the row. For example data bit number 126 is at column 0x6, row 0xf and has a syndrome of 0x1c9.

TABLE 7-9 Key to interpreting TABLE 7-10

Interpretation	Example
Data bit number, decimal	126
Data bit number, hexadecimal	0x7e
Data bit number, 7 bit binary	111 1110
High 4 bits, binary	1111
Low 3 bits, binary	110
High 4 bits, hexadecimal row number	0xf
Low 3 bits, hexadecimal column number	0x6
Syndrome returned for 1bit error in data bit 126	0x1c9

TABLE 7-10 Data Single Bit Error ECC Syndromes

Column - Low 3 bits Row - High 4 bits	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
0x0	03b	127	067	097	10f	08f	04f	02c
0x1	147	0c7	02f	01c	117	032	08a	04a
0x2	01f	086	046	026	09b	08c	0c1	0a1
0x3	01a	016	061	091	052	00e	109	029
0x4	02a	019	105	085	045	025	015	103
0x5	031	00d	083	043	051	089	023	007
0x6	0b9	049	013	0a7	057	00b	07a	187
0x7	0f8	11b	079	034	178	1d8	05b	04c
0x8	064	1b4	037	03d	058	13c	1b1	03e
0x9	1c3	0bc	1a0	1d4	1ca	190	124	13a
0xa	1c0	188	122	114	184	182	160	118
0xb	181	150	148	144	142	141	130	0a8
0xc	128	121	0e0	094	112	10c	0d0	0b0
0xd	10a	106	062	1b2	0c8	0c4	0c2	1f0
0xe	0a4	0a2	098	1d1	070	1e8	1c6	1c5
0xf	068	1e4	1e2	1e1	1d2	1cc	1c9	1b8

TABLE 7-11, shows the 9 bit ECC syndromes that correspond to a single bit error for each of the 9 ECC check bits for the L2-cache, L3-cache and system bus error correcting codes used for data.

TABLE 7-11 Data Check Bit Single Bit Error Syndrome

Check bit number	AFSR.E_SYND
0	0x001
1	0x002
2	0x004
3	0x008
4	0x010
5	0x020
6	0x040
7	0x080
8	0x100

Other syndromes found in the AFSR.E_SYND field indicate either no error (syndrome 0) or a multi-bit error has occurred.

TABLE 7-12, gives the mapping from all E_SYND ECC syndromes to the indicated event.

Legend for TABLE 7-12:
--- - no error
0-127 - Data single bit error, data bit 0-127
C0 - C8 - ECC check single bit error, check bit 0-8
M2 - Probable Double bit error within a nibble
M3 - Probable Triple bit error within a nibble
M4 - Probable Quad bit error within a nibble
M - multi-bit error

Three syndromes in particular from TABLE 7-12, are useful. These are the syndromes corresponding to the three different deliberately inserted bad ECC conditions, the signalling ECC codes, used by the processor.

For a DSTAT = 2 or 3 (BERR or DBERR) event from the system bus for a cacheable load, data bits [1:0] are inverted in the data stored in the L2-cache. The syndrome seen when one of these signalling words is read will be 0x11c.

For an uncorrectable data ECC error from the L3-cache, data bits[127:126] are inverted in data sent to the system bus as part of a writeback or copyout. The syndrome seen when one of these signalling words is read will be 0x071.

For an uncorrectable data ECC error from the L2-cache, data bits[127:126] are inverted in data sent to the system bus as part of a copyout. The syndrome seen when one of these signalling words is read will be 0x071.

For uncorrectable data ECC error on the L2-cache or L3-cache read done to complete a store queue exclusive request, the uncorrectable ECC error information is stored. When the line is evicted from W-cache back to L2-cache, if the uncorrectable ECC error information is asserted, ECC check bits [1:0] are inverted in the data written back to the L2-cache. The syndrome seen when one of these signalling words is read will be 0x003.

For uncorrectable on-chip L2-cache tag or L3-cache tag ECC error, error pin is asserted, and the domain should be reset. The copyout or writeback might be dropped.

TABLE 7-12 ECC Syndromes

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	---	C0	C1	M2	C2	M2	M3	47	C3	M2	M2	53	M2	41	29	M
1	C4	M	M	50	M2	38	25	M2	M2	33	24	M2	11	M	M2	16
2	C5	M	M	46	M2	37	19	M2	M	31	32	M	7	M2	M2	10
3	M2	40	13	M2	59	M	M2	66	M	M2	M2	0	M2	67	71	M
4	C6	M	M	43	M	36	18	M	M2	49	15	M	63	M2	M2	6
5	M2	44	28	M2	M	M2	M2	52	68	M2	M2	62	M2	M3	M3	M4
6	M2	26	106	M2	64	M	M2	2	120	M	M2	M3	M	M3	M3	M4
7	116	M2	M2	M3	M2	M3	M	M4	M2	58	54	M2	M	M4	M4	M3
8	C7	M2	M	42	M	35	17	M2	M	45	14	M2	21	M2	M2	5
9	M	27	M	M	99	M	M	3	114	M2	M2	20	M2	M3	M3	M
a	M2	23	113	M2	112	M2	M	51	95	M	M2	M3	M2	M3	M3	M2
b	103	M	M2	M3	M2	M3	M3	M4	M2	48	M	M	73	M2	M	M3
c	M2	22	110	M2	109	M2	M	9	108	M2	M	M3	M2	M3	M3	M
d	102	M2	M	M	M2	M3	M3	M	M2	M3	M3	M2	M	M4	M	M3
e	98	M	M2	M3	M2	M	M3	M4	M2	M3	M3	M4	M3	M	M	M
f	M2	M3	M3	M	M3	M	M	M	56	M4	M	M3	M4	M	M	M
10	C8	M	M2	39	M	34	105	M2	M	30	104	M	101	M	M	4
11	M	M	100	M	83	M	M2	12	87	M	M	57	M2	M	M3	M
12	M2	97	82	M2	78	M2	M2	1	96	M	M	M	M	M	M3	M2
13	94	M	M2	M3	M2	M	M3	M	M2	M	79	M	69	M	M4	M
14	M2	93	92	M	91	M	M2	8	90	M2	M2	M	M	M	M	M4
15	89	M	M	M3	M2	M3	M3	M	M	M	M3	M2	M3	M2	M	M3
16	86	M	M2	M3	M2	M	M3	M	M2	M	M3	M	M3	M	M	M3
17	M	M	M3	M2	M3	M2	M4	M	60	M	M2	M3	M4	M	M	M2
18	M2	88	85	M2	84	M	M2	55	81	M2	M2	M3	M2	M3	M3	M4
19	77	M	M	M	M2	M3	M	M	M2	M3	M3	M4	M3	M2	M	M
1a	74	M	M2	M3	M	M	M3	M	M	M	M3	M	M3	M	M4	M3
1b	M2	70	107	M4	65	M2	M2	M	127	M	M	M	M2	M3	M3	M
1c	80	M2	M2	72	M	119	118	M	M2	126	76	M	125	M	M4	M3
1d	M2	115	124	M	75	M	M	M3	61	M	M4	M	M4	M	M	M
1e	M	123	122	M4	121	M4	M	M3	117	M2	M2	M3	M4	M3	M	M
1f	111	M	M	M	M4	M3	M3	M	M	M	M3	M	M3	M2	M	M

7.3.4.2 M_SYND

The AFSR.M_SYND field contains a 4 bit ECC syndrome for the 3 bit microtags of the system bus. TABLE 7-13 shows the 4 bit syndrome corresponding to a single bit error in each of the microtag data or correction bits.

TABLE 7-13 Microtag Single Bit Error ECC Syndromes

Bit Number	AFSR.M_SYND
microtag Data 0	0x7
microtag Data 1	0xB
microtag Data 2	0xD
microtag ECC 0	0x1
microtag ECC 1	0x2
microtag ECC 2	0x4
microtag ECC 3	0x8

A complete microtag syndrome table is shown in TABLE 7-13 and TABLE 7-14.

TABLE 7-14 Syndrome table for Microtag ECC

Syndrome[3:0]	Error Indication
0x0	None
0x1	microtag ECC 0
0x2	microtag ECC 1
0x3	Double bit (UE)
0x4	microtag ECC 2
0x5	Double bit (UE)
0x6	Double bit (UE)
0x7	microtag Data 0
0x8	microtag ECC 3
0x9	Double bit (UE)
0xA	Double bit (UE)
0xB	microtag Data 1
0xC	Double bit (UE)
0xD	microtag Data 2
0xE	Multiple bit (UE)
0xF	Double bit (UE)

The M_SYND field is locked by the AFSR.EMC, EMU, IMC, and IMU bits. The E_SYND field is locked by the AFSR.UE, CE, UCU, UCC, EDU, EDC, WDU, WDC, CPU, CPC, IVU, IVC, and L3_UCC, L3_UCU, L3_EDC, L3_EDU, L3_CPC, L3_CPU, L3_WDC, and L3_WDU bits. So, a data ECC error can lead to the data ECC syndrome being recorded in E_SYND, perhaps with a CE status, then a later microtag ECC error event can store the microtag ECC syndrome in M_SYND, perhaps with an EMC status. The two are independent.

7.3.5 Asynchronous Fault Address Register

Primary and secondary AFARs are provided, AFAR1 and AFAR2, associated with AFSR1 and AFSR2. AFAR1 works with the status captured in AFSR1 according to the overwrite policy described in *Overwrite Policy* on page 198. AFAR2 is captured at the time that AFSR2 is captured, and specifically reflects the address of the transaction which causes AFSR2 to be frozen. AFAR2 operates no overwrite policy. AFAR2 becomes available for further updates exactly as AFSR2 does, when bits [62:54, 51:33] of AFSR1 and bits[11:0] of AFSR1_EXT are cleared.

AFAR1 is captured when one of the AFSR1 error status bits that capture address is set (see TABLE 7-16, for details). The address corresponds to the first occurrence, of the highest priority error that captures address according to the AFAR1/AFSR1_EXT overwrite policy, in AFSR1/AFSR1_EXT. Address capture in AFAR1 is reenabled by clearing the corresponding error bit in AFSR1. See above at *Clearing the AFSR and AFSR_EXT* on page 182 for a description of behavior when clearing occurs at the same time as an error.

AFAR1: ASI== 0x4D, VA[63:0]==0x0, private

AFAR2: ASI== 0x4D, VA[63:0]==0x8, private

Name: `ASI_ASYNC_FAULT_ADDRESS`

TABLE 7-15 Asynchronous Fault Address Register

Bits	Field	Description	RW
[63:43]	*Reserved*	Reserved for future implementation.	R
[42:4]	PA[42:4]	Physical address of faulting 16 byte component (bits [5:4] isolate the fault to 128 bit sub-unit within a 512 bit coherency block)	RW (AFAR1, R (AFAR2)
[3:0]	*Reserved*	Reserved for future implementation.	R

TABLE 7-15 describes AFAR1. AFAR2 differs only in being read-only.

PA: Address information for the most recently captured error.

In the event of multiple errors within a 64 byte block, AFAR captures only the first-detected highest priority error.

When there is an asynchronous error and AFAR2 is unfrozen (i.e., AFSR1 bits [62:54, 51:33] and AFSR1_EXT bits[11:0] are zero), a write to AFAR1 will write to both AFAR1 and AFAR2.

7.4 Error Reporting Summary

TABLE 7-16 Error Reporting Summary *(1 of 4)*

Error Event	AFSR status bit	Trap taken	Trap controlled by	FERR?	M_SYND	E_SYND	AFAR Priority	Set PRIV?	Set ME?	Flush	Shared/Private
System unrecoverable error other than CPQ_TO, NCPQ_TO, TID_TO	PERR	none	-	1	0	0	0	0	0	0	Shared
System unrecoverable error, CPQ_TO, NCPQ_TO, TID_TO	PERR	none	-	1	0	0	5	0	0	0	Shared
Internal unrecoverable error	IERR	none	-	1	0	0	0	0	0	0	Shared
Parity error during transfer from e-Fuse array to repairable SRAM array	EFA_PAR_ERR	none	-	1	0	0	0	0	0	0	AFSRs in all running LP's are logged
Bit in Redundancy register is flipped for I-cache, or D-cache, or D-TLB, or I-TLB	RED_ERR	none	-	1	0	0	0	0	0	0	Private
Bit in Redundancy register is flipped for L2-cache tag/data, or L3-cache tag/data	RED_ERR	none	-	1	0	0	0	0	0	0	AFSRs in all running LP's are logged
Incoming system address parity error	ISAP	none	-	1	0	0	0	0	1	0	Shared
Uncorrectable system bus data ECC: instruction fetch	UE	I	NCEEN		0	2	3	1	1	1	Private
Uncorrectable system bus data ECC: load, block load, atomic instructions	UE	D	NCEEN		0	2	3	1	1	1	Private
Uncorrectable system bus data ECC: store queue RTO or prefetch queue read	DUE	C	NCEEN		0	2	3	1	1	0	Private
Uncorrectable system bus data ECC: interrupt vector fetch	IVU	C	NCEEN		0	2	0	1	1	0	Private
HW_corrected system bus data ECC: all but interrupt vector fetch	CE	C	CEEN		0	1	2	1	0	0	Private
HW_corrected system bus data ECC: interrupt vector fetch	IVC	C	CEEN		0	1	0	1	0	0	Private
Uncorrectable system bus microtag ECC: instruction fetch	EMU	I	NCEEN	1	2	0	3	1	1	1	Private
Uncorrectable system bus microtag ECC: All but instruction fetch	EMU	D	NCEEN	1	2	0	3	1	1	1	Private
HW_corrected system bus microtag ECC	EMC	C	CEEN		1	0	2	1	0	0	Private

TABLE 7-16 Error Reporting Summary *(2 of 4)*

Error Event	AFSR status bit	Trap taken	Trap controlled by	FERR?	M_SYND	E_SYND	AFAR Priority	Set PRIV?	Set ME?	Flush	Shared/Private
Uncorrectable L2-cache data ECC: instruction fetch (critical 32-byte and non-critical 32-byte), load (critical 32-byte), atomic instruction (critical 32-byte and non-critical 32-byte)	UCU	FC	UCEEN		0	3	4	1	1	2	Private
Uncorrectable L2-cache data ECC: store queue or prefetch queue operation or load-like instruction (non-critical 32-byte)	EDU	C	NCEEN		0	2	3	0	1	0	Private
Uncorrectable L2-cache data ECC: Block Load	EDU	D	NCEEN		0	2	3	1	1	0	Private
Uncorrectable L2-cache data ECC: writeback	WDU	C	NCEEN		0	2	3	0	1	0	Private
Uncorrectable L2-cache data ECC: copyout	CPU	C	NCEEN		0	2	3	0	1	0	Shared
SW_correctable L2-cache data ECC: instruction fetch (critical 32-byte and non-critical 32-byte), load (critical 32-byte), atomic instruction (critical 32-byte)	UCC	FC	UCEEN		0	3	4	1	1	2	Private
HW_corrected L2-cache data ECC: store queue or prefetch queue operation or load instruction (non-critical 32-byte) or atomic instruction (non-critical 32-byte)	EDC	C	CEEN		0	1	2	0	0	0	Private
HW_corrected L2-cache data ECC: Block Load	EDC	C	CEEN		0	1	2	1	0	0	Private
HW_corrected L2-cache data ECC: writeback	WDC	C	CEEN		0	1	2	0	0	0	Private
HW_corrected L2-cache data ECC: copyout	CPC	C	CEEN		0	1	2	0	0	0	Shared
Uncorrectable L2-cache tag ECC: SIU tag update, or copyback from foreign bus transactions, or snoop operations	TUE_SH	C	NCEEN, L2_tag_ECC_en	1	0	0	4	1	1	0	Shared
Uncorrectable L2-cache tag ECC: all other L2-cache tag accesses	TUE	C	NCEEN, L2_tag_ECC_en	1	0	0	4	1	1	0	Private
HW_corrected L2-cache tag ECC: fetch, load, atomic instruction, writeback, copyout, block load, store queue or prefetch queue read*	THCE	C	CEEN, L2_tag_ECC_en		0	0	2	0	0	0	Private
SW_correctable I-cache data or tag parity: instruction fetch	none	IP	DCR.IPE		0	0	0	0	0	3	Private
SW_correctable D-cache data parity: load-like instruction	none	DP	DCR.DPE		0	0	0	0	0	3	Private

TABLE 7-16 Error Reporting Summary *(3 of 4)*

Error Event	AFSR status bit	Trap taken	Trap controlled by	FERR?	M_SYND	E_SYND	AFAR Priority	Set PRIV?	Set ME?	Flush	Shared/Private
SW_correctable D-cache tag parity: load-like or store-like instruction	none	DP	DCR.DPE		0	0	0	0	0	3	Private
HW_corrected I-cache or D-cache tag parity: snoop invalidation	none	none	-		0	0	0	0	0	0	Private
DSTAT = 2 or 3 response, “bus error”: instruction fetch	BERR	I	NCEEN		0	0	1	1	1	0	Private
DSTAT = 2 or 3 response, “bus error”: load-like, block load, atomic instructions, interrupt vector fetch operations	BERR	D	NCEEN		0	0	1	1	1	0	Private
DSTAT = 2 or 3 response, “bus error”: prefetch queue and store queue operations	DBERR	C	NCEEN		0	0	1	1	1	0	Private
No MAPPED response, “time-out”: instruction fetch	TO	I	NCEEN		0	0	1	1	1	0	Private
No MAPPED response, “time-out”: load-like, block load, atomic, store queue write (WS, WBIO, WIO), writeback, block store instructions, interrupt vector transmit operations	TO	D	NCEEN		0	0	1	1	1	0	Private
No MAPPED response, “time-out”: prefetch queue and store queue read operations	DTO	C	NCEEN		0	0	1	1	1	0	Private
Uncorrectable system bus microtag ECC: interrupt vector fetch	IMU	C	NCEEN	1	2	0	0	1	1	0	Private
Correctable system bus microtag ECC: interrupt vector fetch	IMC	C	CEEN		1	0	0	1	0	0	Private
Uncorrectable L3-cache data ECC: writeback	L3_WDU	C	NCEEN		0	2	3	0	1	0	Private
HW_corrected L3-cache data ECC: writeback	L3_WDC	C	CEEN		0	1	2	0	0	0	Private
Uncorrectable L3-cache data ECC: copyback	L3_CPU	C	NCEEN		0	2	3	0	1	0	Shared

TABLE 7-16 Error Reporting Summary *(4 of 4)*

Error Event	AFSR status bit	Trap taken	Trap controlled by	FERR?	M_SYND	E_SYND	AFAR Priority	Set PRIV?	Set ME?	Flush	Shared/Private
HW_corrected L3-cache data ECC: copyback	L3_CPC	C	CEEN		0	1	2	0	0	0	Shared
Uncorrectable L3-cache data ECC: instruction fetch (critical 32-byte and non-critical 32-byte), load (critical 32-byte), atomic instruction (critical 32-byte)	L3_UCU	FC	UCEEN		0	3	4	1	1	2	Private
SW_correctable L3-cache data ECC: instruction fetch (critical 32-byte and non-critical 32-byte), load (critical 32-byte), atomic instruction (critical 32-byte)	L3_UCC	FC	UCEEN		0	3	4	1	1	2	Private
Uncorrectable L3-cache data ECC: store queue or prefetch queue operation or load instruction (critical 32-byte) or atomic instruction (non-critical 32-byte)	L3_EDU	C	NCEEN		0	2	3	0	1	0	Private
Uncorrectable L3-cache data ECC: block load operation	L3_EDU	D	NCEEN		0	2	3	1	1	0	Private
HW_corrected L3-cache data ECC: store queue or prefetch queue operation or load instruction (critical 32-byte) or atomic instruction (non-critical 32-byte)	L3_EDC	C	CEEN		0	1	2	0	0	0	Private
HW_corrected L3-cache data ECC: block load operation	L3_EDC	C	CEEN		0	1	2	1	0	0	Private
Uncorrectable L3-cache tag ECC: SIU tag update, or copyback from foreign bus transactions, or snoop operations	L3_TUE_SH	C	NCEEN, ET_ECC_en	1	0	0	4	1	1	0	Shared
Uncorrectable L3-cache tag ECC: all other L3-cache tag accesses	L3_TUE	C	NCEEN, ET_ECC_en	1	0	0	4	1	1	0	Private
HW_corrected L3-cache tag ECC: writeback, copyout, block load, store queue or prefetch queue read**	L3_THCE	C	CEEN, ET_ECC_en		0	0	2	0	0	0	Private

The notes below provide the detailed explanation of the entries in the TABLE 7-16.

Note – When copyout from L2-cache or snoop or tag update due to foreign transaction encounters HW_corrected L2-cache tag ECC error, CMT error steering register will be used to decide which logical processor to log THCE.

When copyout from L3-cache or snoop or tag update due to foreign transaction encounters HW_corrected L3-cache tag ECC error, CMT error steering register will be used to decide which logical processor to log L3_THCE.

When L2_tag_ecc_en bit of `ASI_L2CACHE_CONTROL` (ASI 0x6D) is set to 0, no L2-cache tag error is reported. This applies to TUE_SH, TUE, and THCE.

When ET_ECC_en bit of `ASI_L3CACHE_CONTROL` (ASI 0x75) is set to 0, no L3-cache tag error is reported. This applies to L3_TUE_SH, L3_TUE, and L3_THCE.

When IPE or DPE bit of Dispatch Control Register (DCR, ASR 0x12) is set to 0, no I-cache or D-cache data/tag parity error is reported, respectively.

Trap types:

I: *instruction_access_error* trap, because the error is always the result of an instruction fetch: always deferred.

D: *data_access_error* trap: always deferred.

C: *ECC_error* trap: always disrupting

FC: *fast_ECC_error* trap: always precise

IP: *icache_parity_error* trap: always precise

DP: *dcache_parity_error* trap: always precise

none: no trap is taken, processor continues normal execution.

FERR: fatal error. If there is a 1 in the FERR column, the processor will assert its ERROR output pin for this event. Detailed processor behavior not specified. It is expected that the system will reset the processor.

Priority:

All “priority” entries in the above TABLE 7-16 work as follows: AFAR1 and AFSR1/AFSR1_EXT have an overwrite policy. Associated with the AFAR1 and with the AFSR1.M_SYND and E_SYND fields is a separate stored “priority” for the data in the field. When AFSR1 and AFSR1_EXT are empty and no errors have been logged, the effective priority stored for each field is 0. Whenever an event to be logged in AFSR1/AFSR1_EXT or AFAR1 occurs, compare the priority specified for each field for that event to the priority stored internal to the processor for that field. If the priority for the field for the event is numerically higher than the priority stored internal to the processor for that field, update the field with the value appropriate for the event that has just occurred, and update the stored priority in the processor with the priority specified in the table for that field and new event.

Note – This implies that fields with a 0 priority in the above table are never stored for that event.

For instance, if first a UE occurs to capture AFSR1.E_SYND, then an EDU, the EDU doesn't update AFSR1.E_SYND because it has the same priority as UE. Trap handler software clears AFSR1.UE, but leaves AFSR1.EDU set. AFSR1.E_SYND will be unchanged. When a CE occurs, AFSR1.E_SYND will not be changed, since CE has lower priority than EDU.

PRIV:

A 1 in the "set PRIV" column implies that the specified event will set the AFSR.PRIV bit if the PSTATE.PRIV bit is 1 at the time the event is detected. A 0 implies that the event has no effect on the AFSR.PRIV bit. AFSR.PRIV accumulates the privilege status of the specified error events detected since the last time the AFSR.PRIV bit was cleared.

ME:

A 1 in the "set ME" column implies that the specified event will cause the AFSR.ME bit to be set if the status bit specified for that event is already set at the time the event happens. A 0 in the "set ME" column implies that multiple events will not cause the AFSR.ME bit to be set. AFSR.ME accumulates the multiple error status of the specified error events detected since the last time the AFSR.ME bit was cleared.

Flushing:

The "flush needed" column contains a 0 if a D-cache flush is never needed for correctness. It contains a 1 if a D-cache flush is needed only if the read access is to a cacheable address. It contains a 2 if a D-cache flush is always needed. It contains a 3 if an I-cache, D-cache, and P-cache flush is needed. Note that, for some of these errors, an L2-cache or L3-cache flush or a main memory update is desirable to eliminate errors still stored in L2-cache or L3-cache or DRAM. However, the system does not need these to ensure that the data stored in the caches does not lead to undetected data corruption. The entries in the table only deal with data correctness.

D-cache flushes should not be needed for *instruction_access_error* traps, but it is simplest to invalidate the D-cache for both *instruction_access_error* and *data_access_error* traps.

Shared/Private:

Shared/Private column specifies if the corresponding error event can be traced back to the logical processor that caused the error. If the error can be traced back to a particular logical processor, it is listed as a private event. If the error cannot be traced back to a particular logical processor, it is listed as a shared event. For shared error events, the CMT Error Steering register (`ASI_CMP_ERROR_STEERING`) determines which logical processor the error should be reported to. See *CMT Error Steering Register (ASI_CMP_ERROR_STEERING)* on page 177 for details about how the CMT Error Steering register is used to report shared errors.

7.5 Overwrite Policy

This section describes the overwrite policy for error status when multiple errors have occurred. Errors are captured in the order that they are detected, not necessarily in program order. This policy applies only to AFSR1 and AFAR1. AFAR2 and AFSR2 have no overwrite mechanism, they are either frozen or unfrozen. AFSR2 and AFAR2 capture the first event after AFSR1 status lock bits are all cleared.

The overwrite policies are described by the “priority” entries in TABLE 7-16. The descriptions here set the policies out at length.

For the behavior when errors arrive at the same time as the AFSR is being cleared by software, see *Clearing the AFSR and AFSR_EXT* on page 182.

7.5.1 AFAR1 Overwrite Policy

Class 5: PERR (the highest priority)

Class 4: UCU, UCC, TUE_SH, TUE, L3_TUE_SH, L3_TUE, L3_UCU, L3_UCC

Class 3: UE, DUE, EDU, EMU, WDU, CPU, L3_EDU, L3_WDU, L3_CPU

Class 2: CE, EDC, EMC, WDC, CPC, THCE, L3_THCE, L3_EDC, L3_WDC, L3_CPC

Class 1: TO, DTO, BERR, DBERR (the lowest priority)

Class 5 errors are hardware time-outs associated with the EESR status bits CPQ_TO and NCPQ_TO. These are transactions that have exceeded the permitted time, unlike AFSR.TO events which are just transactions for which the system bus did not assert MAPPED. These are all fatal errors. AFSR.PERR events other than these three do not capture AFAR1.

Priority for AFAR1 updates: (PERR) > (UCU, UCC, TUE_SH, TUE, L3_TUE_SH, L3_TUE, L3_UCU, L3_UCC) > (UE, DUE, EDU, EMU, WDU, CPU, L3_EDU, L3_WDU, L3_CPU) > (CE, EDC, EMC,WDC, CPC, THCE, L3_THCE, L3_EDC, L3_WDC, L3_CPC) > (TO, DTO, BERR, DBERR)

There is one exception to the above AFAR1 overwrite policy. Within the same priority class, it is possible that multiple errors from system bus, L2-cache tag, L2-cache data, L3-cache tag, and L3-cache data might be reported at the same clock cycle. For this case, all the errors will be logged into AFSR1/AFSR1_EXT, and the priority for AFAR1 is (system bus error) > (L3-cache data error) > (L3-cache tag error) > (L2-cache data error, L2-cache tag error). Note that when L2-cache tag correctable error occurs, it will retry the request except snoop request, and it will not report any L2-cache data errors. If L2-cache tag uncorrectable error occurs, the error pin is asserted. In this case, it will not report any L2-data error, since it does not know which way to access. It is not possible for software to differentiate this event from the same errors arriving on different clock cycles, but the probability of having simultaneous errors is extremely low. This difficulty only applies to AFAR1 and AFAR2. AFSR1/AFSR1_EXT and AFSR2/AFSR2_EXT fields do not suffer this confusion on simultaneously arriving errors, and the normal overwrite priorities apply there.

The policy of flushing the entire D-cache on a deferred *data_access_error* trap or a precise *fast_ECC_error* trap avoids problems with the AFAR showing an inappropriate address when

1. Multiple errors occur.

2. Simultaneous L2-cache/L3-cache and system bus errors occur.
3. An L2-cache or L3-cache error is captured in AFSR and AFAR, yet no trap is generated because the event was a speculative instruction later discarded. Later, a trap finds the old AFAR.
4. A UE was detected in the first half on a 64 byte block from system bus, but the second half of the 64-byte block, also in error, was loaded into the D-cache.

7.5.2 AFSR1.E_SYND Data ECC Syndrome Overwrite Policy

Class 3: UCU, UCC, L3_UCU, L3_UCC (the highest priority)

Class 2: UE, DUE, IVU, EDU, WDU, CPU, L3_EDU, L3_WDU, L3_CPU

Class 1: CE, IVC, EDC, WDC, CPC, L3_EDC, L3_WDC, L3_CPC (the lowest priority)

Priority for E_SYND updates: (UCU, UCC, L3_UCU, L3_UCC) > (UE, DUE, IVU, EDU, WDU, CPU, L3_EDU, L3_WDU, L3_CPU) > (CE, IVC, EDC, WDC, CPC, L3_EDC, L3_WDC, L3_CPC)

7.5.3 AFSR1.M_SYND microtag ECC Syndrome Overwrite Policy

Class 2: EMU, IMU (the highest priority)

Class 1: EMC, IMC (the lowest priority)

Priority for M_SYND updates: (EMU, IMU) > (EMC, IMC)

7.6 Multiple Errors and Nested Traps

The AFSR1.ME bit is set when there are multiple uncorrectable errors or multiple SW_correctable errors associated with the same sticky bit in different data transfers.

Multiple occurrences of all uncorrectable errors (ISAP, EMU, IVU, TO, DTO, BERR, DBERR, UCU, TUE_SH, TUE, CPU, WDU, EDU, DUE, UE, L3_UCU, L3_UCC, L3_EDU, L3_WDU, L3_CPU, L3_TUE_SH, L3_TUE or L3_MECC errors) will set the AFSR1.ME bit. For example, one ISAP error and one EMU error will not set the ME bit, but two ISAP errors will.

Multiple occurrences of SW_correctable errors that set AFSR1.ME include UCC and L3_UCC errors only. This is to make diagnosis easier for the unrecoverable event of an L2-cache/L3-cache error while handling a previous L2-cache/L3-cache error.

If multiple errors leading to the same trap type are reported before a trap is taken due to any one of them, then only one trap will be taken for all those errors.

If multiple errors leading to different trap types are reported before a trap is taken for any one of them, then one trap of each type will be taken. One *instruction_access_error* and one *data_access_error,* and so on.

Multiple errors occurring in separate correction words of a single transaction, an L2-cache read or L3-cache read or a system bus read, do not set the AFSR1.ME bit. AFSR2.ME is never set.

7.7 Further Details on Detected Errors

This section includes a more extensive description for detailed diagnosis.

A simplified block diagram of the on-chip caches and the external interfaces is provided here to illustrate the main data paths and the terminologies used below for logging the different kind of errors. TABLE 7-16, is the main reference for all aspects of each individual error. The descriptive paragraphs in the following sections are meant to clarify the key concepts.

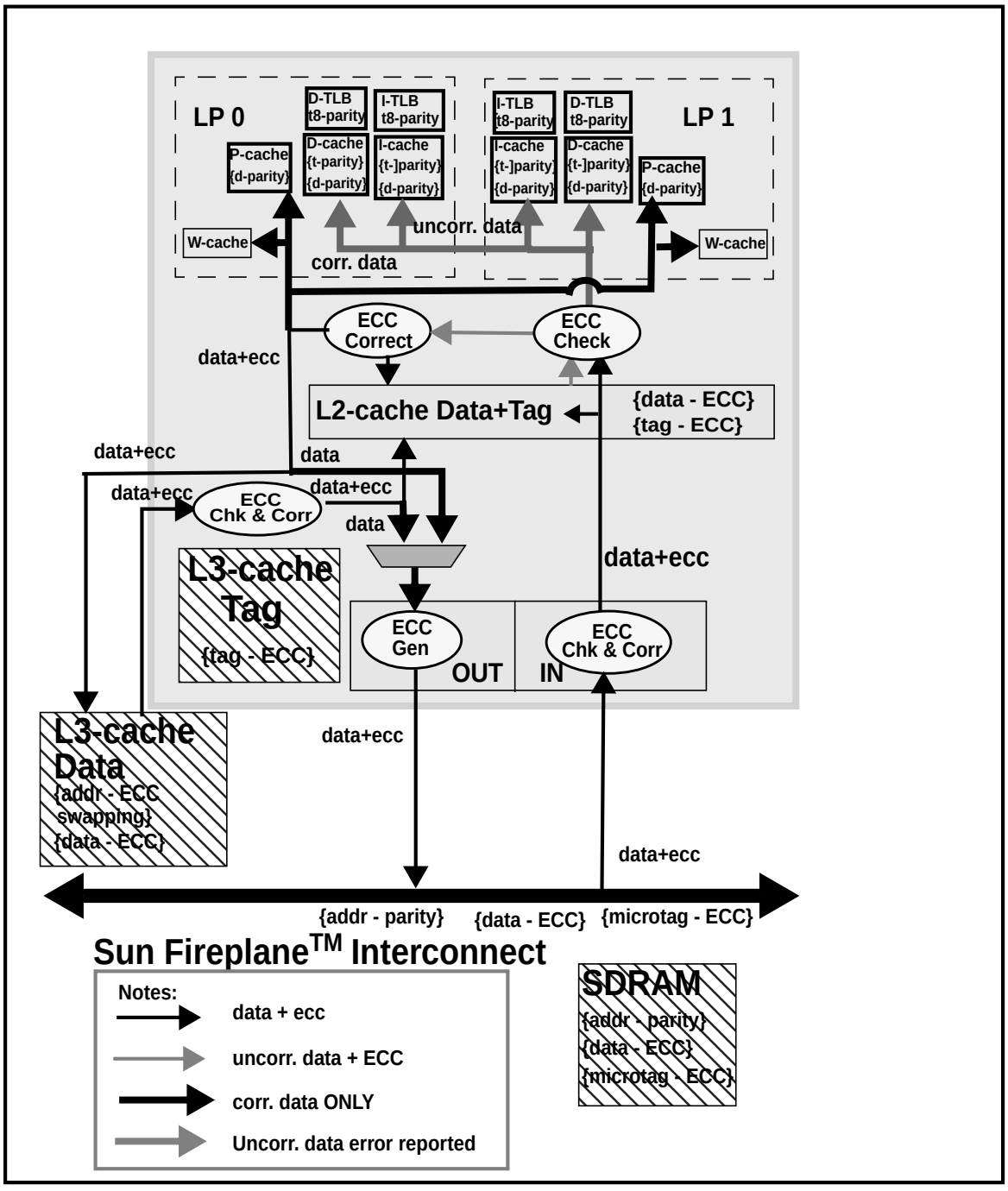

FIGURE 7-1 The UltraSPARC IV+ Processor RAS Diagram

7.7.1 L2-cache Data ECC Error

UCC:

When an instruction fetch misses the I-cache, a load-like instruction misses the D-cache, or an atomic operation is performed, and it hits the L2-cache, data is read from the L2-cache SRAM and will be checked for the correctness of its ECC. If a single bit error is detected in critical 32-byte data for load and atomic operations or in either critical or non-critical 32-byte data for I-cache fetch, the UCC bit will be set to log this error condition. This is a SW_correctable error. A precise *fast_ECC_error* trap will be generated provided that the UCEEN bit of the Error Enable Register is set. For correctness, a software-initiated flush of the D-cache is required, because the faulty word will already have been loaded into the D-cache, and will be used if the trap routine retries the faulting instruction.

L2-cache errors are not loaded into the I-cache or P-cache, so there is no need to flush them.

A software-initiated L2-cache flush, which evicts the corrected line into L3-cache, is desirable so that the corrected data can be brought back from L3-cache later. Without the L2-cache flush, a further single-bit error is likely the next time this word is fetched from L2-cache.

Multiple occurrences of this error will cause the AFSR1.ME to be set.

In the event that the UCC event is for an instruction fetch which is later discarded without the instruction being executed, no trap will be generated.

UCU:

When a cacheable load instruction misses the I-cache or D-cache, or an atomic operation misses the D-cache, and it hits the L2-cache, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a multi-bit error is detected in critical 32-byte data for load and atomic operations or in either critical or non-critical 32-byte data for I-cache fetch, it will be recognized as an uncorrectable error and the UCU bit will be set to log this error condition. A precise *fast_ECC_error* trap will be generated provided that the UCEEN bit of the Error Enable Register is set.

For correctness, a software-initiated flush of the D-cache is required, because the faulty word may already have been loaded into the D-cache, and will be used without any error trap if the trap routine retries the faulting instruction.

Corrupt data is never stored in the I-cache or P-cache.

A software-initiated flush of the L2-cache, which evicts the corrupted line into L3-cache, then a software-initiated flush of the L3-cache, which evicts the corrupted line from L3-cache back to DRAM, is required if this event is not to recur the next time the word is fetched from L2-cache. This may need to be linked with a correction of a multi-bit error in L2-cache if that is corrupted too.

Multiple occurrences of this error will cause the AFSR1.ME to be set.

In the event that the UCU event is for an instruction fetch which is later discarded without the instruction being executed, no trap will be generated.

EDC:

The AFSR.EDC status bit is set by errors in block loads to the processor, and errors in reading L2-cache as the result of store queue exclusive request, and errors as the result of prefetch queue operations.

When a block-load instruction misses the D-cache and hits the L2-cache, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a single bit error is detected, the EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware will proceed to correct the error.

A software PREFETCH instruction writes a command to the prefetch queue which operates autonomously from the execution unit. A correctable L2-cache data ECC error as the result of a read operation initiated by a prefetch queue entry will set EDC and a disrupting *ECC_error* trap will be generated. No data will be installed in the P-cache.

When a store instruction misses the W-cache and hits the L2-cache in E or M state, the store queue will issue an exclusive request to the L2-cache. The exclusive request will perform an L2-cache read, and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a single bit error is detected, the EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware will proceed to correct the error.

Note – The UltraSPARC IV+ processor's W-cache has been improved to contain entire modified line data. When the modified line is evicted from W-cache to L2-cache, it is written into L2-cache directly, without the sequence of reading, merging, and scrubbing actions in the UltraSPARC III processor. Therefore, W-cache eviction will not generate any EDC or EDU error in the UltraSPARC IV+ processor.

When an atomic instruction misses from the W-cache and hits the L2-cache in the E or M state, a store queue will issue an atomic exclusive request to L2-cache. The exclusive request will perform an L2-cache read, and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a single bit error is detected in a non-critical 32-byte, the EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware will proceed to correct the error.

EDU:

The AFSR.EDU status bit is set by errors in block loads to the processor, and errors in reading the L2-cache for store operations, and prefetch queue operations.

When a block load misses the D-cache and hits the L2-cache, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the EDU bit will be set to log this error condition. A deferred *data_access_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register.

A software PREFETCH instruction writes a command to the prefetch queue which operates autonomously from the execution unit. An uncorrectable L2-cache data ECC error as the result of a read operation initiated by a prefetch queue entry will set EDU. No data will be stored in the P-cache.

When a store instruction misses the W-cache and hits the L2-cache in the E or M state, a store queue will issue an exclusive request to the L2-cache. The exclusive request will perform an L2-cache read, and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If uncorrectable ECC error is detected, the EDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register. When the L2-cache data is delivered to the W-cache, the associated UE information will be stored with data. When W-cache evicts a line with UE data information, it will generate ECC based on the data stored in W-cache, then ECC check bits C[1:0] of the 128 bit word are inverted before the word is scrubbed back to the L2-cache.

When an atomic instruction misses from the W-cache and hits the L2-cache in E or M state, a store queue will issue an exclusive request to the L2-cache. The exclusive request will perform an L2-cache read, and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If uncorrectable ECC error is detected in non-critical 32-byte data, the EDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register. When the L2-cache data is delivered to the W-cache, the associated UE information will be stored with data. When W-cache evicts a line with UE data information, it will generate ECC based on the data stored in the W-cache, then ECC check bits C[1:0] of the 128 bit word are inverted before the word is scrubbed back to the L2-cache.

If an L2-cache uncorrectable error is detected as the result of either a store queue exclusive request, or an atomic request, or a block load or a prefetch queue operation, and the AFSR.EDU status bit is already set, AFSR1.ME will be set.

WDC:

For an L2-cache writeback operation, when a line which is in either clean or dirty state in the L2-cache is being victimized to make way for a new line, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. The data read back from L2-cache will be put in the writeback buffer as the staging area for the writeback operation. If a single bit error is detected, the WDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the CEEN bit is set in the Error Enable Register. Hardware will proceed to correct the error. Corrected data will be written out to the L3-cache.

WDU:

For an L2-cache writeback operation, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. The data read back from L2-cache will be put in the L2-cache writeback buffer as the staging area for the writeback operation. When a multi-bit error is detected, it will be recognized as an uncorrectable error and the WDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case, provided that the NCEEN bit is set in the Error Enable Register.

The uncorrectable L2-cache writeback data will be written into L3-cache. Therefore, the trap handler should perform a displacement flush to flush out the line in the L3-cache.

Multiple occurrences of this error will cause ASFR1.ME to be set.

CPC:

For a copyout operation to serve a snoop request from another processor, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a single bit error is detected, the CPC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the CEEN bit is set in the Error Enable Register. Hardware will proceed to correct the error and corrected data will be sent to the snooping device.

This bit is not set if the copyout happens to hit in the L2-cache writeback buffer because the line is being victimized. Instead, the WDC bit is set. Please refer to the section ‘WDC” for an explanation of this.

CPU:

For a copyout operation, an L2-cache read will be performed and the data read back from the L2-cache SRAM will be checked for the correctness of its ECC. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the CPU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case, provided that the NCEEN bit is set in the Error Enable Register.

Multiple occurrences of this error will cause AFSR1.ME to be set.

When the processor reads uncorrectable L2-cache data and writes it to the system bus in a copyback operation, it will compute correct system bus ECC for the corrupt data, then invert bits [127:126] of the data to signal to other devices that the data is not usable.

This bit is not set if the copyout hits in the writeback buffer. Instead, the WDU bit is set. Please refer to the section “WDU:” for an explanation of this.

The copyback data with UE information in W-cache, C[1:0] will be inverted in the data written back to the L2-cache, and D[127:126] will be inverted in the data written to the system bus.

7.7.2 L2-cache Tag ECC Errors

THCE:

For an instruction fetch, a load, or atomic operation, or writeback, copyout, store queue exclusive request, block store and prefetch queue operations, L2-cache data fill, processor hardware corrects single bit errors in L2-cache tags without software intervention, then retry the operation. For a snoop read operation, the processor will return L2-cache snoop result based on the corrected L2-cache tag and correct L2-cache tag at the same time. For these events, AFSR.THCE is set, and a disrupting *ECC_error* trap is generated.

For all hardware corrections of L2-cache tag ECC errors, not only does the processor hardware automatically correct the erroneous tag, but it also writes the corrected tag back to the L2-cache tag RAM.

Diagnosis software can use correctable error rate discrimination to determine if a real fault is present in the L2-cache tags, rather than a succession of soft errors.

TUE:

For any access except ASI access, and SIU tag update or copyback from foreign bus transaction, and snoop request, if an uncorrectable tag error is discovered, the processor sets AFSR.TUE. The processor asserts its ERROR output pin and it is expected that the coherence domain has suffered a fatal error and must be restarted.

TUE_SH:

For any access due to SIU tag update or copyback from foreign bus transaction, and snoop request, if an uncorrectable tag error is discovered, the processor sets AFSR.TUE_SH. The processor asserts its ERROR output pin and it is expected that the coherence domain has suffered a fatal error and must be restarted.

7.7.3 L3-cache Data ECC Errors

L3_UCC:

When an instruction fetch misses the I-cache, or a load instruction misses the D-cache, or an atomic operation is performed, and it hits the L3-cache, the line is moved from the L3-cache to L2-cache, and will be checked for the correctness of its ECC. If a single bit error is detected in critical 32-byte data for load and atomic operations or in either critical or non-critical 32-byte data for I-cache fetch, the L3_UCC bit will be set to log this error condition. This is a SW_correctable error. A precise *fast_ECC_error* trap will be generated provided that the UCEEN bit of the Error Enable Register is set.

For correctness, a software-initiated flush of the D-cache is required, because the faulty word will already have been loaded into the D-cache, and will be used if the trap routine retries the faulting instruction.

L3-cache errors are not loaded into the I-cache or P-cache, so there is no need to flush them.

Note that when the line moved from the L3-cache to the L2-cache, raw data read from the L3-cache without correction is stored in the L2-cache. Since the L2-cache and the L3-cache are mutually exclusive, once the line is read from the L3-cache to the L2-cache, it will not exist in the L3-cache. A software-initiated the L2-cache flush which will flush the data back to the L3-cache is desirable so that the corrected data can be brought back from the L3-cache later. Without the L2-cache flush, a further single-bit error is likely the next time this word is fetched from the L2-cache.

Multiple occurrences of this error will cause the AFSR1.ME to be set.

In the event that the L3_UCC event is for an instruction fetch which is later discarded without the instruction being executed, no trap will be generated.

If both the L3_UCC and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_UCU:

When a cacheable load instruction misses the I-cache or D-cache, or an atomic operation misses the D-cache, and it hits the L3-cache, the line is moved from the L3-cache to the L2-cache, and will be checked for the correctness of its ECC. If a multi-bit error is detected in critical 32-byte

data for load and atomic operations or in either critical or non-critical 32-byte data for I-cache fetch, it will be recognized as an uncorrectable error and the UCU bit will be set to log this error condition. A precise *fast_ECC_error* trap will be generated provided that the UCEEN bit of the Error Enable Register is set.

For correctness, a software-initiated flush of the D-cache is required, because the faulty word may already have been loaded into the D-cache, and will be used without any error trap if the trap routine retries the faulting instruction.

Note that when the line moved from the L3-cache to the L2-cache, raw data read from the L3-cache without correction is stored in the L2-cache. Since the L2-cache and the L3-cache are mutual exclusive, once the line is read from the L3-cache to the L2-cache, it will not exist in the L3-cache. Software-initiated flushes of the L2-cache and L3-cache are required if this event is not to recur the next time the word is fetched from the L2-cache. This may need to be linked with a correction of a multi-bit error in the L3-cache if that is corrupted too.

Multiple occurrences of this error will cause the AFSR1.ME to be set.

In the event that the L3_UCU event is for an instruction fetch which is later discarded without the instruction being executed, no trap will be generated.

If both the L3_UCU and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_EDC:

The AFSR.L3_EDC status bit is set by errors in block loads to the processor, and errors in reading L3-cache as the result of store queue exclusive request, and errors as the result of prefetch queue operations.

When a block load instruction misses the D-cache and hits the L3-cache, the line is moved from the L3-cache to the L2-cache, and will be checked for the correctness of its ECC. If a single bit error is detected, the L3_EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware deliver the corrected data to P-cache block-load data buffer.

When a store instruction misses the W-cache and hits the L3-cache in E, S, O, Os, or M state, a store queue will issue an exclusive request. The exclusive request will perform an L3-cache read, and the data read back from the L2-cache will be checked for the correctness of its ECC. If a single bit error is detected, the L3_EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware writes the corrected data to the W-cache.

When an atomic operation misses from the W-cache and hits the L3-cache in E, S, O, Os, or M state, a store queue will issue an exclusive request. The exclusive request will perform an L3-cache read, and the data read back from the L2-cache will be checked for the correctness of its ECC. If a single bit error is detected in non-critical 32-byte data, the L3_EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case while hardware writes the corrected data to W-cache.

When a software PREFETCH instruction misses the P-cache and hits the L3-cache, the line is moved from the L3-cache to the L2-cache, and is checked for the correctness of its ECC. If a single bit error is detected, the L3_EDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case. No data will be installed in the P-cache.

Note that when the line moved from L3-cache to L2-cache, raw data read from L3-cache without correction is stored in L2-cache. Since L2-cache and L3-cache are mutually exclusive, once the line is read from L3-cache to L2-cache, it will not exist in L3-cache. A software-initiated L2-cache flush which will flush the data back to L3-cache is desirable so that the corrected data can be brought back from L3-cache later. Without the L2-cache flush, a further single-bit error is likely the next time this word is fetched from L2-cache.

If both the L3_EDC and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_EDU:

The AFSR.EDU status bit is set by errors in block loads to the processor, and errors in reading L3-cache to merge data to complete store-like operations, and prefetch queue operations.

When a block-load instruction misses the D-cache and hits the L3-cache, the line is moved from the L3-cache to L2-cache, and will be checked for the correctness of its ECC. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the L3_EDU bit will be set to log this error condition. A deferred *data_access_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register.

When a store instruction misses the W-cache and hits L3-cache in E, S, O, Os, or M state, a store queue will issue an exclusive request to L3-cache. The exclusive request will perform an L3-cache read, and the line is moved from the L3-cache to L2-cache, and will be checked for the correctness of its ECC. If uncorrectable ECC error is detected, the L3_EDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register. When L3-cache data is delivered to the W-cache, the associated UE information will be stored with data. When the W-cache evicts a line with the UE data information, it will generate ECC based on the data stored in the W-cache, then ECC check bits C[1:0] of the 128-bit word are inverted before the word is scrubbed back to the L2-cache.

When an atomic operation misses from the W-cache and hits the L3-cache in E/S/O/Os/M state, a store queue will issue an exclusive request to the L3-cache. The exclusive request will perform an L3-cache read, and the line is moved from the L3-cache to the L2-cache, and will be checked for the correctness of its ECC. If uncorrectable ECC error is detected in non-critical 32-byte data, the L3_EDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the NCEEN bit is set in the Error Enable Register. When the L3-cache data is delivered to the W-cache, the associated UE information will be stored with data. When the W-cache evicts a line with UE data information, it will generate ECC based on the data stored in the W-cache, then ECC check bits C[1:0] of the 128 bit word are inverted before the word is scrubbed back to the L2-cache.

When a software PREFETCH instruction misses the P-cache and hits the L3-cache, the line is moved from the L3-cache to the L2-cache, and will be checked for the correctness of its ECC. If an uncorrectable ECC error is detected, the L3_EDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case. No data will be installed in the P-cache.

Note that when the line moved from the L3-cache to the L2-cache, raw data read from the L3-cache without correction is stored in the L2-cache. Since the L2-cache and the L3-cache are mutually exclusive, once the line is read from the L3-cache to the L2-cache, it will not exist in the L3-cache. Software-initiated flushes of the L2-cache and L3-cache are required if this event is not to recur the next time the word is fetched from the L2-cache. This may need to be linked with a correction of a multi-bit error in the L3-cache if that is corrupted too.

If an L3-cache uncorrectable error is detected as the result of either a store queue exclusive request or a block load or a prefetch queue operation, and the AFSR.EDU status bit is already set, AFSR1.ME will be set.

If both the L3_EDU and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_WDC:

For an L3-cache writeback operation, when a modified line in the L3-cache is being victimized to make way for a new line, an L3-cache read will be performed and the data read back from the L3-cache will be checked for the correctness of its ECC. The data read back from L3-cache will be put in the writeback buffer as the staging area for the writeback operation. If a single bit error is detected, the L3_WDC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case providing that the CEEN bit is set in the Error Enable Register. Hardware will proceed to correct the error. Corrected data will be written out to memory through the system bus.

If both the L3_WDC and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_WDU:

For an L3-cache writeback operation, an L3-cache read will be performed and the data read back from the L3-cache will be checked for the correctness of its ECC. The data read back from L3-cache will be put in the writeback buffer as the staging area for the writeback operation. When a multi-bit error is detected, it will be recognized as an uncorrectable error and the L3_WDU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case, provided that the NCEEN bit is set in the Error Enable Register.

When the processor reads uncorrectable L3-cache data and writes it to the system bus in a writeback operation, it will compute correct system bus ECC for the corrupt data, then invert bits [127:126] of the data to signal to other devices that the data is not usable.

Multiple occurrences of this error will cause ASFR1.ME to be set.

If both the L3_WDU and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_CPC:

For an L3-cache copyout operation to serve a snoop request from another processor, an L3-cache read will be performed and the data read back from the L3-cache will be checked for the correctness of its ECC. If a single bit error is detected, the L3_CPC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case provided that the CEEN bit is set in the Error Enable Register. Hardware will proceed to correct the error and corrected data will be sent to the snooping device.

This bit is not set if the copyout happens to hit in the writeback buffer because the line is being victimized. Instead, the WDC bit is set. Please refer to the section “L3_WDC:” for an explanation of this.

If both the L3_CPC and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_CPU:

For an L3-cache copyout operation, an L3-cache read will be performed and the data read back from the L3-cache will be checked for the correctness of its ECC. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the L3_CPU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case, provided that the NCEEN bit is set in the Error Enable Register.

Multiple occurrences of this error will cause AFSR1.ME to be set.

When the processor reads uncorrectable L3-cache data and writes it to the system bus in a copyback operation, it will compute correct system bus ECC for the corrupt data, then invert bits [127:126] of the data to signal to other devices that the data is not usable.

This bit is not set if the copyout hits in the writeback buffer. Instead, the WDU bit is set. Please refer to the section “L3_WDU:” for an explanation of this.

If both the L3_CPU and the L3_MECC bits are set, it indicates that an address parity error has occurred.

L3_MECC:

L3-cache data access errors on both 16-byte data of L3-cache data access.

7.7.4 L3-cache Tag ECC Errors

L3_THCE:

For an instruction fetch, a load, atomic operation, writeback, copyout, store queue exclusive request, block store and prefetch queue operations, and L3-cache data fill, processor hardware corrects single bit errors in the L3-cache tags without software intervention, then retries the operation. For a snoop read operation, the processor will return L3-cache snoop result based on the corrected L3-cache tag and correct L3-cache tag at the same time. For these events, AFSR.L3_THCE is set, and a disrupting *ECC_error* trap is generated.

For all hardware corrections of L3-cache tag ECC errors, not only does the processor hardware automatically correct the erroneous tag, but it also writes the corrected tag back to the L3-cache tag RAM.

Diagnosis software can use correctable error rate discrimination to determine if a real fault is present in the L3-cache tags, rather than a succession of soft errors.

L3_TUE:

For any access except ASI access, and SIU tag update or copyback from foreign bus transaction, and snoop request, if an uncorrectable L3-cache tag error is discovered, the processor sets AFSR.TUE. The processor asserts its ERROR output pin and it is expected that the coherence domain has suffered a fatal error and must be restarted.

L3_TUE_SH:

For any access for SIU tag update or copyback from foreign bus transaction, and snoop request, if an uncorrectable L3-cache tag error is discovered, the processor sets AFSR.TUE. The processor asserts its ERROR output pin and it is expected that the coherence domain has suffered a fatal error and must be restarted.

7.7.5 System Bus ECC Errors

CE:

When data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for the correctness of its ECC. If a single bit error is detected, the CE bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated in this case if the CEEN bit is set in the Error Enable Register. Hardware will correct the error.

UE:

When data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for the correctness of its ECC. For load-like, block load or atomic operations, if a multi-bit error is detected, it will be recognized as an uncorrectable error and the UE bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a deferred *instruction_access_error* or *data_access_error* trap will be generated, depending on whether the read was to satisfy an instruction fetch or a load operation.

Multiple occurrences of this error will cause AFSR1.ME to be set.

DUE:

When data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for the correctness of its ECC. For prefetch queue and store queue operations, if a multi-bit error is detected, it will be recognized as an uncorrectable error and the DUE bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a disrupting *ECC_error* trap will be generated.

Multiple occurrences of this error will cause AFSR1.ME to be set.

EMC:

When data are entering the UltraSPARC IV+ processor from the system bus, the microtags will be checked for the correctness of ECC. If a single bit error is detected, the EMC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated, provided that the CEEN bit is set in the Error Enable Register. Hardware will correct the error.

EMU:

When data are entering the UltraSPARC IV+ processor from the system bus, the microtags will be checked for the correctness of ECC. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the EMU bit will be set to log this error condition. Provided that the

NCEEN bit is set in the Error Enable Register, a deferred *instruction_access_error* or *data_access_error* trap will be generated, depending on whether the read was to satisfy an instruction fetch or a load operation.

Multiple occurrences of this error will cause the AFSR.ME to be set.

IVC:

When interrupt vector data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for ECC correctness. If a single bit error is detected, the IVC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated, provided that the CEEN bit is set in the Error Enable Register. Hardware will correct the error.

IVU:

When interrupt vector data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for ECC correctness. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the IVU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated, provided that the NCEEN bit is set in the Error Enable Register.

Multiple occurrences of this error will cause AFSR.ME to be set.

A multi-bit error in received interrupt vector data still causes the data to be stored in the interrupt receive registers, but does not cause an *interrupt_vector* disrupting trap.

IMC:

When interrupt vector data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for microtags correctness. If a single bit error is detected, the IMC bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated, provided that the CEEN bit is set in the Error Enable Register. Hardware will correct the error.

IMU:

When interrupt vector data are entering the UltraSPARC IV+ processor from the system bus, the data will be checked for microtags correctness. If a multi-bit error is detected, it will be recognized as an uncorrectable error and the IMU bit will be set to log this error condition. A disrupting *ECC_error* trap will be generated, provided that the NCEEN bit is set in the Error Enable Register.

Multiple occurrences of this error will cause AFSR.ME to be set.

A multi-bit error in received interrupt vector data still causes the data to be stored in the interrupt receive registers, but does not cause an *interrupt_vector* disrupting trap.

7.7.6 System Bus Status Errors

BERR:

When the UltraSPARC IV+ processor performs a system bus read access, DSTAT = 2 or 3 status may be returned. The processor treats both Sun Fireplane Interconnect termination code DSTAT = 2, “time-out error”, and DSTAT = 3, “bus error”, as the same event. For a bus error due to a instruction fetch, load-like, block load, or atomic operation, the BERR bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a deferred *instruction_access_error* or *data_access_error* trap will be generated, depending on whether the read was to satisfy an instruction fetch or a load operation.

Multiple occurrences of this error will cause AFSR1.ME to be set.

DBERR:

When the UltraSPARC IV+ processor performs a system bus read access, DSTAT = 2 or 3 status may be returned. The processor treats both Sun Fireplane Interconnect termination code DSTAT = 2, “time-out error”, and DSTAT = 3, “bus error”, as the same event. For a bus error due to a system bus read from memory or I/O caused by prefetch queue, or a system bus read from memory caused by read-to-own store queue operation, the DBERR bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a deferred *data_access_error* trap will be generated.

Multiple occurrences of this error will cause AFSR1.ME to be set.

TO:

When the UltraSPARC IV+ processor performs a system bus read or write access, it is possible that no device responds with a MAPPED status. This is not a hardware time-out operation, which causes an AFSR.PERR event. It is also different from a DSTAT = 2 “time-out” response for a Sun Fireplane Interconnect transaction, which causes an AFSR.BERR or AFSR.DBERR. For an unmapped bus error due to a system bus read for an instruction fetch, load-like, block load, or atomic operation, or a system bus write for block store to memory (WS), store to I/O (WIO), block store to I/O (WBIO), writeback from L3-cache, or interrupt vector transmit operation, the TO bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a deferred *instruction_access_error* or *data_access_error* trap will be generated, depending on whether the read was to satisfy an instruction fetch or a load operation.

Multiple occurrences of this error will cause AFSR1.ME to be set.

DTO:

When the UltraSPARC IV+ processor performs a system bus read access, it is possible that no device responds with a MAPPED status. This is not a hardware time-out operation, which causes an AFSR.PERR event. It is also different from a DSTAT = 2 “time-out” response for a Sun Fireplane Interconnect transaction, which causes an AFSR.BERR or AFSR.DBERR. For an unmapped bus error due to a system bus read for a prefetch queue or read-to-own store queue operation, the DTO bit will be set to log this error condition. Provided that the NCEEN bit is set in the Error Enable Register, a deferred *data_access_error* trap will be generated.

Multiple occurrences of this error will cause AFSR1.ME to be set.

Note – For foreign WIO to the UltraSPARC IV+ processor internal ASI registers such as MCU parameters, both CE error or UE error in the data will not be logged. For CE error in the data, it will be corrected automatically. For UE error in the data, bad data will be installed in the ASI registers and no traps will be generated.

7.7.7 SRAM e-Fuse Array Related Errors

EFA_PAR_ERR:

When parity error occurs during transfer from e-Fuse array to repairable SRAM array, AFSR.EFA_PAR_ERR is set to 1, and error pin is asserted. To clear AFSR.EFA_PAR_ERR, user must pull hard power-on reset which will initiate e-Fuse array transfer, and clear the parity error.

RED_ERR:

When e-Fuse error occurs in I-cache, D-Cache, DTLBs, or I-TLB SRAM redundancy, or shared L2-cache/L3-cache tag/L2-cache data SRAM redundancy, AFSR.RED_ERR is set to 1, and error pin is asserted. To clear AFSR.RED_ERR, user must pull hard power-on reset which will initiate e-Fuse array transfer, and clear the parity error.

7.8 Further Details of ECC Error Processing

7.8.1 System Bus ECC Errors

7.8.1.1 ECC Error Detection

For incoming data from the system bus, ECC error checking is turned on when the Sun Fireplane Interconnect DSTAT bits indicate that the data is valid and ECC has been generated for this data. ECC is not checked for system bus data which returns with DSTAT = 1, 2 or 3.

ECC errors may occur in either the data or microtag field. The UltraSPARC IV+ processor can store only one data ECC syndrome and one microtag ECC syndrome for every 64 bytes of incoming data even though it does detect errors in every 16 bytes of data.

The syndrome of the first ECC error detected, whether HW_corrected or uncorrectable, is saved in an internal error register.

If the first occurrence of an ECC error is uncorrectable, the error register is locked and all subsequent errors within the 64 byte block are ignored.

If the first occurrence of an ECC error is HW_corrected, then subsequent correctable errors within the 64-byte block will be corrected but not logged. A subsequent uncorrectable error will overwrite the syndrome of the correctable error. At this point, the error register is locked.

7.8.1.2 Signalling ECC Error

Not only does the UltraSPARC IV+ processor perform ECC checking for incoming system bus data, but it also generates ECC check bits for outgoing system bus data. A problem occurs when new ECC check bits are generated for data that contains an uncorrectable error (ECC or bus error). With new ECC check bits, there is no way to detect that the original data was bad.

To fix this problem without having to add an additional interface, a new uncorrectable ECC error is injected into the data after the new ECC check bits have been generated. In this way, the receiver will detect the uncorrectable error when it performs its own ECC checking. This deliberately bad ECC is known as *signalling* ECC.

For DSTAT = 2 or 3 events coming from the system bus and being stored with deliberately bad signalling ECC in the L2-cache, an uncorrectable error is injected by inverting data bits [1:0] after correct ECC is generated for the corrupt data.

For a no MAPPED event coming from the system bus, the data and ECC values present on the system bus at the time that the unmapped error is detected are not stored in the L2-cache. Any result can be returned when the L2-cache line affected is read.

For UE and DUE events coming from the system bus, the data and ECC values present on the system bus are stored unchanged in the L2-cache. An uncorrectable error should be returned when the L2-cache line is read, but the syndrome is not defined.

For uncorrectable ECC errors detected in copyout data from L2-cache/L3-cache, or writeback data from the L3-cache, an uncorrectable error is injected into outgoing data by inverting data bits [127:126] after correct ECC is generated for the corrupt data.

For uncorrectable ECC errors detected in an L2-cache or L3-cache read to complete a store queue exclusive request associated with a store-like operation, ECC check bits [1:0] will be inverted in the data scrubbed back to the L2-cache when W-cache evict this line.

A line which arrives as DSTAT = 2 or 3 and is stored in the L2-cache with data bits [1:0] inverted can then be rewritten as part of a store queue operation with check bits [1:0] inverted and eventually written back out to the system bus with data bits [127:126] inverted.

The E_SYND reported for correction words with data bits [1:0] inverted is always 0x11c.

The E_SYND reported for correction words with data bits [127:126] inverted is always 0x071.

The E_SYND reported for correction words with check bits [1:0] inverted is always 0x003.

7.8.2 L2-cache and L3-cache Data ECC Errors

ECC error checking for L2-cache data is turned on for all read transactions from the L2-cache whenever the L2_data_ecc_en bit of the L2-cache Control Register is asserted.

ECC error checking for L3-cache data is turned on for all read transactions from the L3-cache whenever the EC_ECC_ENABLE bit of the L3-cache Control Register is asserted.

The UltraSPARC IV+ processor can store only one data ECC syndrome and one microtag ECC syndrome for every 32 bytes of L2-cache or L3-cache data even though it is possible to detect errors in every 16 bytes of data.

The syndrome of the first ECC error detected, whether HW_corrected, SW_correctable, or uncorrectable, is saved in an internal error register.

If the first occurrence of an ECC error is uncorrectable, the internal error register is locked and all subsequent errors within the 32 bytes are ignored.

If the first occurrence of an ECC error is correctable, then subsequent correctable errors within the 32 bytes are ignored. A subsequent uncorrectable error will overwrite the syndrome of the correctable error stored in the internal error register. At this point, the internal error register is locked. This applies to both HW_corrected and SW_correctable errors.

The internal error register is cleared and unlocked once the error has been logged in the AFSR.

7.8.3 When Are Traps Taken?

Precise traps such as *fast_instruction_access_mmu_miss* and *fast_ECC_error* are taken explicitly when the instruction with the problem is executed.

Disrupting and deferred traps are not associated with particular instructions. In fact, the processor only takes these traps when a valid instruction that will definitely be executed (not discarded as a result of speculation) is moving through particular internal pipelines.

TABLE 7-17 Traps and when they are taken

Type of Traps	Initiate trap processing when a valid instruction is in
instruction_access_error	**Note:** The instruction_access_error events, and many precise traps, produce instructions that cannot be executed. Therefore, the instruction fetcher takes the affected instruction and dispatches it to the BR pipe, specially marked to cause a trap. It is true to say that instruction_access_error traps are only taken when an instruction is in the BR pipe, but this has no effect, because the error creates an instruction in the BR pipe itself. So, instruction_access_error traps are taken as soon as the instruction fetcher dispatches the faulty instruction.
data_access_error	BR or MS pipe
interrupt_vector	BR, MS, A0 or A1 pipe (but see note above)
ECC_error	BR or MS pipe
interrupt_level_n	BR, MS, A0 or A1 pipe (but see note above)

The above table specifies when the processor will “initiate trap processing”, meaning consider what trap to take. When the processor has initiated trap processing, it will take one of the outstanding traps, but not necessarily the one which caused the trap processing to be initiated.

When the processor encounters an event which should lead to a trap, that trap type becomes *pending*. The processor continues to fetch and execute instructions until a valid instruction is issued to a pipe which is sensitive to a trap which is pending. During this time, further traps may become pending.

When the next committed instruction is issued to a pipe which is sensitive to any pending trap,

1. The processor ceases to issue instructions in the normal processing stream.
2. If the pending trap is a precise or disrupting trap, the processor waits for all the system bus reads which have already started, to complete. The processor does not do this if a deferred trap will be taken. During this waiting time, more traps can become pending.
3. The processor takes the highest priority pending trap. If a deferred trap, *data_access_error* or *instruction_access_error*, is pending, then the processor begins execution of either the *data_access_error* or *instruction_access_error* trap code. If both *data_access_error* and *instruction_access_error* traps are pending, the *instruction_access_error* trap will be taken, because it is higher priority. Taking a *data_access_error* trap clears the pending status for data access errors. Taking an *instruction_access_error* trap clears the pending status for instruction access errors, and has no effect on the pending status of data access errors. Because of priorities, there cannot be an *instruction_access_error* trap pending at the time a *data_access_error* trap is taken. Taking any trap makes all precise traps no longer pending.

The description above implies that a pipe has to have a valid instruction to initiate trap handling, but once trap handling is initiated, any of the pending traps can be taken, not just ones to which that pipe is sensitive. So, if the processor is executing A0 pipe instructions, and a *data_access_error* is pending but cannot be taken, an *interrupt_vector* can arrive, and enable the *data_access_error* trap to be executed even though only A0 pipe instructions are present.

If a *data_access_error* trap becomes pending but cannot be taken because neither the BR or MS pipe has a valid instruction, the processor continues to fetch and execute instructions. If an *instruction_access_error* trap then becomes pending, the offending instruction will be issued to the BR pipe to allow trap processing to be initiated. The processor then will examine both the pending traps, and take the *instruction_access_error* trap, say at TL = 1, because it is higher priority. The *data_access_error* remains pending. When the first BR or MS pipe instruction is executed in the *instruction_access_error* trap routine, the *data_access_error* trap routine will run, say at TL = 2.

Despite the fact that the *data_access_error* has lower priority than the *instruction_access_error* trap, the *data_access_error* trap routine runs at a higher TL, within an enclosing *instruction_access_error* trap routine, and before the bulk of that routine. This is the opposite of the usual effect that interrupt priorities have.

The result of this is that at the time that the trap handler begins, only one *data_access_error* trap is executed for all data access errors that have been detected by this time, and only one *instruction_access_error* trap is executed for all instruction access errors.

Processor action is always determined by the trap priorities, except for one special case, and that is for a precise *fast_ECC_error* trap pending at the same time as a deferred *data_access_error* or *instruction_access_error* trap. In this one case only, the higher priority deferred trap will be taken, and the precise trap will no longer be pending.

If a deferred trap is taken while a precise trap is pending, that precise trap will no longer be pending. So, a *data_access_error* trap routine might see multiple events logged in AFSR associated with both data and instruction access errors, and might also see an L2-cache or L3-cache error event logged. The L2-cache or L3-cache error event would normally be associated with a precise trap but the deferred trap happened to arrive at the same time and make the precise trap no longer pending. If the deferred trap routine executes RETRY (an unlikely event in itself) then the precise trap may become pending again, but this would depend on the L2-cache or L3-cache giving the same error again.

Pending disrupting traps are affected by the current state of PSTATE.IE and PSTATE.PIL. All the disrupting traps, *interrupt_vector*, *ECC_error* and *interrupt_level_[1-15]* are temporarily inhibited (i.e. their pending status is hidden) when PSTATE.IE is 0. *Interrupt_level_[1-15]* traps are also temporarily inhibited when PSTATE.PIL is greater than or equal to the interrupt level.

As an example, consider the following sequence of events.

1. An interrupt vector arrives with a HW_corrected system bus data ECC error. This makes *ECC_error* and *interrupt_vector* traps pending.

The processor continues to execute instructions looking for a BR, MS, A0 or A1 pipe instruction.

2. The processor executes a system bus read, the RTO associated with an earlier store-like instruction, and detects a DSTAT = 2 or 3 response. This makes a *ECC_error* trap pending.

The processor continues to execute instructions looking for a BR, MS, A0 or A1 pipe instruction.

3. The processor reads an instruction from the L2-cache and detects an L2-cache data ECC error. This makes a precise *fast_ECC_error* trap pending.
4. An earlier instruction prefetch from the system bus by the instruction fetcher (not a prefetch queue operation), of an instruction now known not to be used, completes. This instruction has a UE, which makes an *instruction_access_error* pending.

The instruction fetcher dispatches the corrupt instruction, specially marked, in the BR pipe. Because the processor can now take a trap, it inhibits further instruction execution and waits for outstanding system bus reads to complete. When the reads have completed, the processor then examines the various pending traps and begins to execute the deferred *instruction_access_error* trap, because deferred traps are handled before *fast_ECC_error* (as a special case), and that has the higher priority. This makes the *instruction_access_error* trap and all precise traps no longer pending.

The processor takes only one trap at a time. It will begin executing the *instruction_access_error* trap by fetching the exception vector and executing the instruction there. As part of SPARC V9 trap processing, the processor clears PSTATE.IE, so the *ECC_error* and *interrupt_vector* traps cannot be taken at the moment, so are no longer pending (although they’re still remembered, just hidden). The processor will now be running with TL = 1.

When the *instruction_access_error* trap routine executes a BR or MS pipe instruction, the *data_access_error* trap routine will run, at TL = 2. If that routine explicitly set PSTATE.IE, then the *interrupt_vector* and *ECC_error* traps would become pending again. After the next BR, MS, A0 or A1 pipe instruction, the processor, after waiting for outstanding reads to complete, would take the *interrupt_vector* trap, which would run at TL = 3.

However, assuming the *data_access_error* trap routine does not set PSTATE.IE, then it will run uninterrupted to completion at TL = 2. It’s a deferred trap, so it’s not possible to return to the TL = 1 routine correctly. Recovery at this time is a matter for the system designer.

7.8.4 When Are Interrupts Taken?

The processor is only sensitive to *interrupt_vector* and *interrupt_level_[1-15]* traps when a valid instruction is in the BR, MS, A0 or A1 pipes. If the processor is executing only FGA or FGM pipe instructions, it will not take the interrupt. This could lead to unacceptably long delays in interrupt processing.

To avoid this problem, if the processor is handling only floating point instructions, with PSTATE.PEF and FPRS.FEF both set, and an *interrupt_vector* or *interrupt_level_[1-15]* trap becomes pending, and PSTATE.IE is set, and (for *interrupt_level_[1-15]* traps) PSTATE.PIL is less than the pending interrupt level, the processor automatically disables its floating point unit by clearing PSTATE.PEF.

If the next instruction executed is an FGA or FGM pipe instruction, a precise *fp_disabled* trap will be taken. This has the side-effect of clearing PSTATE.IE, so the disrupting traps can still not be taken. However, the *fp_disabled* trap routine is specially arranged to first set PSTATE.PEF again, then set PSTATE.IE again, then execute a BR, MS, A0 or A1 pipe instruction. When this occurs, the *interrupt_vector* or *interrupt_level_[1-15]* trap routine is executed. Eventually the trap routine executes a `RETRY` instruction to retry the faulted floating point operation.

If the next instruction after the processor clears PSTATE.PEF is not an FGA or FGM pipe instruction, then it must be a BR, MS, A0 or A1 pipe instruction, and the disrupting trap routine can be executed anyway. In this case, the next floating point operation will trigger an *fp_disabled* trap, which results in the floating point unit being enabled again.

Automatic clearing of PSTATE.PEF is not triggered by pending disrupting *ECC_error* traps. The *ECC_error* trap routine can be indefinitely delayed if the processor is handling only FGA and FGM pipe instructions.

7.8.4.1 Error Barriers

A `MEMBAR #Sync` instruction causes the processor to wait until all system bus reads are complete and the store queue is empty before continuing. Stores will have completed any system bus activity (including any RTO for a cacheable store) but the store data may still be in the W-cache and may not have reached the L2-cache.

A `DONE` or `RETRY` instruction behaves exactly like a `MEMBAR #Sync` instruction for error isolation. The processor waits for outstanding data loads or stores to complete before continuing.

Traps do not serve as error barriers in the way that `MEMBAR #Sync` does.

User code can issue a store instruction which misses in the D-cache, L2-cache and L3-cache, therefore generating a system bus RTO operation. After the store instruction, the user code can go on to trap into privileged code, through an explicit `TRAP` instruction, a TLB miss, a spill or fill trap, or an arithmetic exception (such as a floating point trap). None of these trap events wait for the user code's pending stores to be issued, let alone to complete. The processor's store queue can hold several outstanding stores, any or all of which can require system bus activity. In the UltraSPARC IV+ processor, an uncorrectable system bus data ECC error as the result of a store queue or prefetch queue RTO leads only to a disrupting trap, not to a deferred trap. When the disrupting trap is taken is of no particular importance. These stores can issue and complete on the system bus after a trap has changed PSTATE.PRIV to 1, and errors as the result of the stores are *not* logged with AFSR.PRIV = 1, because they come from user code.

A DSTAT = 2 or 3 response to a prefetch queue or store queue read operation from user code can cause a disrupting trap with AFSR.PRIV = 1, after the user code has trapped into system space.

It happens that the detailed timing behavior of the processor prevents the same anomaly with load or atomic operations. Uncorrectable errors on these user operations will always present a deferred *data_access_error* trap with AFSR.PRIV = 0.

It is possible to use a MEMBAR #Sync instruction near the front of all trap routines, and to handle specially deferred traps that occur there, properly to isolate user-space hardware faults from system space faults. However, the cost in execution time for the error-free case is significant, and programmers may choose to decide that the additional small gain in possible system availability is not worth the cost in throughput. If there is no MEMBAR #Sync, the effect will be that a very small fraction of errors that might perhaps have terminated only one user process will, instead, result in a reboot of the affected coherence domain.

Neither traps nor explicit MEMBAR #Sync instructions provide a barrier for prefetch queue operations. Software PREFETCH instructions which are executed before a trap or MEMBAR #Sync can plant an operation in the prefetch queue. This operation can cause system bus activity after the trap or MEMBAR #Sync. In the UltraSPARC IV+ processor, an uncorrectable system bus data ECC error as the result of a prefetch queue operation results in a disrupting trap. It is not particularly important when this is taken.

One way to enforce an error barrier for a software PREFETCH for read(s) (prefetch fcn = 0, 1, 20, or 21) is to issue a floating-point load instruction that uses the prefetched data. This will force an interlock when the load is issued. The load miss request will not be sent, waiting for the completion of the prefetch request (with or without error). Upon completion, the load instruction is recirculated, any error will then be replayed and generate a precise trap. Note that in implementation, the floating-point load instruction has to be scheduled at least 4 cycles after the PREFETCH instruction.

Disrupting and precise traps do not act as though a MEMBAR #Sync was executed at the time of the trap. This is because the disrupting and precise traps wait for all reads that have been issued on the system bus to be complete, but not for the store queue to be empty, before beginning to execute the trap.

Store queue write operations (WS, WIO, or WBIO) can result in deferred traps, but only if the target device does not assert MAPPED (AFSR.TO). These are generally indicative of a fault more serious than a transient data upset. This can lead to the problem described in the next paragraph.

If several stores of type WS, WIO, or WBIO are present in the store queue, each store can potentially result in a *data_access_error* trap as a result of a system bus problem. Because deferred trap processing does not wait for all stores to complete, the *data_access_error* trap routine can start as soon as an error is detected as the result of the first store. (Execution of the trap routine still may be delayed until the right pipe includes a valid instruction, though). Once the *data_access_error* routine has started, a further store from the original store queue can result in system bus activity which eventually returns an error and causes another *data_access_error* trap to become pending. This can (once the correct pipe has a valid instruction in it) start another *data_access_error* trap routine, at TL = 2. This can continue until all available trap levels are exhausted and the processor begins RED_state execution.

To overcome this problem, we need to insert a MEMBAR #Sync or RETRY instruction at the beginning of the deferred trap handler. This avoids for nested deferred traps going to RED_state. The MEMBAR #Sync or RETRY requires the store queue to be empty before it can be issued. This forces the hardware to merge multiple deferred traps (while in TL = 1) into one deferred trap to stop at TL = 2.

In the case of other store types that fill STQ and generate system bus read operation (read-to-own), multiple disrupting traps might be generated back-to-back. In this case one disrupting trap is serviced at a time, no nested traps since the TL goes back to 0 before serving the next disrupting trap. This is the case because when a trap is taken, the hardware automatically sets the PSTATE.IE bit to 0 which disable further disrupting traps (and interrupts). Subsequent disrupting traps (and interrupts) are blocked, but not dropped.

7.9 IERR/PERR Error Handling

In the UltraSPARC IV+ processor, additional error detection and handling are added in the memory arrays within the logical processors and EMU (External Memory Unit) to improve the RAS features of the processor.

7.9.1 Error Detection and Reporting Structures

The errors detected in the memory arrays within each logical processor and the EMU are divided into three categories:

- Bus protocol errors: Error conditions that violate system bus protocol, and the UltraSPARC IV+ processor Coherency tables.
- Internal errors: Internal error conditions that point to inconsistent, or illegal operations on some of the finite state machines of the memory arrays within each logical processor and the EMU.
- Tag errors: Uncorrectable ECC errors on the L2-cache tag and L3-cache tag.

7.9.1.1 Asynchronous Fault Status Register

Bus protocol errors are reported by setting the PERR field of the AFSR. Internal errors are reported by setting the IERR bit in the AFSR. Tag errors are reported by setting the TUE bit, or TUE_SH bit, or L3_TUE bit, or L3_TUE_SH bit in the AFSR.

When one of the fatal error bits in the AFSR is set, the processor will assert its error pin for 8 consecutive system cycles. For information on AFSR, please refer to *AFSR Register and AFSR_EXT Register* on page 180.

The Asynchronous Fault Status Register and three other registers—Error Status, Error Mask, and report IERR/PERR errors.

7.9.2 Fatal Error (FERR)

It is usually impossible to recover a domain which suffers a system snoop request parity error, invalid coherence state, L2-cache tag uncorrectable error, L3-cache tag uncorrectable error, system interface protocol error, or internal error at the processor level. When these errors occur, the normal recovery mechanism is to reset the coherence domain of the effected processor. When one of these fatal errors is detected by a processor, the processor asserts its ERROR output pin. The response of the system when an ERROR pin is asserted depends on the system design.

Since the AFSR is not reset by a system reset event, error logging information is preserved. The system can generate a domain reset in response to assertion of an ERROR pin, and software can then examine the system registers to determine that the reset was due to an FERR. The AFSR of all processors can be read to determine the source and cause of the FERR.

Most errors which lead to FERR do not cause any special processor behavior. However, an uncorrectable error in the MTags, L2-cache tags, or L3-cache tag behaves differently than normal, causing the processor to both assert its ERROR output pin and to begin trap execution. Uncorrectable errors in the MTags, L2-cache tags or L3-cache tags are normally fatal and reset the affected coherence domain.

In the UltraSPARC IV+ processor, on-chip SRAMs are e-Fuse repairable. A parity error detected during the transfer of data between the e-Fuse array and the repairable SRAM array will cause the processor to assert the ERROR output pin. During normal operation, bit flipping in REDUNDANCY registers will cause the processor to assert the ERROR output pin.

After an FERR event, system reset to the processor can be cycled to regain control, and code can then read out the internal register values and run diagnostic tests. Following this diagnosis phase, if it is determined that the processor is to be integrated into a new domain, cycling POK (not necessarily power) initializes the processor as consistently as possible.

7.9.3 Entering RED_state

In the event of a catastrophic hardware fault which produces repeated errors, or a variety of programming faults, the processor can take a number of nested traps, leading to an eventual entry into RED_state.

RED_state entry is not normally recoverable. However, programs in RED_state can provide a useful diagnosis of the problem encountered prior to attempting corrective action.

The I-cache, P-cache and D-cache are automatically disabled by the hardware clearing the IC, PC and DC bits in the DCU Control Register (DCUCR) on entering RED_state. The L2-cache, L3-cache and W-cache state are unchanged.

7.10 Behavior on L2-cache DATA Error

TABLE 7-18 L2-cache data CE and UE errors *(1 of 3)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data
I-cache fill request with CE in the critical 32-byte L2-cache data and the data do get used later	UCC	Yes	Original data + original ecc
I-cache fill request with UE in the critical 32-byte L2-cache data and the data do get used later	UCU	Yes	Original data + original ecc
I-cache fill request with CE in the non-critical 32-byte (the 2nd 32-byte) L2-cache data, and the data (either the critical or the non-critical 32-byte data) do get used later	UCC	Yes	Original data + original ecc
I-cache fill request with UE in the non-critical 32-byte (the 2nd 32-byte) L2-cache data and the data (either the critical or the non-critical 32-byte data) do get used later	UCU	Yes	Original data + original ecc
I-cache fill request with CE in the critical 32-byte L2-cache data but the data never get used OR I-cache fill request with CE in the non-critical 32-byte (the 2nd 32-byte) L2-cache data but the data never get used	UCC	Yes	Original data + original ecc
I-cache fill request with UE in the critical 32-byte L2-cache data but the data never get used OR I-cache fill request with UE in the non-critical 32-byte (the 2nd 32-byte) L2-cache data but this data never get used	UCU	Yes	Original data + original ecc
D-cache 32-byte load request with CE in the critical 32-byte L2-cache data	UCC	Yes	Original data + original ecc
D-cache 32-byte load request with UE in the critical 32-byte L2-cache data	UCU	Yes	Original data + original ecc
D-cache 32-byte load request with CE in the non-critical 32-byte L2-cache data	EDC	No	Original data + original ecc
D-cache 32-byte load request with UE in the non-critical 32-byte L2-cache data	EDU	No	Original data + original ecc
D-cache FP-64-bit load request with CE in the critical 32-byte L2-cache data	UCC	Yes	Original data + original ecc
D-cache FP-64-bit load request with UE in the critical 32-byte L2-cache data	UCU	Yes	Original data + original ecc
D-cache FP-64-bit load request with CE in the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
D-cache FP-64-bit load request with UE in the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
D-cache block-load request with CE in the 1st 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
D-cache block-load request with UE in the 1st 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
D-cache atomic request with CE in the critical 32-byte L2-cache data	UCC	Yes	Original data + original ecc
D-cache atomic request with UE in the critical 32-byte L2-cache data	UCU	Yes	Original data + original ecc

TABLE 7-18 L2-cache data CE and UE errors *(2 of 3)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data
D-cache atomic request with CE in the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
D-cache atomic request with UE in the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache-for-several-reads-0/20 request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache-for-several-reads-0/20 request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache-for-one-read-1/21 request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache-for-one-read-1/21 request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache-for-several-writes-2/22 request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache-for-several-writes-2/22 request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache-for-one-write-3/23 request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache-for-one-write-3/23 request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache-for-instruction-17 request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache-for-instruction-17 request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
P-cache HW prefetch request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
P-cache HW prefetch request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc

TABLE 7-18 L2-cache data CE and UE errors *(3 of 3)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data
W-cache exclusive request with CE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDC	No	Original data + original ecc
W-cache exclusive request with UE in the critical 32-byte or the 2nd 32-byte L2-cache data	EDU	No	Original data + original ecc
W-cache eviction request, and CE in the critical 32-byte or the 2nd 32-byte L2-cache data	No error logged	No	W-cache eviction data overwrite the Original data
W-cache eviction request, and UE in the critical 32-byte or the 2nd 32-byte L2-cache data	No error logged	No	W-cache eviction data overwrite the original data
Direct ASI L2 data read request with CE in the 1st 32-byte or 2nd 32-byte L2-cache data	No error logged	No	Original data + original ecc
Direct ASI L2 data read request with tag UE in the 1st 32-byte or 2nd 32-byte L2-cache data	No error logged	No	Original data + original ecc
ASI L2 displacement flush read request with CE in the 1st 32-byte or 2nd 32-byte L2-cache data	WDC	No	L2-cache data flushed out, and corrected data written into L3-cache
ASI L2 displacement flush read request with UE in the 1st 32-byte or 2nd 32-byte L2-cache data	WDU	No	L2-cache data flushed out and written into L3-cache
Direct ASI L2 data write request with CE in the 1st 32-byte or 2nd 32-byte L2-cache data	No error logged	No	ASI write data overwrite original data
Direct ASI L2 data write request with tag UE in the 1st 32-byte or 2nd 32-byte L2-cache data	No error logged	No	ASI write data overwrite original data

TABLE 7-19 L2-cache data Writeback and Copyback Errors

Event	Error logged in AFSR	Data sent to L3-cache	Comment
L2 Writeback encountering CE in the 1st 32-byte or 2nd 32-byte L2-cache data	WDC	Corrected data + corrected ecc	Disrupting trap
L2 Writeback encountering UE in the 1st 32-byte or 2nd 32-byte L2-cache data	WDU	Original data + original ecc	Disrupting trap
Copyout hits in the L2 writeback buffer because the line is being victimized where a CE has already been detected	WDC	Corrected data + corrected ecc	Disrupting trap
Copyout hits in the L2 writeback buffer because the line is being victimized where a UE has already been detected	WDU	Original data + original ecc	Disrupting trap
Copyout encountering CE in the 1st 32-byte or 2nd 32-byte L2-cache data	CPC	Corrected data + corrected ecc	Disrupting trap
Copyout encountering UE in the 1st 32-byte or 2nd 32-byte L2-cache data	CPU	SIU flips the most significant 2 bits of data D[127:126] of the corresponding upper or lower 16-byte data	Disrupting trap

7.11 Behavior on L3-cache DATA Error

TABLE 7-20 L3-cache Data CE and UE errors *(1 of 4)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data	L1-cache data	Pipeline Action	Comment
I-cache fill request with CE in the critical 32-byte L3-cache data and the data do get used later	L3_UCC	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	Precise trap
I-cache fill request with UE in the critical 32-byte L3-cache data and the data do get used later	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	Precise trap
I-cache fill request with CE in the Non-critical 32-byte (the 2nd 32-byte) L3-cache data, and the data (either the critical or the non-critical 32-byte data) do get used later	L3_UCC	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	Precise trap
I-cache fill request with UE in the non-critical 32-byte (the 2nd 32-byte) L3-cache data and the data (either the critical or the non-critical 32-byte data) do get used later	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	Precise trap
I-cache fill request with CE in the critical 32-byte L3-cache data but the data never get used OR I-cache fill request with CE in the non-critical 32-byte (the 2nd 32-byte) L3-cache data but the data never get used	L3_UCC	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	No trap
I-cache fill request with UE in the critical 32-byte L3-cache data but the data never get used OR I-cache fill request with UE in the non-critical 32-byte (the 2nd 32-byte) L3-cache data but the data never get used	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in I-cache	Bad data dropped	No trap
D-cache 32-byte load request with CE in the critical 32-byte L3-cache data	L3_UCC	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data in D-cache	Bad data dropped	Precise trap
D-cache 32-byte load request with UE in the critical 32-byte L3-cache data	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data in D-cache	Bad data dropped	Precise trap
D-cache 32-byte load request with CE in the non-critical 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in D-cache	Good data taken	Disrupting trap
D-cache 32-byte load request with UE in the non-critical 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in D-cache	Good data taken	Disrupting trap

TABLE 7-20 L3-cache Data CE and UE errors *(2 of 4)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data	L1-cache data	Pipeline Action	Comment
D-cache FP-64-bit load request with CE in the critical 32-byte L3-cache data	L3_UCC	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data in D-cache, but good data in P-cache	Bad data dropped	Precise trap
D-cache FP-64-bit load request with UE in the critical 32-byte L3-cache data	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad data in D-cache, but not in P-cache	Bad data dropped	Precise trap
D-cache FP-64-bit load request with CE in the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in D-cache and P-cache	Good data taken	Disrupting trap
D-cache FP-64-bit load request with UE in the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Good 32-byte data in D-cache only	Good 32-byte data taken	Disrupting trap
D-cache block-load request with CE in the 1st 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in P-cache buffer	Good data taken	Disrupting trap
D-cache block-load request with UE in the 1st 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data in P-cache buffer	Bad data in FP register file	Deferred trap
D-cache atomic request with CE in the critical 32-byte L3-cache data	L3_UCC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in W-cache	Bad data dropped	Precise trap
D-cache atomic request with UE in the critical 32-byte L3-cache data	L3_UCU	Yes	Original data + original ecc moved from L3-cache to L2-cache	Bad critical 32-byte data and UE information, and and good critical 32-byte in W-cache	Bad data dropped	Precise trap (when the line is evicted out from the W-cache again, based on the UE status bit, W-cache flips the 2 least significant ecc check bits C[1:0] in both lower and upper 16-byte)
D-cache atomic request with CE in the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data in W-cache	Good data taken	Disrupting trap

TABLE 7-20 L3-cache Data CE and UE errors *(3 of 4)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data	L1-cache data	Pipeline Action	Comment
D-cache atomic request with UE in the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Good critical 32-byte data and bad non-critical 32-byte data and UE information in W-cache	Good 32-byte data taken	Disrupting trap (when the line is evicted out from the W-cache again, based on the UE status bit, W-cache flips the 2 least significant ecc check bits C[1:0] in both lower and upper 16-byte)
P-cache-for-several-reads-0/20 request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data not in P-cache	No action	Disrupting trap
P-cache-for-several-reads-0/20 request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in P-cache	No action	Disrupting trap
P-cache-for-one-read-1/21 request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data not in P-cache	No action	Disrupting trap
P-cache-for-one-read-1/21 request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in P-cache	No action	Disrupting trap
P-cache-for-several-writes-2/22 request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data not in P-cache	No action	Disrupting trap
P-cache-for-several-writes-2/22 request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in P-cache	No action	Disrupting trap
P-cache-for-one-write-3/23 request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data not in P-cache	No action	Disrupting trap
P-cache-for-one-write-3/23 request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in P-cache	No action	Disrupting trap
P-cache-for-instruction-17 request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	Good data not in P-cache	No action	Disrupting trap

TABLE 7-20 L3-cache Data CE and UE errors *(4 of 4)*

Event	Error logged in AFSR	fast ecc error trap	L2-cache data	L1-cache data	Pipeline Action	Comment
P-cache-for-instruction-17 request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	Bad data not in P-cache	No action	Disrupting trap
W-cache exclusive request with CE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDC	No	Original data + original ecc moved from L3-cache to L2-cache	W-cache gets the permission to modify the data	W-cache proceeds to modify the data	Disrupting trap
W-cache exclusive request with UE in the critical 32-byte or the 2nd 32-byte L3-cache data	L3_EDU	No	Original data + original ecc moved from L3-cache to L2-cache	W-cache gets the permiss-ion to modify the data, UE information is stored in W-cache	W-cache proceeds to modify the data	Disrupting trap (when the line is evicted out from the W-cache again, based on the UE status bit sent from L2, W-cache flips the 2 least significant ecc check bits C[1:0] in both lower and upper 16-byte)

TABLE 7-21 L3-cache ASI Access Errors

Event	Error logged in AFSR	fast ecc error trap	L3-cache data	L1-cache data	Pipeline Action	Comment
Direct ASI L3-cache data read request with CE in the 1st 32-byte or 2nd 32-byte L3-cache data	No error logged	No	Original data + original ecc	N/A	No action	No trap (ASI access will get corrected data if ec_ecc_en is asserted. Otherwise, the original data is returned.)
Direct ASI L3-cache data read request with tag UE in the 1st 32-byte or 2nd 32-byte L3-cache data	No error logged	No	Original data + original ecc	N/A	No action	No trap
Direct ASI L3-cache data write request with CE in the 1st 32-byte or 2nd 32-byte L3-cache data	No error logged	No	ASI write data overwrite original data	N/A	No action	No trap
Direct ASI L3-cache data write request with tag UE in the 1st 32-byte or 2nd 32-byte L3-cache data	No error logged	No	ASI write data overwrite original data	N/A	No action	No trap

TABLE 7-22 L3-cache Data Writeback and Copyback Errors

Event	Error logged in AFSR	Data sent to SIU	Comment
L3 Writeback encountering CE in the 1st 32-byte or 2nd 32-byte L3-cache data	L3_WDC	Corrected data + corrected ecc	Disrupting trap
L3 Writeback encountering UE in the 1st 32-byte or 2nd 32-byte L3-cache data	L3_WDU	SIU flips the most significant 2 bits of data D[127:126] of the corresponding upper or lower 16-byte data	Disrupting trap
Copyout hits in the L3 writeback buffer because the line is being victimized where a CE has already been detected	L3_WDC	Corrected data + corrected ecc	Disrupting trap
Copyout hits in the L3 writeback buffer because the line is being victimized where a UE has already been detected	L3_WDU	SIU flips the most significant 2 bits of data D[127:126] of the corresponding upper or lower 16-byte data	Disrupting trap
Copyout encountering CE in the 1st 32-byte or 2nd 32-byte L3-cache data	L3_CPC	Corrected data + corrected ecc	Disrupting trap
Copyout encountering UE in the 1st 32-byte or 2nd 32-byte L3-cache data	L3_CPU	SIU flips the most significant 2 bits of data D[127:126] of the corresponding upper or lower 16-byte data	Disrupting trap

---

Note – “D-Cache FP-64-bit load” means any of the following 4 kinds of FP-load instructions, that is, LDDF, or LDF, or LDDFA (with some ASIs), or LDFA (with some ASIs).

AFAR points to 16-byte boundary as dictated by the system bus.

When UE and CE occur in the same 32-byte data, both CE and UE will be reported but AFAR will point to the UE case on the 16-byte boundary.

7.12 Behavior on L2-cache TAG Errors

This section presents information about L2-cache tag errors.

TABLE 7-23 L2-cache Tag CE and UE errors *(1 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
L2-cache access for I-cache request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
L2-cache access for I-cache request with tag UE but the data returned to I-cache get used later	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2-cache access for I-cache request with tag UE but the data returned to I-cache do not get used later	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
D-cache 32-byte load request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
D-cache 32-byte load request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
D-cache FP-64-bit load request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
D-cache FP-64-bit load request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2-cache access for D-cache block load request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
L2-cache access for D-cache block load request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
D-cache atomic request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
D-cache atomic request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2-cache access for Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17 request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
L2-cache access for Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17 request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)

TABLE 7-23 L2-cache Tag CE and UE errors *(2 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
L2-cache access for P-cache HW prefetch request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
L2-cache access for P-cache HW prefetch request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
(stores) W-cache exclusive request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
(stores) W-cache exclusive request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
W-cache block store request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (L2-cache pipe retries the request)
W-cache block store request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
W-cache eviction request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	No action	Disrupting trap (eviction data written into L2-cache)
W-cache eviction request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (W-cache eviction is dropped)
L2-cache eviction tag read request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	No action	Disrupting trap (L2-cache pipe retries the request)
L2-cache eviction tag read request with tag UE	TUE	No	Yes	Original tag	N/A	No action	Disrupting trap (eviction is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the 64-byte data to I-cache and writes the 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the 64-byte data to I-cache and writes the 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for 32-byte load and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)

TABLE 7-23 L2-cache Tag CE and UE errors *(3 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for 32-byte load and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for FP-64-bit load and writes 64-byte data to P-cache and L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for FP-64-bit load and writes 64-byte data to P-cache and L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the 64-byte to P-cache block load buffer for block load and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the 64-byte to P-cache block load buffer for block load and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for atomic request and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for atomic request and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the critical 32-byte and then 2nd 32-byte to P-cache for Prefetch 0,1,2,3 request and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)

TABLE 7-23 L2-cache Tag CE and UE errors *(4 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the critical 32-byte and then 2nd 32-byte to P-cache for Prefetch 0,1,2,3 request and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache to L2-cache fill request with L2-cache tag CE (forwards the critical 32-byte and then 2nd 32-byte to W-cache for W-cache exclusive request and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache to L2-cache fill request with L2-cache tag UE (forwards the critical 32-byte and then 2nd 32-byte to W-cache for W-cache exclusive request and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU fill request with L2-cache tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU fill request with L2-cache tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward and fill request with L2-cache tag CE (forwards the critical 64-byte to I-cache and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward and fill request with L2-cache tag UE (forwards the critical 64-byte to I-cache and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward only request with L2-cache tag CE (forwards the critical 32-byte to I-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward only request with L2-cache tag UE (forwards the critical 32-byte to I-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward and fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for 32-byte load and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)

TABLE 7-23 L2-cache Tag CE and UE errors *(5 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
SIU forward and fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for 32-byte load and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (data will not be stored in L2-cache)
SIU forward only request with L2-cache tag CE (forwards the critical 32-byte to D-cache for 32-byte load)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward only request with L2-cache tag UE (forwards the critical 32-byte to D-cache for 32-byte load)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward and fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for FP-64-bit load and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward and fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for FP-64-bit load and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward only request with L2-cache tag CE (forwards the critical 32-byte to D-cache for FP-64-bit load)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward only request with L2-cache tag UE (forwards the critical 32-byte to D-cache for FP-64-bit load)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward and fill request with L2-cache tag CE (forwards the critical 32-byte to D-cache for atomic request and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward and fill request with L2-cache tag UE (forwards the critical 32-byte to D-cache for atomic request and writes 64-byte data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)

TABLE 7-23 L2-cache Tag CE and UE errors *(6 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
SIU forward and fill request with L2-cache tag CE (forwards the critical 32-byte and then 2nd 32-byte to P-cache for Prefetch 0,1,2,3 request and writes 64-byte data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward and fill request with L2-cache tag UE (forwards the critical 32-byte and then 2nd 32-byte to P-cache for Prefetch 0,1,2,3 request and writes data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU forward and fill request with L2-cache tag CE (forwards the critical 32-byte and then 2nd 32-byte to W-cache for W-cache exclusive request and writes data to L2-cache)	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU forward and fill request with L2-cache tag UE (forwards the critical 32-byte and then 2nd 32-byte to W-cache for W-cache exclusive request and writes data to L2-cache)	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2-cache Tag update request by SIU with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L2-cache Tag update request by SIU local transaction with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2-cache Tag update request by SIU foreign transaction with tag UE	TUE_SH	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
SIU copyout request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
SIU copyout request with tag UE	TUE_SH	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)

TABLE 7-23 L2-cache Tag CE and UE errors *(7 of 7)*

Event	Errors logged in AFSR	flag fast ecc error	Error Pin	L2-cache Tag	L1-cache data	Pipeline Action	Comment
ASI L2 data read request with tag CE	No error logged	No	No	Original tag	N/A	No action	No trap
ASI L2 data read request with tag UE	No error logged	No	No	Original tag	N/A	No action	No trap
ASI L2 data write request with tag CE	No error logged	No	No	Original tag	N/A	No action	No trap
ASI L2 data write request with tag UE	No error logged	No	No	Original tag	N/A	No action	No trap
direct ASI L2 tag read request with tag CE	No error logged	No	No	Original tag	N/A	No action	No trap (ASI access get original tag)
direct ASI L2 tag read request with tag UE	No error logged	No	No	Original tag	N/A	No action	No trap (ASI access get original tag)
ASI L2 displacement flush read request with tag CE	THCE	No	No	L2 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
ASI L2 displacement flush read request with tag UE	TUE	No	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
ASI tag write request with tag CE	No error logged	No	No	new tag written in	N/A	No action	No trap
ASI tag write request with tag UE	No error logged	No	No	new tag written in	N/A	No action	No trap

TABLE 7-24 L2-cache Tag CE and UE Errors

Event	Error logged in AFSR	Error Pin	L2-cache tag	Snoop result sent to SIU	Comment
Snoop request with tag CE and without E-to-S upgrade	THCE	No	L2 Pipe corrects the tag	Good L2-cache state	Disrupting trap
Snoop request with tag CE and with E-to-S upgrade	THCE	No	L2 Pipe performs E-to-S based on corrected tag	Good L2-cache state	Disrupting trap
Snoop request with tag UE	TUE_SH	Yes	Original state	force a miss snoop result	Disrupting trap

7.13 Behavior on L3-cache TAG Errors

This section presents information about L3-cache tag errors.

TABLE 7-25 L3-cache Tag CE and UE Errors *(1 of 4)*

Event	Errors logged in AFSR	Error Pin	L3-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache access for I-cache request with tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	Good data installed in I-cache	Good data taken	Disrupting trap
L3-cache access for I-cache request with tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache access for I-cache request with tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	Good data installed in I-cache	Good data taken	Disrupting trap
L3-cache access for I-cache request with tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
D-cache 32-byte load request with L3 tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	Good data installed in D-cache	Good data taken	Disrupting trap
D-cache 32-byte load request with L3 tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
D-cache 32-byte load request with L3 tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	Good data installed in D-cache	Good data taken	Disrupting trap
D-cache 32-byte load request with L3 tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	No action	Disrupting trap (the request is dropped)
D-cache FP-64-bit load request with L3 tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	Good data installed in D-cache and P-cache	Good data taken	Disrupting trap
D-cache FP-64-bit load request with L3 tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
D-cache FP-64-bit load request with L3 tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	Good data installed in D-cache and P-cache	Good data taken	Disrupting trap
D-cache FP-64-bit load request with L3 tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache access for D-cache block load request with tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	Good data in P-cache block load buffer	Good data taken	Disrupting trap

TABLE 7-25 L3-cache Tag CE and UE Errors *(2 of 4)*

Event	Errors logged in AFSR	Error Pin	L3-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache access for D-cache block load request with tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache access for D-cache block load request with tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	Good data in P-cache block load buffer	Good data taken	Disrupting trap
L3-cache access for D-cache block load request with tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
D-cache atomic request with L3 tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	Good data installed in D-cache	Good data taken	Disrupting trap
D-cache atomic request with L3 tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
D-cache atomic request with L3 tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	Good data installed in D-cache and W-cache	Good data taken	Disrupting trap
D-cache atomic request with L3 tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L3-cache access for Prefetch 0, 1, 2, 20, 21, 22 request with tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	data installed in P-cache	N/A	Disrupting trap
L3-cache access for Prefetch 3, 23, 17 request with tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	data not installed in P-cache	N/A	Disrupting trap
L3-cache access for Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17 request with tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
L3-cache access for Prefetch 0, 1, 2, 20, 21, 22 request with tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	data installed in P-cache	N/A	Disrupting trap
L3-cache access for Prefetch 3, 23, 17 request with tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	data not installed in P-cache	N/A	Disrupting trap

TABLE 7-25 L3-cache Tag CE and UE Errors *(3 of 4)*

Event	Errors logged in AFSR	Error Pin	L3-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache access for Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17 request with tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
(stores) W-cache exclusive request with L3 tag CE (request hits L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	L2 pipe sends (valid & grant) to W-cache	N/A	Disrupting trap
(stores) W-cache exclusive request with L3 tag CE (request misses L2-cache)	L3_THCE	No	L3 Pipe corrects the tag	N/A	N/A	Disrupting trap (the request is retried)
(stores) W-cache exclusive request with L3 tag UE (request hits L2-cache)	L3_TUE	Yes	Original tag	L2 Pipe gives both (valid & grant) to W-cache	N/A	Disrupting trap
(stores) W-cache exclusive request with L3 tag UE (request misses L2-cache)	L3_TUE	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
L2 Writeback request with L3 tag CE	L3_THCE	No	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
L2 Writeback request with L3 tag UE	L3_TUE	Yes	Original tag	N/A	No action	Disrupting trap (the request is dropped)
L3-cache eviction tag read request with tag CE	L3_THCE	No	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
L3-cache eviction tag read request with tag UE	L3_TUE	Yes	Original tag	N/A	No action	Disrupting trap (the request is dropped)
L3-cache Tag update request by SIU with tag CE	L3_THCE	No	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)

TABLE 7-25 L3-cache Tag CE and UE Errors *(4 of 4)*

Event	Errors logged in AFSR	Error Pin	L3-cache Tag	L1-cache data	Pipeline Action	Comment
L3-cache Tag update request by SIU local transaction with tag UE	L3_TUE	Yes	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
L3-cache Tag update request by SIU foreign transaction with tag UE	L3_TUE_SH	Yes	Original tag	N/A	No action	Disrupting trap (the request is dropped)
SIU copyback request with tag CE	L3_THCE	No	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
SIU copyback request with tag UE	L3_TUE_SH	Yes	Original tag	N/A	N/A	Disrupting trap (the request is dropped)
Direct ASI L3-cache tag read request with tag CE	No error logged	No	Original tag	N/A	No action	No trap (ASI access get original tag
Direct ASI L3-cache tag read request with tag UE	No error logged	No	Original tag	N/A	No action	No trap (ASI access get original tag
ASI L3-cache displacement flush read request with tag CE	L3_THCE	No	L3 Pipe corrects the tag	N/A	No action	Disrupting trap (the request is retried)
ASI L3-cache displacement flush read request with tag UE	L3_TUE	Yes	Original tag	N/A	No action	Disrupting trap (the request is dropped)
ASI tag write request with tag CE	No error logged	No	new tag written in	N/A	No action	No trap
ASI tag write request with tag UE	No error logged	No	new tag written in	N/A	No action	No trap

Note – “D-cache FP-64-bit load” means any of the following 4 kinds of FP-load instructions, that is, LDDF, or LDF, or LDDFA (with some ASIs), or LDFA (with some ASIs).

TABLE 7-26 L3-cache Tag CE and UE Errors

Event	Error logged in AFSR	Error Pin	L3-cache tag	snoop result sent to SIU	Comment
Snoop request with L3 tag CE and without E-to-S upgrade	L3_THCE	No	L3 Pipe corrects the tag	Good L3-cache state	Disrupting trap
Snoop request with L3 tag CE and with E-to-S upgrade	L3_THCE	No	E-to-S upgrade based on corrected tag	Good L3-cache state	Disrupting trap
Snoop request with L3 tag UE	L3_TUE_SH	Yes	Original state	Force a miss snoop result	Disrupting trap

7.14 Behavior on System Bus Errors

1. Note that dropping the precise trap and taking the deferred trap instead is not visible to software.

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(1 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
I-cache fill request with CE in the critical 32-byte data from system bus	CE	No	Corrected data + corrected ecc	S	Good data in I-cache	Good data taken	Disrupting trap
I-cache fill request with UE in the critical 32-byte data from system bus	UE	Yes	Raw UE data + raw ecc	S	Bad data not installed in I-cache	Bad data dropped	Deferred trap (UE) will be taken and Precise trap (fast ecc error) will be dropped[1]
I-cache fill request with CE in the non-critical (2nd) 32-byte data from system bus	CE	No	Corrected data + corrected ecc	S	No action	No action	Disrupting trap
I-cache fill request with UE in the non-critical (2nd) 32-byte data from system bus	UE	No	Raw UE data + raw ecc	S	No action	No action	Deferred trap
D-cache load 32-byte fill request with CE in the critical 32-byte data from system bus	CE	No	Corrected data + corrected ecc	E/S	Good data in D-cache	Good data taken	Disrupting trap
D-cache load 32-byte fill request with UE in the critical 32-byte data from system bus	UE	Yes	Raw UE data + raw ecc	E/S	Bad data in D-cache	Bad data dropped	Deferred trap (UE) will be taken and Precise trap (fast ecc error) will be dropped[1]
D-cache load 32-byte fill request with CE in the non-critical (2nd) 32-byte data from system bus	CE	No	Corrected data + corrected ecc	E/S	No action	No action	Disrupting trap
D-cache load 32-byte fill request with UE in the non-critical (2nd) 32-byte data from system bus	UE	No	Raw UE data + raw ecc	E/S	No action	No action	Deferred trap
D-cache FP-64-bit load fill request with CE in the critical 32-byte data from system bus	CE	No	Corrected data + corrected ecc	E/S		Good data taken	Disrupting trap
D-cache FP-64-bit load fill request with UE in the critical 32-byte data from system bus	UE	Yes	Raw UE data + raw ecc	E/S	Bad data in D-cache, but not in P-cache	Bad data dropped	Deferred trap (UE) will be taken and Precise trap (fast ecc error) will be dropped[1]

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(2 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
D-cache FP-64-bit load fill request with CE in the non-critical (2nd) 32-byte data from system bus	CE	No	Corrected data + corrected ecc	E/S	Good & critical 32-byte data in D-Cache and good & critical 32-byte or full 64-byte data in P-cache	No action	Disrupting trap
D-cache FP-64-bit load fill request with UE in the non-critical (2nd) 32-byte data from system bus	UE	No	Raw UE data + raw ecc	E/S	Bad data not in P-cache	No action	Deferred trap
D-cache block-load fill request with CE in the critical 32-byte data from system bus OR D-cache block-load fill request with CE in the non-critical (2nd) 32-byte data from system bus	CE	No	Not installed	N/A	Good data in P-cache block-load buffer	Good data taken	Disrupting trap
D-cache block-load fill request with UE in the critical 32-byte data from system bus OR D-cache block-load fill request with UE in the non-critical (2nd) 32-byte data from system bus	UE	No	Not installed	N/A	Bad data in P-cache block-load buffer	Bad data in FP register file	Deferred trap
D-cache atomic fill request with CE in the critical 32-byte data from system bus	CE	No	Corrected data + corrected ecc	M	Good & critical 32-byte data and non-critical 32-byte data in W-cache	Good data taken	Disrupting trap
D-cache atomic fill request with UE in the critical 32-byte data from system bus	UE	Yes	Raw UE data + raw ecc	M	Bad critical 32-byte data and UE information, and good non-critical 32-byte data in W-cache	Bad data dropped	Deferred trap (UE) will be taken and Precise trap (fast ecc error) will be dropped[1] (when the line is evicted out from the W-cache again, based on the UE status bit sent from L2-cache, W-cache flips the 2 least significant ecc check bits C[1:0] in both lower and upper 16-byte)
D-cache atomic fill request with CE in the non-critical (2nd) 32-byte data from system bus	CE	No	Corrected data + corrected ecc	M	Good & critical 32-byte data and non-critical 32-byte data in W-cache	Good critical 32-byte data taken	Disrupting trap

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(3 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
D-cache atomic fill request with UE in the non-critical (2nd) 32-byte data from system bus	UE	No	Raw UE data + raw ecc	M	Good critical 32-byte data, and bad non-critical 32-byte data and UE information is in W-cache	Good critical 32-byte data taken	Deferred trap (when the line is evicted out from the W-cache again, based on the UE status bit sent from L2-cache, W-cache flips the 2 least significant ecc check bits C[1:0] in both lower and upper 16-byte)
Prefetch-for-several-reads-0/20 fill request with CE in the critical 32-byte data from system bus due to non-RTSR transaction OR Prefetch-for-several-reads-0/20 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to non-RTSR transaction	CE	No	Corrected data + corrected ecc	E/S	Good data in P-cache	No action	Disrupting trap
Prefetch-for-several-reads-0/20 fill request with CE in the critical 32-byte data from system bus due to RTSR transaction OR Prefetch-for-several-reads-0/20 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to RTSR transaction	CE	No	Corrected data + corrected ecc	M/O/Os	Good data in P-cache	No action	Disrupting trap
Prefetch-for-one-read-1/21 fill request with CE in the critical 32-byte data from system bus due to non-RTSR transaction OR Prefetch-for-one-read-1/21 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to non-RTSR transaction	CE	No	Not installed	N/A	Good data in P-cache	No action	Disrupting trap

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(4 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
Prefetch-for-one-read-1/21 fill request with CE in the critical 32-byte data from system bus due to RTSR transaction OR Prefetch-for-one-read-1/21 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to RTSR transaction	CE	No	Corrected data + corrected ecc	M/O/ Os	Good data in P-cache	No action	Disrupting trap
Prefetch-for-several-writes-2/22 fill request with CE in the critical 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction OR Prefetch-for-several-writes-2/22 fill request with CE in the non-critical (2nd) 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction	CE	No	Corrected data + corrected ecc	M	Data installed in P-cache	No action	Disrupting trap
Prefetch-for-one-write-3/23 fill request with CE in the critical 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction OR Prefetch-for-one-write-3/23 fill request with CE in the non-critical (2nd) 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction	CE	No	Corrected data + corrected ecc	M	Data not installed in P-cache	No action	Disrupting trap
Prefetch-for-instruction 17 fill request with CE in the critical 32-byte data from system bus due to non-RTSR transaction OR Prefetch-for-instruction 17 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to non-RTSR transaction	CE	No	Corrected data + corrected ecc	N/A	Good data not installed in P-cache	No action	Disrupting trap

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(5 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
Prefetch-for-instruction 17 fill request with CE in the critical 32-byte data from system bus due to RTSR transaction OR Prefetch-for-instruction 17 fill request with CE in the non-critical (2nd) 32-byte data from system bus due to RTSR transaction	CE	No	Corrected data + corrected ecc	M/O/Os	Good data not installed in P-cache	No action	Disrupting trap
Prefetch 0, 1, 20, 21, 17 fill request with UE in the critical 32-byte data from system bus due to non-RTSR transaction OR Prefetch 0, 1, 20, 21, 17 fill request with UE in the 2nd 32-byte data from system bus due to non-RTSR transaction	DUE	No	Not installed	N/A	Bad data not installed in P-cache	No action	Disrupting trap
P-cache 0, 1, 20, 21, 17 fill request with UE in the critical 32-byte data from system bus due to RTSR transaction OR P-cache 0, 1, 20, 21, 17 fill request with UE in the 2nd 32-byte data from system bus due to RTSR transaction	DUE	No	Raw UE data + raw ecc	M/O/Os	Bad data not installed in P-cache	No action	Disrupting trap
Prefetch 2, 3, 22, 23 fill request with UE in the critical 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction OR Prefetch2, 3, 22, 23 fill request with UE in the 2nd 32-byte data from system bus regardless non-RTOR transaction or RTOR transaction	DUE	No	Raw UE data + raw ecc	M	Data not installed in P-cache	No action	Disrupting trap

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(6 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
(stores) W-cache exclusive fill request with CE in the critical 32-byte data from system bus OR (stores) W-cache exclusive fill request with CE in the 2nd 32-byte data from system bus	CE	No	Corrected data + corrected ecc	M	W-cache gets the permission to modify the data after all 64-byte of data have been received	W-cache proceeds to modify the data	Disrupting trap
(stores) W-cache exclusive fill request with UE in the critical 32-byte data from system bus OR (stores) W-cache exclusive fill request with UE in the 2nd 32-byte data from system bus	DUE	No	Raw UE data + raw ecc	M	W-cache gets the permission to modify the data after all 64-byte of data have been received	W-cache proceeds to modify the data and UE information is sent to and stored in W-cache	Disrupting trap
Cacheable I-cache fill request for unmapped address	TO	Yes	Not installed	N/A	garbage data not installed in I-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Non-cacheable I-cache fill request for unmapped address	TO	Yes	Not installed	N/A	garbage data not installed in I-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Cacheable D-cache 32-byte fill request for unmapped address	TO	Yes	Not installed	N/A	garbage data installed in D-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Non-cacheable D-cache 32-byte fill request for unmapped address	TO	Yes	not installed	N/A	garbage data not installed in D-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Cacheable D-cache FP-64-bit load fill request for unmapped address	TO	Yes	Not installed	N/A	garbage data installed in D-cache, but not in P-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Non-cacheable D-cache FP-64-bit load fill request for unmapped address	TO	Yes	Not installed	N/A	garbage data not installed in both D-cache and P-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Cacheable D-cache block-load request for unmapped address	TO	No	Not installed	N/A	garbage data installed in P-cache block-load buffer	garbage data in FP register file	Deferred trap
Non-cacheable D-cache block-load request for unmapped address	TO	No	Not installed	N/A	garbage data installed in P-cache block-load buffer	garbage data in FP register file	Deferred trap

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(7 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
Cacheable D-cache atomic fill request for unmapped address	TO	Yes	not installed	N/A	L2L3 unit gives both (~grant) and valid to W-cache	garbage data not taken	Deferred trap (TO) will be taken and Precise trap (fast ecc error) will be dropped[1]
Cacheable Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17, fill request for unmapped address	DTO	No	not installed	N/A	garbage data not installed in P-cache	No action	Disrupting trap
Non-cacheable Prefetch 0, 1, 2, 3, 20, 21, 22, 23, 17, fill request for unmapped address	DTO	No	not installed	N/A	garbage data not installed in P-cache	No action	Disrupting trap
(cacheable stores) W-cache exclusive fill request with unmapped address	DTO	No	not installed	N/A	L2L3 unit gives both (~grant) and valid to W-cache	W-cache will drop the store	Disrupting trap
W-cache cacheable block store missing L2-cache (write stream) with unmapped address OR W-cache cacheable block store commit request (write stream) with unmapped address	TO	No	N/A	N/A	N/A	N/A	Deferred trap
Non-cacheable W-cache store with unmapped address	TO	No	N/A	N/A	N/A	N/A	Deferred trap
Non-cacheable W-cache block store with unmapped address	TO	No	N/A	N/A	N/A	N/A	Deferred trap
Ecache eviction with unmapped address	TO	No	N/A	N/A	N/A	N/A	Deferred trap writeback operation is terminated & data coherency is lost
Outgoing interrupt request with unmapped address	TO	No	N/A	N/A	N/A	N/A	Deferred trap the corresponding Busy bit in Interrupt Vector Dispatch Status Register will be cleared
Non-cacheable I-cache fill request with BERR response	BERR	Yes	Not installed	N/A	Not installed	garbage data not taken	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1]
Non-cacheable D-cache 32-byte fill request with BERR response	BERR	Yes	Not installed	N/A	Not installed	garbage data not taken	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1]
Non-cacheable D-cache FP-64-bit fill request with BERR response	BERR	Yes	Not installed	N/A	Not installed	garbage data not taken	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1]

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(8 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
Non-cacheable D-cache block-load fill request with BERR response	BERR	No	Not installed	N/A	Not installed	garbage data not taken	Deferred trap
Non-cacheable Prefetch 0, 1, 2, 3 fill request with BERR response	DBERR	No	Not installed	N/A	Not installed	garbage data not taken	Disrupting trap
Cacheable I-cache fill request with BERR in the critical 32-byte data from system bus	BERR	Yes	garbage data installed	S	Not installed	garbage data not taken	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1] the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable I-cache fill request with BERR in the non-critical (2nd) 32-byte data from system bus	BERR	No	garbage data installed	S	No action	No action	Deferred trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache load 32-byte fill request with BERR in the critical 32-byte data from system bus	BERR	Yes	garbage data installed	E/S	Installed	garbage data not taken	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1] the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache load 32-byte fill request with BERR in the non-critical (2nd) 32-byte data from system bus	BERR	No	garbage data installed	E/S	No action	No action	Deferred trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache FP-64-bit load fill request with BERR in the critical 32-byte data from system bus	BERR	Yes	garbage data installed	E/S	garbage data in D-cache, but not in P-cache	garbage data dropped	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1] the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache FP-64-bit load fill request with BERR in the non-critical (2nd) 32-byte data from system bus	BERR	No	garbage data installed	E/S	garbage data not in P-cache	No action	Deferred trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache block-load fill request with BERR in the critical 32-byte data from system bus OR Cacheable D-cache block-load fill request with BERR in the non-critical (2nd) 32-byte data from system bus	BERR	No	Not installed	N/A	garbage data in P-cache block-load buffer	garbage data in FP register file	Deferred trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(9 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
Cacheable D-cache atomic fill request with BERR in the critical 32-byte data from system bus	BERR	Yes	garbage data installed	M	garbage critical 32-byte data and UE information is sent and stored in W-cache	garbage data dropped	Deferred trap (BERR) will be taken and Precise trap (fast ecc error) will be dropped[1] the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable D-cache atomic fill request with BERR in the non-critical (2nd) 32-byte data from system bus	BERR	No	garbage data installed	M	garbage non-critical 32-byte data and UE information is sent and stored in W-cache	Good data taken	Deferred trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable Prefetch 0, 1, 20, 21, 17 fill request with BERR in the critical 32-byte data from system bus due to non-RTSR transaction OR Cacheable Prefetch 0, 1, 20, 21, 17 fill request with BERR in the 2nd 32-byte data from system bus due to non-RTSR transaction	DBERR	No	Not installed	N/A	garbage data not installed in P-cache	No action	Disrupting trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
Cacheable P-cache 0, 1, 20, 21, 17 fill request with BERR in the critical 32-byte data from system bus due to RTSR transaction OR Cacheable P-cache 0, 1, 20, 21, 17, fill request with BERR in the 2nd 32-byte data from system bus due to RTSR transaction	DBERR	No	garbage data installed	M/O/Os	garbage data not installed in P-cache	No action	Disrupting trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped

TABLE 7-27 System Bus CE, UE, TO, DTO, BERR, DBERR errors *(10 of 10)*

Event	Error logged in AFSR	flag fast ecc error	L2-cache data	L2-cache state	L1-cache data	Pipeline Action	Comment
cacheable Prefetch-for-several-writes-2/22 fill request with BERR in the critical 32-byte data from system bus regardless non-RTSR transaction or RTSR transaction OR cacheable Prefetch-for-several-writes-2/22 fill request with BERR in the 2nd 32-byte data from system bus regardless non-RTSR transaction or RTSR transaction	DBERR	No	garbage data installed	M	garbage data not installed in P-cache	No action	Disrupting trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
cacheable Prefetch-for-one-write-3/23 fill request with BERR in the critical 32-byte data from system bus regardless non-RTSR transaction or RTSR transaction OR cacheable Prefetch-for-one-write-3/23 fill request with BERR in the 2nd 32-byte data from system bus regardless non-RTSR transaction or RTSR transaction	DBERR	No	garbage data installed	M	garbage data not installed in P-cache	No action	Disrupting trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped
(cacheable stores) W-cache exclusive fill request with BERR in the critical 32-byte data from system bus OR (cacheable stores) W-cache exclusive fill request with BERR in the 2nd 32-byte data from system bus	DBERR	No	garbage data installed	M	W-cache gets the permission to modify the data after all 64-byte of data have been received	W-cache proceeds to modify the data and UE information is sent to and stored in W-cache	Disrupting trap the 2 least significant data bits [1:0] in both lower and upper 16-byte are flipped

TABLE 7-28 System Bus EMC and EMU errors *(1 of 2)*

Event	Error logged AFSR	Error Pin	Comment
I-cache fill request with CE in the microtag of the critical 32-byte data from system bus OR I-cache fill request with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMC	No	Disrupting trap
I-cache fill request with UE in the microtag of the critical 32-byte data from system bus OR I-cache fill request with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMU	Yes	Deferred trap
D-cache fill request with CE in the microtag of the critical 32-byte data from system bus OR D-cache fill request with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMC	No	Disrupting trap
D-cache fill request with UE in the microtag of the critical 32-byte data from system bus OR D-cache fill request with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMU	Yes	Deferred trap
D-cache atomic fill request with CE in the microtag of the critical 32-byte data from system bus OR D-cache atomic fill request with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMC	No	Disrupting trap
D-cache atomic fill request with UE in the microtag of the critical 32-byte data from system bus OR D-cache atomic fill request with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMU	Yes	Deferred trap
P-cache fill request with CE in the microtag of the critical 32-byte data from system bus OR P-cache fill request with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMC	Yes	Disrupting trap
P-cache fill request with UE in the microtag of the critical 32-byte data from system bus OR P-cache fill request with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMU	Yes	Deferred trap
(stores) W-cache exclusive fill request with CE in the microtag of the critical 32-byte data from system bus OR (stores) W-cache exclusive fill request with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMC	No	Disrupting trap

TABLE 7-28 System Bus EMC and EMU errors *(2 of 2)*

Event	Error logged AFSR	Error Pin	Comment
(stores) W-cache exclusive fill request with UE in the microtag of the critical 32-byte data from system bus OR (stores) W-cache exclusive fill request with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	EMU	Yes	Deferred trap

Note – For microtag error, when data is delivered to L2-cache, then SIU will report the error. If the data is not delivered to L2-cache, SIU will not report the error.

"D-cache FP-64-bit load" means any of the following 4 kinds of FP-load instructions, that is, LDDF, or LDF, or LDDFA (with some ASIs), or LDFA (with some ASIs) where the cited ASIs are tabulated below.

TABLE 7-29 System Bus IVC and IVU errors

Event	Error logged in AFSR	Error Pin	Interrupt Vector Receive Register Busy bit setting	Interrupt Data Register	Comment
Interrupt vector with CE in the critical 32-byte data from system bus OR Interrupt vector with CE in the non-critical (2nd) 32-byte data from system bus	IVC	No	Yes	Corrected interrupt data	Disrupting trap & Interrupt taken
Interrupt vector with UE in the critical 32-byte data from system bus OR Interrupt vector with UE in the non-critical (2nd) 32-byte data from system bus	IVU	No	No	garbage data	Disrupting trap & Interrupt dropped
Interrupt vector with CE in the microtag of the critical 32-byte data from system bus OR Interrupt vector with CE in the microtag of the non-critical (2nd) 32-byte data from system bus	IMC	No	Yes if no IVU	Corrected interrupt data	Disrupting trap & Interrupt taken if no IVU Interrupt dropped if IVU
Interrupt vector with UE in the microtag of the critical 32-byte data from system bus OR Interrupt vector with UE in the microtag of the non-critical (2nd) 32-byte data from system bus	IMU	Yes	Yes if no IVU	Received interrupt data	Disrupting trap & Interrupt taken if no IVU Interrupt dropped if IVU

8

Exceptions, Traps and Trap Types

The UltraSPARC IV+ processor implements all mandatory SPARC V9 exceptions as described in the *UltraSPARC III Cu Processor User's Manual*. In addition, the UltraSPARC IV+ processor implements the exceptions listed in TABLE 8-1, which are specific to the UltraSPARC IV+ processor.

Chapter Topics

8.1 Traps

The four main types of traps are discussed in detail in the following sections:

- Precise traps
- Deferred traps
- Disrupting traps
- Multiple traps

8.1.1 Precise Traps

A precise trap occurs before any program-visible state has been modified by the instruction to which the trap-saved program counter (TPC) points. When a precise trap occurs, several conditions are true:

- The program counter (PC) saved in TPC[TL] points to a valid instruction which will be executed by the program. The next program counter (nPC) saved in TNPC[TL] points to the instruction that will be executed following that one.
- All instructions issued before the one pointed to by the TPC have completed execution.
- Any instructions issued after the one pointed to by the TPC remain unexecuted.

The UltraSPARC IV+ processor generates three varieties of precise traps associated with data errors: *dcache_parity_error, icache_parity_error,* and *fast_ECC_error.*

A precise *dcache_parity_error* trap is generated for parity errors detected in the D-cache data, physical tag arrays, or P-cache data arrays as the result of instructions that perform a load.

A precise *icache_parity_error* trap is generated for a parity error detected in the I-cache data or physical tag arrays as the result of an I-fetch.

A precise *fast_ECC_error* trap also occurs when an uncorrectable L2-cache tag, a data error correcting code (ECC) error, or L3-cache data ECC error is detected as the result of a D-cache load miss, atomic instruction, or I-cache miss for I-fetch. All other single-bit data ECC errors are corrected by hardware and sent to the P-cache or W-cache. In any case, the L2-cache or L3-cache data is not corrected. Therefore, software support is required to correct the single-bit ECC errors.

A precise trap also occurs when an uncorrectable L2-cache data ECC error or L3-cache data ECC error is detected as the result of a D-cache load miss, atomic instruction, or I-cache miss. If the affected line is in the E or S MOESI state, software can recover from this problem in the precise trap handler. If the affected line is in the M or O states, a process or the whole domain must be terminated.

An I-cache or D-cache error can be detected for speculative instruction or data fetches. An L2-cache or L3-cache error can be detected when the instruction fetch is missed in the I-cache. Errors can also be detected when the I-cache autonomously fetches the second 32-byte line of a 64-byte L2-cache or L3-cache line. If an error detected in this way is on an instruction which is never executed, the precise trap associated with the error is never taken. However, L2-cache or L3-cache errors of this kind will be logged in the primary and secondary Asynchronous Fault Status Register (AFSR) and Asynchronous Fault Address Register (AFAR).

When a speculative request is canceled after an associated L2-cache or L3-cache error was generated, these errors will not be logged in the AFSR or AFAR. The speculative request, when canceled, will not load data into the D-cache and will never cause a precise trap.

8.1.2 Deferred Traps

Deferred traps may corrupt the processor state. Such traps lead to termination of the currently executing process or result in a system reset if the system state has been corrupted. Error logging information allows software to determine if the system state has been corrupted.

8.1.2.1 Error Barriers

A `MEMBAR #Sync` instruction provides an error barrier for deferred traps. It ensures that deferred traps from earlier accesses will not be reported after the `MEMBAR`. A `MEMBAR #Sync` should be used when context switching or any time the PSTATE.PRIV bit is changed to provide error isolation between processes.

`DONE` and `RETRY` instructions implicitly provide the same function as `MEMBAR #Sync` so that they act as error barriers. Errors reported as the result of fetching user code after a `DONE` or `RETRY` are always reported after the `DONE` or `RETRY`.

8.1.2.2 TPC, TNPC and Deferred Traps

After a deferred trap, the contents of TPC[TL] and TNPC[TL] are undefined except for the special peek sequence described below. Because they do not generally contain the oldest unexecuted instruction and its next PC, execution cannot normally be resumed from the point that the trap is taken. Instruction access errors are reported before executing the instruction that caused the error, but TPC[TL] does not necessarily point to the corrupted instruction.

8.1.2.3 Enabling Deferred Traps

When an error occurs which leads to a deferred trap, the trap will only be taken if the NCEEN bit is set in the Error Enable Register. See *Error Enable Register* on page 178. The deferred trap is an *instruction_access_error* if the error occurred as the result of an I-fetch. The deferred trap is a *data_access_error* if the error occurred as the result of a load, store, block load, block store, or atomic data access instruction.

The NCEEN bit should normally be set. If NCEEN is clear, the processor will compute using corrupt data and instructions when an uncorrectable error occurs.

The NCEEN bit also controls a number of disrupting traps associated with uncorrectable errors.

8.1.2.4 Errors Leading to Deferred Traps

Deferred traps are generated by the following:

- Uncorrectable L2-cache data ECC error as the result of a block load operation.
- Uncorrectable L3-cache data ECC error as the result of a block load operation.
- Uncorrectable system bus data ECC error in system bus read of memory or I/O for I-fetch, load, block load, or atomic operations. Uncorrectable ECC errors on L2-cache fills will be reported for any ECC error in the cache block, not just the referenced word (*UE*).
- Uncorrectable system bus MTag ECC error for any incoming data, but not including interrupt vectors. These errors also cause the processor to assert its ERROR output pin, so whether the trap is ever executed depends on system design.
- Bus error (*BERR*) as the result of a system bus read of memory or I/O for I-fetch, load, block load, or atomic operations.
- Timeout (*TO*) as the result of the following:
  - A system bus read of memory or I/O for I-fetch, load, block load, or atomic operations.
  - A system bus write of memory for block store and writeback operations.
  - A system bus write of I/O for store and block store operations.
  - An interrupt vector transmit operation.

8.1.2.5 Special Access Sequence for Recovering Deferred Traps

A special access sequence is required for intentional peeks and pokes to determine device presence and correctness and for I/O accesses from hardened drivers which must survive faults in an I/O device. This special access sequence allows the *data_access_error* trap handler to recover predictably, even though the trap is deferred. One possible sequence is described here.

```
The_peeker:
    <Set the peek_sequence_flag to indicate that a special peek sequence
is about to occur. This flag includes specifying the handler as
Special_peek_handler if a deferred TO/BERR does occur.>
    MEMBAR #Sync /* error barrier for deferred traps. [1] See explanation
below. */
    <Call routine to do the peek.>
    <Reset the peek_sequence_flag.>
    <Check success/failure indication from peek.>

Do_the_peek_routine:
    <Perform load. If a deferred trap occurs, execution will never resume
here.>
    MEMBAR #Sync /* error barrier. Make sure load takes. */
    <Indicate peek success.>
    <Return to peeker.>

Special_peek_handler:
    <Indicate peek failure.>
    <Return to peeker as if returning from Do_the_peek_routine.>

Deferred_trap_handler: (TL=1)
    <If the deferred trap handler sees a UE or TO or BERR and the
peek_sequence_flag is set, it resumes execution at the Special_peek_handler
by setting TPC and TNPC.>
```

FIGURE 8-1 Recovering Deferred Traps

Other than the load (or store in the case of poke), the *Do_the_peek_routine* should not have any other side effect because the deferred trap means that the code is not restartable. Execution after the trap is resumed in the *Special_peek_handler.*

The code in *Deferred_trap_handler* must be able to recognize any deferred traps that happen as a result of hitting the error barrier in *The_peeker* as not being from the peek operation. This situation is typically part of setting the *peek_sequence_flag*.

A `MEMBAR #Sync` is required as the first instruction in the trap table entry for *Deferred_trap_handler* to collect all potential trapping stores together to avoid a *RED_state* exception.

You can determine whether a deferred trap has come from a peek or poke sequence by using TPC or AFAR, as follows.

- If TPC is used, the locality of the trap to the *Do_the_peek_routine* must be assured using an error barrier as in the example above.
- If AFAR is used, the presence of orphaned errors resulting from the asynchronous activity of the instruction fetcher must be considered.
- If an orphaned error occurs, the source of the *TO* or *BERR* report cannot be determined from the AFAR.

Given the error barrier sequence above, it is reasonable to expect that the *TO* or *BERR* resulted from the peek or poke and proceed accordingly. To reduce the likelihood of this event, orphaned errors can be cleaned at point [1] shown in FIGURE 8-1. The source of the *TO* or *BERR* can be confirmed by retrying the peek or poke:

- If the *TO* or *BERR* happens again, the system can continue with the normal peek or poke failure case.
- If the *TO* or *BERR* does not happen, the system must panic.

The peek access should be preceded and followed by MEMBAR #Sync instructions. The state of the destination register of the access may be corrupted; however, other states will not be affected. If TPC is pointing to the MEMBAR #Sync following the access, then the *data_access_error* trap handler knows that a recoverable error has occurred and resumes execution after setting a status flag. The trap handler will have to set TNPC to TPC + 4 before resuming because the contents of TNPC are otherwise undefined.

8.1.2.6 Deferred Trap Handler Functionality

The following is a possible sequence for handling unexpected deferred errors within the trap handler:

1. Log the error(s).
2. Reset the error logging bits in AFSR1.
3. Perform a MEMBAR #Sync to complete internal ASI stores.
4. Panic if AFSR1.PRIV is set and not performing an intentional peek/poke; otherwise try to continue.
5. Invalidate the D-cache by writing each line of the D-cache with ASI_DCACHE_TAG. This may not be required for *instruction_access_error* events, but it is the simplest way to invalidate the D-cache in all cases.
6. Abort the current process.
7. For user process *UE* errors in a conventional UNIX® system: once all processes using the physical page in error have been signaled and terminated as part of the normal page recycling mechanism, clear the *UE* from main memory by writing the page zero routine to use block store instructions. The trap handler does not usually have to clear out a *UE* in main memory.
8. Resume execution.

8.1.3 Disrupting Traps

Disrupting traps, like deferred traps, may cause program-visible state change. However, disrupting traps are similar to precise traps in the following ways:

1. The PC saved in TPC[TL] points to a valid instruction which will be executed by the program, and the nPC saved in TNPC[TL] points to the instruction that will be executed after that one.
2. All instructions issued before the one pointed to by the TPC have completed execution.
3. Any instructions issued after the one pointed to by the TPC remain unexecuted.

Errors which lead to disrupting traps are the following:

- *HW_corrected* ECC errors in the L2-cache data (AFSR.EDC, CPC, WDC) and in the L2-cache tag (AFSR.THCE).
- *HW_corrected* ECC errors in L3-cache data (AFSR_EXT.L3_EDC, L3_CPC, L3_WDC) and in L3-cache tag (AFSR_EXT.L3_THCE), correctable MTag error (AFSR.EMC), correctable interrupt vector (AFSR.IVC), correctable MTag error in interrupt vector (AFSR.IMC).
- Uncorrectable L2-cache data errors as the result of store operation, writeback and copyout operations (AFSR.EDU, WDU, CPU).
- Uncorrectable L2-cache tag errors (AFSR.TUE, TUE_SH).

- Uncorrectable L3-cache errors as the result of prefetch, store, writeback, and copyout operations (AFSR_EXT.L3_EDU, L3_WDU, L3_CPU).
- Uncorrectable interrupt vector, prefetch queue and store queue read system bus errors (AFSR.IVU, AFSR.DUE, AFSR.DTO, AFSR.DBERR).
- Uncorrectable MTag error in interrupt vector (AFSR.IMU).

The disrupting trap handler should log the error. No special operations, such as cache flushing, are required for correctness after a disrupting trap. However, for many errors, it is appropriate to correct the data which produced the original error so that a later access to the same data does not produce the same trap again. For uncorrectable errors, it is the responsibility of the software to determine the recovery mechanism with the minimum system impact.

HW_corrected ECC errors result from detection of a single-bit *ECC* error as the result of a system bus read or L2-cache or L3-cache access. *HW_corrected* errors are logged in the Asynchronous Fault Status Register and, except for interrupt vector fetches, in the Asynchronous Fault Address Register. If the *Correctable_Error* (*CEEN*) trap is enabled in the Error Enable Register, an *ECC_error* trap is generated.

L2-cache data *ECC* errors are discussed in *L2-cache Data ECC Errors* on page 159. Uncorrectable L2-cache data *ECC* errors as the result of a read to satisfy store queue exclusive request, prefetch requests, writeback, or copyout require only logging on the processor not using the affected data. Consequently, a disrupting *ECC_error* trap is taken instead of a deferred trap. This avoids panics when the system displaces corrupted user data from the cache.

L3-cache data *ECC* errors are discussed in *L3-cache Data ECC Errors* on page 165. Uncorrectable L3-cache data *ECC* errors as the result of a read to satisfy a store queue exclusive request, prefetch requests, writeback, or copyout require only logging on the processor not using the affected data. Consequently, a disrupting *ECC_error* trap is taken instead of a deferred trap thus avoiding panics when the system displaces corrupted user data from the cache.

Uncorrectable errors causing disrupting traps need no immediate action to guarantee data correctness. However, it is likely that an event signaled by AFSR.DUE, AFSR.DTO, AFSR.DBERR, AFSR.IVU, or AFSR.IMU will be followed later by some unrecoverable event that requires process death. The appropriate action by the handler for the disrupting trap is to log the event and return. A later event will cause the right system recovery action to be taken.

The *ECC_error* disrupting trap is enabled by PSTATE.IE. PSTATE.PIL has no effect on *ECC_error* traps.

Note – To prevent multiple traps from the same error, software should not re-enable interrupts until after the disrupting error status bit in AFSR1 is cleared.

8.1.4 Multiple Traps

See *When Are Traps Taken?* on page 215 for a discussion on what happens when multiple traps occur at once.

8.2 Exceptions Specific to the UltraSPARC IV+ Processor

Note – Floating-Point Control: Two state bits, PSTATE.PEF and FPRS.FEF, in the SPARC V9 architecture provide the means to disable direct floating-point execution. If either field is set to 0, an *fp_disabled* exception is taken when any floating-point instruction is encountered.
Graphics instructions that use the floating-point register file and instructions that read or update the Graphic Status Register (GSR) are treated as floating-point instructions. They cause an *fp_disabled* exception if either PSTATE.PEF or FPRS.FEF is zero. See *Graphics Status Register (GSR) (ASR 19)* the *UltraSPARC III Cu Processor User's Manual* for more information.

The exceptions specific to the UltraSPARC IV+ processor are described in TABLE 8-1.

TABLE 8-1 Exceptions Specific to the UltraSPARC IV+ Processor

Exception or Interrupt Request	Description	TT	Global Register Set	Priority
fast_ECC_error	Taken on software-correctable L2-cache or L3-cache data ECC errors, uncorrectable L2-cache tag, data ECC errors, or L3-cache data ECC errors detected as a result of a D-cache load miss, atomic instruction, or I-cache miss for instruction fetch. The trap handler is required to flush the cache line containing the error from the D-cache, L2-cache, and L3-cache because incorrect data would have already been written into the D-cache. The UltraSPARC IV+ processor hardware will automatically correct single-bit ECC errors on the L2-cache writeback and L3-cache writeback when the trap handler performs the L2-cache flush and L3-cache flush. After the caches are flushed, the instruction that encountered the error should be retried; the corrected data will then be brought back in from memory and reinstalled in the D-cache and L2-cache. On fast_ECC_error detection during D-cache load miss fill, D-cache installs the uncorrected data. Because the fast_ECC_error trap is precise, hardware can rely on software to help clean up the bad data. In case of I-cache miss, however, bad data never gets installed in the I-cache. Unlike D-cache and I-cache parity error, a D-cache/I-cache miss request that returns with fast_ECC_error will not automatically turn off D-cache and/or I-cache. Because I-cache is not filled on error detection, the trap code can safely run off I-cache, where the first step is to have software turn off D-cache and/or I-cache as needed. See *Software Correctable L2-cache ECC Error Recovery Actions* on page 160 and *Software Correctable L3-cache ECC Error Recovery Actions* on page 166 for details about fast_ECC_error trap handler actions.	070_{16}	AG	2
dcache_parity_error	Taken on parity error detected when a load instruction gets its data from the D-cache or the P-cache. See *D-cache Error Recovery Actions* on page 151 for details about dcache_parity_error trap handler actions.	071_{16}	AG	2
icache_parity_error	Taken on parity error detected when instructions are fetched from the I-cache. See *I-cache Error Recovery Actions* on page 146 for details about icache_parity_error trap handler actions.	072_{16}	AG	2

The exceptions listed in TABLE 8-1 result in precise traps. In the UltraSPARC IV+ processor, all traps are precise except for the deferred traps described in *Deferred Traps* on page 254 and the disrupting traps described in *Disrupting Traps* on page 257.

8.3 Trap Priority

To ensure the appropriate processing of trap information a priority system is employed. Arriving traps are processed by the hardware depending on their priority and the length of time that the trap has been waiting for processing,

8.3.1 Precise Trap Priority

All traps with priority 2 are precise traps. Miss/error traps with priority 2 that arrive at the same time are processed by hardware according to their age or program order. The oldest instruction with an miss/error will get the trap. However, there are two cases where the same instruction generates multiple traps:

Case 1: Singular trap type with highest priority.
The processing order is determined by the priority number: lowest number that has the highest priority is processed first.

Case 2: Multiple traps having same priority.
For trap priority 2, the only possible combination is simultaneous traps due to I-cache parity error and I-TLB miss. In this case the hardware processing order is:
icache_parity_error > fast_instruction_access_MMU_miss.

All other priority 2 traps have staggered arrivals and therefore will not result in simultaneous traps. D-cache access is further down the pipeline after instruction fetch from I-cache. Thus D-cache parity error on a load instruction (if any) will be detected after I-cache parity error (if any) and I-TLB miss (if any). The other priority 2 trap, *fast_ECC_error,* can only be caused by an I-cache miss or D-cache load miss; therefore it arrives even later.

To summarize, precise traps are processed in the following order:
program order > trap priority number > hardware implementation order

8.4 I-cache Parity Error Trap

An I-cache physical tag or data parity error results in an *icache_parity_error* precise trap. Hardware does not provide any information as to whether the *icache_parity_error* trap occurred due to a tag or a data parity error.

8.4.1 Hardware Action on Trap for I-cache Data Parity Error

Parity error detected during I-cache instruction fetch will take an *icache_parity_error* trap, TT = 0x072, priority = 2, globals = AG.

Note – I-cache data parity error detection and reporting is suppressed if DCR.IPE = 0, or cache line is invalid.

Precise traps are used to report an I-cache data parity error.

I-cache and D-cache are immediately disabled by hardware when a parity error is detected by clearing DCUCR.IC and DCUCR.DC bits.

In the trap handler, software should invalidate the entire I-cache, D-cache, and P-cache. See *I-cache Error Recovery Actions* on page 146 for details.

No special parity error status bit or address information will be logged in hardware. Because *icache_parity_error* trap is precise, software has the option to log the parity error information on its own.

I-cache data parity error is determined based on per-instruction fetch group granularity. Unused or annulled instructions are not masked out during the parity check.

If an I-cache data parity error is detected while in another event (I-cache miss or I-TLB miss), the behavior is described in TABLE 8-2.

TABLE 8-2 I-cache Data Parity Error Behavior on Instruction Fetch

Canceled/Retried Instruction	Reporting if Parity Error	icache_parity_error Trap Taken?
I-cache miss due to invalid line (valid=0), microtag miss or tag mismatch	icache_parity_error detection is suppressed.	No
I-TLB miss	icache_parity_error has priority over fast_instruction_access_MMU_miss.	Yes

8.4.2 Hardware Action on Trap for I-cache Physical Tag Parity Error

Parity error is reported the same way (same trap type and precise-trap timing) as in I-cache data array parity error. See *Hardware Action on Trap for I-cache Data Parity Error* on page 260.

Note – I-cache physical tag parity error detection is suppressed if DCR.IPE = 0, or cache line is invalid.

I-cache physical tag parity error is determined based on per instruction fetch group granularity. Unused or annulled instructions are not masked out during the parity check.

If an I-cache physical tag parity error is detected while in another event (I-cache miss or I-TLB miss), the behavior is described in TABLE 8-3.

TABLE 8-3 I-cache Physical Tag Parity Error Behavior on Instruction Fetch

Canceled/Retried Instruction	Reporting if Parity Error	icache_parity_error Trap Taken?
I-cache miss due to invalid line (valid=0), microtag miss or tag mismatch	icache_parity_error detection is suppressed.	No
I-TLB miss	icache_parity_error has priority over fast_instruction_access_MMU_miss.	Yes

8.4.3 Hardware Action on Trap for I-cache Snoop Tag Parity Error

Parity error detected in I-cache snoop tag will not cause a trap, nor will it be reported/logged.

An invalidate transaction snoops all four ways of the I-cache in parallel.

On an invalidate transaction, each entry over the four ways that has a parity error will be invalidated in addition to those that have a true match to the invalidate address. Entries which do not possess parity errors or do not match the invalidate address are not affected.

Note – I-cache snoop tag parity error detection is suppressed if DCR.IPE = 0, or cache line is invalid.

8.5 D-cache Parity Error Trap

A D-cache physical tag or data parity error results in a *dcache_parity_error* precise trap. Hardware does not provide any information as to whether the *dcache_parity_error* trap occurred due to a tag or a data parity error.

8.5.1 Hardware Action on Trap for D-cache Data Parity Error

Parity error detected during D-cache load operation will take a *dcache_parity_error* trap, TT = 0x071, priority = 2, globals = AG.

Note – D-cache data parity error reporting is suppressed if DCR.DPE = 0. D-cache data parity error checking ignores cache line's valid bit, microtag hit/miss, and physical tag hit/miss. Parity error checking is only done for load instructions, but not for store instructions. On store update to D-cache, a parity bit is generated for every byte of store data written to D-cache.

Precise traps are used to report a D-cache data parity error.

On detection of a D-cache data parity error, hardware turns off the D-cache and I-cache by clearing DCUCR.DC and DCUCR.IC bits.

In the trap handler, software should invalidate the entire I-cache, D-cache, and P-cache. See *D-cache Error Recovery Actions* on page 151 for details.

No special parity error status bit or address information will be logged in hardware. Because *dcache_parity_error* trap is precise, software has the option to log the parity error information on its own.

There are some cases where a speculative access to D-cache belongs to a canceled or retried instruction. Also the access could be from a special load instruction. The error behaviors in such cases are described in TABLE 8-4. The general explanation is that in hardware, trap information must be attached to an instruction. Thus if the instruction is canceled (for example, in a wrong branch path), no trap is taken.

TABLE 8-4 D-cache Data Parity Error Behavior on Canceled/Retried/Special Load

Canceled/Retried/Special Load	Reporting if Parity Error	*dcache_parity_error* Trap Taken?
D-cache miss (real miss without tag parity error) due to invalid line or tag mismatch	dcache_parity_error is not suppressed.	Yes
D-TLB miss	dcache_parity_error has priority over fast_data_access_MMU_miss.	Yes
Preceded by trap or retry of an older instruction (for example, following wrong path of a mispredicted branch instruction)	Dropped: suppressed by age at the trap logic.	No
Annulled in delay slot of a branch	Dropped: an annulled instruction never enters an execution pipeline thus no D-cache access.	No
Block load, internal ASI load, atomic	Suppressed.	No
Integer LDD (needs two load accesses to D-cache)	The hardware checks the full eight bytes on the first access and will report any errors at that time. If an error occurs during the second load access, the error will not be detected.	Yes (On first load access)
Quad load	Quad load accesses the D-cache but does not get the data from the D-cache. A quad load always forces a D-cache miss so that the data is loaded from the L2/L3-cache or the memory. The D-cache access may cause a parity error to be observed and reported, even though the corresponding data will not be used.	Yes (Can detect an error, but the error will correspond to a line not actually being used.)

8.5.2 Hardware Action on Trap for D-cache Physical Tag Parity Error

Parity error is reported the same way (same trap type and precise-trap timing) as in D-cache data array parity error. See *Hardware Action on Trap for D-cache Data Parity Error* on page 262.

Note – Unlike in D-cache data array, parity error checking in D-cache physical tag is further qualified with valid bit and microtag hit but ignores physical tag hit/miss.

On store, atomic, or block store instruction issue, D-cache physical tag is also read to determine whether it is a D-cache store hit/miss. Consequently, the D-cache physical tag parity error checking is also done on store or block store.

There are some cases where a speculative access to D-cache belongs to a canceled or retried instruction. Also, the access could be from a special load instruction. The error behaviors on D-cache physical tag in such cases are described in TABLE 8-5.

TABLE 8-5 D-cache Physical Tag Parity Error Behavior on Canceled/Retried/Special Load

Canceled/Retried/Special Load	Reporting if Parity Error	*dcache_parity_error* Trap Taken?
D-cache miss due to invalid line (valid=0)	dcache_parity_error detection is suppressed.	No
D-cache miss due to microtag miss	dcache_parity_error detection is suppressed for physical tag array.	No
D-TLB miss	dcache_parity_error has priority over fast_data_access_MMU_miss.	Yes
Preceded by trap or retry of an older instruction (for example, following wrong path of a mispredicted branch instruction)	Dropped: suppressed by age at the trap logic.	No
Microtag hit but D-cache miss due to Physical Tag parity error	dcache_parity_error is reported.	Yes
Annulled in delay slot of a branch	Dropped: annulled instructions never enter an execution pipeline thus no D-cache access.	No
Block load, internal ASI load	Suppressed.	No

8.5.3 Hardware Action on Trap for D-cache Snoop Tag Parity Error

Parity error detected in D-cache snoop tag will *not* cause a trap nor will it be reported/logged.

An invalidate transaction snoops all four ways of the D-cache in parallel.

On an invalidate transaction, each entry over the four ways that has a parity error will be invalidated in addition to those that have a true match to the invalidate address. Entries which do not possess parity errors or do not match the invalidate address are not affected.

Note – D-cache snoop tag parity error detection is suppressed if DCR.DPE = 0, or cache line is invalid.

8.6 P-cache Parity Error Trap

A P-cache data parity error results in a *dcache_parity_error* precise trap. Hardware does not provide any information as to whether the *dcache_parity_error* trap occurred due to a D-cache (tag/data) error or a P-cache (data) error.

8.6.1 Hardware Action on Trap for P-cache Data Parity Error

For parity error detected for floating point load through P-cache, the P-cache data parity error is reported the same way (same trap type and precise-trap timing) as in D-cache data array parity error. See *Hardware Action on Trap for D-cache Data Parity Error* on page 262.

Note – P-cache data parity error detection is suppressed if DCR.PPE = 0.

The error behavior on P-cache data parity is described in TABLE 8-6.

TABLE 8-6 P-cache Data Parity

Instructions	Reporting if Parity Error	*dcache_parity_error* Trap Taken?
FP load miss P-cache	dcache_parity_error detection is suppressed.	No
FP load hit P-cache	dcache_parity_error detected.	Yes
Software prefetch hit P-cache	dcache_parity_error detected.	No
Internal ASI load	dcache_parity_error detected.	No

9

Registers

For general register information see the *UltraSPARC III Cu Processor User Manual*. The registers specific to the UltraSPARC IV+ processor are discussed in this chapter.

Chapter Topics

Note – In the UltraSPARC IV+ processor, all user visible registers are private registers. This includes all General-Purpose Registers (GPRs) and Floating-Point Registers (FPRs) as well as all Ancillary State Registers (ASRs) and privileged registers. Some ASI (Address Space Identifier) registers are private, while some ASI registers are shared by both logical processors. *ASI Assignments* on page 280 lists all the ASI registers in the UltraSPARC IV+ processor and specifies if each register is a private or a shared register.

9.1 Floating-Point State Register (FSR)

9.1.1 FSR_nonstandard_fp (NS)

If a floating-point operation generates a subnormal value on the UltraSPARC IV+ processor and FSR.NS = 1, the subnormal value is replaced by a floating-point zero value of the same sign. In the UltraSPARC IV+ processor, this replacement is performed in hardware for all cases. See *Subnormal Handling Override* on page 140 for details.

Note – Earlier processors in the UltraSPARC III processor family performed this replacement in hardware for some cases. While for other cases, the processor generates an *fp_exception_other* exception with FSR.FTT = 2 (*unfinished_FPop*) expecting that the trap handler will perform the replacement.

9.1.2 FSR_floating_point_trap_type (FTT)

The UltraSPARC IV+ processor triggers *fp_exception_other* with trap type *unfinished_FPop* under the conditions described in *Response to Subnormal Operands* on page 138.

9.2 PSTATE Register

The UltraSPARC IV+ processor supports two additional sets (privileged only) of eight 64-bit global registers: the interrupt globals and MMU globals. These additional registers are called the *trap globals*. Two 1-bit fields, PSTATE.IG and PSTATE.MG, have been added to the PSTATE register to select which set of global registers to use.

While PSTATE.AM = 1:

- The UltraSPARC IV+ processor writes the full 64-bit program counter (PC) value to the destination register of a `CALL`, `JMPL`, or `RDPC` instruction.
- When a trap occurs the UltraSPARC IV+ processor writes the truncated (i.e., high-order 32 bits set to 0) PC and nPC value to TPC[TL] and TNPC[TL].
- The UltraSPARC IV+ processor truncates (i.e., high-order 32 bits set to 0) the branch target address (sent to nPC) of a `CALL/JMPL` instruction as well as the value loaded to PC/nPC from TPC[TL]/TNPC[TL] on returning from a trap using `DONE/RETRY`.
- When an exception occurs, the UltraSPARC IV+ processor writes the full 64-bit address to the D-SFAR.

Note – Exiting RED_state by writing 0 to PSTATE.RED in the delay slot of a `JMPL` instruction is not recommended. A noncacheable instruction prefetch can be made to the `JMPL` target, which can be in a cacheable memory area, which may result in a bus error on some systems and cause an *instruction_access_error* trap. Programmers can mask the trap by setting the NCEEN bit in the L3-cache Error Enable Register to 0, but this solution masks all non-correctable error checking. Exiting RED_state with `DONE` or `RETRY` avoids the problem.

9.3 Ancillary State Registers (ASRs)

9.3.1 Performance Control Register (PCR) (ASR 16)

Bits[47:32], [26:17], and bit[3] of the PCR are unused in the UltraSPARC IV+ processor. These bits read as zeroes and writes to these bits are ignored.

9.3.2 Performance Instrumentation Counter (PIC) Register (ASR 17)

The performance Instrumentation counters, previously known as PIC1 and PIC0 in earlier processor documentation are now known as PICU and PICL.

9.3.3 Dispatch Control Register (DCR) (ASR 18)

The Dispatch Control Register is accessed through ASR 18. This register should only be accessed in privileged mode. Non-privileged accesses to this register causes a *privileged_opcode* trap. The Dispatch Control Register is described in TABLE 9-1.

Note – The bit fields IPE, DPE, ITPE, DTPE, and PPE are 0 by default (disabled) after power-on or system reset.

TABLE 9-1 Dispatch Control Register ***(1 of 2)***

Bit	Field	Description
[63:19]	*Reserved*	Reserved for future implementation.
[18]	PPE[1]	Prefetch Cache Parity Error Enable. If cleared, no parity checking is done at the Prefetch Cache SRAM arrays (data, physical tag, and virtual tag arrays).
[17]	DTPE[2]	D-TLB Parity Error Enable. If cleared, no parity checking is done at the D-TLB arrays (data and tag arrays).
[16]	ITPE[3]	I-TLB Parity Error Enable. If cleared, no parity checking is done at the I-TLB arrays (data, tag, and content-addressable-memory arrays).

TABLE 9-1 Dispatch Control Register *(2 of 2)*

<table>
<tr><th>Bit</th><th>Field</th><th>Description</th></tr>
<tr><td>[15]</td><td>JPE[4]</td><td>Jump Prediction Enable. If set, the BTB (Branch Target Buffer) is used to predict target address of JMPL instructions which do not match the format of the RET/RETL synthetic instructions.</td></tr>
<tr><td>[14:13]</td><td>BPM</td><td>Branch Predictor Mode. Branch Predictor Mode can be configured to use a separate history register when operating in privileged mode. It can also be configured not to use the history register when indexing the Branch Prediction table, using PC-based indexing instead.
<table>
<tr><th>DCR [14:13]</th><th>Branch Predictor Mode</th></tr>
<tr><td>00</td><td>One history register, gshare mode (US III mode)</td></tr>
<tr><td>01</td><td>Two history registers, both gshare mode</td></tr>
<tr><td>10</td><td>PC-based indexing</td></tr>
<tr><td>11</td><td>Two history registers, normal uses gshare, privileged uses PC-based</td></tr>
</table></td></tr>
<tr><td>[12]</td><td>DPE[5]</td><td>Data Cache Parity Error Enable. If cleared, no parity checking is done at the Data Cache SRAM arrays (data, physical tag, and snoop tag arrays).</td></tr>
<tr><td>[11:6]</td><td>OBS</td><td>Observability Bus. Bits [11:6] can be programmed to select the set of signals to be observed at obsdata[9:0]. See TABLE 9-2. for bit settings.</td></tr>
<tr><td>[5]</td><td>BPE</td><td>See the UltraSPARC III Cu Processor User's Manual for a description of the BPE.</td></tr>
<tr><td>[4]</td><td>RPE</td><td>See the UltraSPARC III Cu Processor User's Manual for a description of the RPE.</td></tr>
<tr><td>[3]</td><td>SI</td><td>Single Issue Disable. See the UltraSPARC III Cu Processor User's Manual and has no additional side effects.</td></tr>
<tr><td>[2]</td><td>IPE[6]</td><td>Instruction Cache Parity Error Enable. If cleared, no parity checking will be done at the Instruction Cache and IPB SRAM arrays (data, physical tag, snoop tag, and IPB arrays).</td></tr>
<tr><td>[1]</td><td>IFPOE[7]</td><td>Interrupt Floating-Point Operation Enable. This bit enables system software to take interrupts on FP instructions.</td></tr>
<tr><td>[0]</td><td>MS</td><td>Multiscalar Dispatch Enable. See the UltraSPARC III Cu Processor User's Manual for a description of the MS bit.</td></tr>
</table>

1.Implies no *dcache_parity_error* trap (TT 0x071) will ever be generated. However, parity bits are still generated automatically and correctly by hardware.

2.Implies no *data_access_exception* trap (TT 0x030) will ever be generated. However, parity bits are still generated automatically and correctly by hardware.

3.Implies no *instruction_access_exception* trap (TT 0x008) will ever be generated. However, parity bits are still generated automatically and correctly by hardware.

4.This 32-entry direct-mapped BTB is updated when a non-`RETURN JMPL` is mispredicted. If a `JMPL` is not a `RET/RETL`, and jump prediction is enabled in the DCR, and there is not currently a valid entry in the prepare-to-jump register, its target will be predicted by reading the BTB using VA[9:5].

In addition to BTB, the target of indirect branches (those which don't contain the target address as part of the instruction, but get the target from a register, such as `JMPL` or `RETURN`) will be predicted using one of two structures. Note: if PSTATE.RED=1, the predicted target is forced to a known safe address instead of using the standard methods of prediction.

The Return Address Stack (RAS) - This 8-entry stack contains the predicted address for `RET/RETL/RETURN` instructions. When a `CALL` is executed, the "PC + 8" of the `CALL` is pushed onto this stack. If return prediction is enabled in the DCR, a return instruction will use the top entry of the RAS as its predicted target. A return instruction will also decrement the stack pointer.

The Branch Target Buffer (BTB) - The prepare-to-jump register will be used to predict one non-`RETURN JMPL` after it is written if `JPE` is enabled in the DCR. After one prediction, it will be invalidated until the next write. This register is written as a side-effect of a `MOVcc` with %o7 (%r15) as the destination register.

5.Implies no *dcache_parity_error* trap (TT 0x071) will ever be generated. However, parity bits are still generated automatically and correctly by hardware.

6.Implies no *icache_parity_error* trap (TT 0x072) will ever be generated. However, parity bits are still generated automatically and correctly by hardware.

7.This enable bit is cleared by hardware at power-on. System software must set the bit as needed. When this bit is enabled, the UltraSPARC IV+ processor forces an *fp_disabled* trap when an interrupt occurs on Floating-point only code.

The trap handler is then responsible for checking whether the Floating-point is indeed disabled. If it is not, the trap handler then enables interrupts to take the pending interrupt. This behavior deviates from the SPARC V9 trap priorities in that interrupts are of lower priorities than the other two types of Floating-point Exceptions: (*fp_exception_ieee_754, fp_exception_other*).

This mechanism is triggered for a Floating-point instruction only if none of the approximately 12 preceding instructions across the two integer, load/store, and branch pipelines are valid, under the assumption that they are better suited to take the interrupt (only one trap entry/exit).

Upon entry, the handler must check both TSTATE.PEF and FPRS.FEF bits. If TSTATE.PEF = 1 and FPRF.FEF = 1, the handler has been entered because of an interrupt, either *interrupt_vector* or *interrupt_level*. In such a case:

The *fp_disabled* handler should enable interrupts (that is, set PSTATE.IE = 1), then issue an integer instruction (for example, add %g0,%g0,%g0). An interrupt is triggered on this instruction.
The UltraSPARC IV+ processor then enters the appropriate interrupt handler (PSTATE.IE is turned off here) for the type of interrupt.

At the end of the handler, the interrupted instruction is RETRY'd after returning from the interrupt. The add %g0,%g0,%g0 is RETRY'd.

The *fp_disabled* handler then returns to the original process with a RETRY.

The "interrupted" FPop is then retried (taking an *fp_exception_ieee_754* or *fp_exception_other* at this time if needed).

TABLE 9-2 shows the mapping between the settings on bits [11:6] of the DCR and the values seen on the observability bus.

TABLE 9-2 Signals Observed at obsdata[9:0] for Settings on Bits[11:6] of the DCR *(1 of 2)*

DCR Bits [11:6]	Signal Source	obsdata [9]	obsdata [8]	obsdata [7]	obsdata [6]	obsdata [5]	obsdata [4]	obsdata [3]	obsdata [2]	obsdata [1]
101xxx	ECC* (Default)	0	l3t_cor	l2t_cor	l3d_cor	l3d_uncor	l2d_cor	l2d_uncor	sys_cor	sys_uncor
100010	Clk grid	l2clk/4	l2clk/16	l2clk	l1clk/4	l1clk/16	l1clk	l2clk/8	l1clk/8	bootclk
110110	LP 0 IU*	a0_valid	a1_valid	ms_valid	br_valid	fa_valid	fm_valid	ins_comp	mispred	recirc
110100	LP 1 IU*	a0_valid	a1_valid	ms_valid	br_valid	fa_valid	fm_valid	ins_comp	mispred	recirc
110101	2 LP IU*	LP0 trap_tak	LP0 ins_disp	LP0 ins_comp	LP0 recirc	LP1 trap_tak	LP1 ins_disp	LP1 ins_comp	LP1 recirc	master
111000	IOT impctl[2]	delta (up)	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]
111001	IOT impctl [3]	delta (down)	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]
111011	IOL impctl[0]	delta (up)	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]
111010	IOL impctl[1]	delta (down)	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]
111110	IOR impctl[4]	delta (up)	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]
111111	IOR impctl[5]	delta (down)	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]
111101	IOB impctl[6]	delta (up)	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]	Zcu[3]	Zcu[2]	Zcu[1]	Zcu[0]
111100	IOB impctl[7]	delta (down)	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]	Zcd[3]	Zcd[2]	Zcd[1]	Zcd[0]

TABLE 9-2 Signals Observed at obsdata[9:0] for Settings on Bits[11:6] of the DCR *(2 of 2)*

DCR Bits [11:6]	Signal Source	obsdata [9]	obsdata [8]	obsdata [7]	obsdata [6]	obsdata [5]	obsdata [4]	obsdata [3]	obsdata [2]	obsdata [1]
110000	L2-cache pipeline signal	L2-cache_ hit	l2t_LP_id	l2t_valid	any_retr_ wr5	l2t_src[4]	l2t_src[3]	l2t_src[2]	l2t_src[1]	l2t_src[0]
110001	L3-cache pipeline signal	L3-cache_ hit	l3t_LP_id	l3t_valid	any_retr_ wr5	l3t_src[4]	l3t_src[3]	l3t_src[2]	l3t_src[1]	l3t_src[0]
100000	L2-cache /L3-cache livelock signal	livelock_ fix	queue_ under_ watch_[3]	queue_ under_ watch_[2]	queue_ under_ watch_[1]	queue_ under_ watch_[0]	stop_w-cache0	stop_w-cache1	l2hit_fill	l2miss_fi ll
100001	LP0 L1-cache signal	0	l2_to_ic_ fill	ipb_to_ic_ fill	ipb_fill	D-cache_fill	D-cache_sto re	P-cache_fill	W-cache_fill	W-cache_st ore
100100	LP1 L1-cache signal	0	l2_to_ic _fill	ipb_to_ic_ fill	ipb_fill	D-cache_fill	D-cache_sto re	P-cache_fill	W-cache_fill	W-cache_st ore
100101	Empty	0	0	0	0	0	0	0	0	0
100110	Empty	0	0	0	0	0	0	0	0	0

* ECC = Error Correcting Code; IU = Integer Unit

A few requirements and recommendations for using the DCR to control the observability bus are outlined below.

- Use only one or the other mode of control for the observability bus, that is, control either by pulses at obsdata[0] or by programming of the DCR bits.
- As long as the por_n pin is asserted, the state of obsdata[9:0] will always be 0. Once the device has been reset, the default state becomes visible on the bus. Note that the DCR OBS control bits are reset to all 0's on a software POR trap. Until the DCR bit 11 is programmed to 1, the obsdata[0] mode of control has precedence.
- There is a latency of approximately 5 cycles between writing the DCR and the signals corresponding to the setting appearing at obsdata[9:0].
- In the UltraSPARC IV+ processor, there are two sets of DCR[11:6], one for each logical processor. Either set can be used to control the state of obsdata[9:0]. When one set is programmed with a value, the other set must be programmed to 0. When a logical processor is disabled or parked, its DCR[11:6] should be programmed to 0.

Note – Every valid setting of the DCR OBS control bits, bit 11 must be set to 1.

9.4 Registers Referenced Through ASIs

9.4.1 Data Cache Unit Control Register (DCUCR)

ASI 45_{16} (`ASI_DCU_CONTROL_REGISTER`), VA = 0_{16}

The Data Cache Unit Control Register contains fields that control several memory-related hardware functions. These functions include instruction, prefetch, write and data caches, MMUs, and watchpoint setting. Most of the DCUCR's functions are described in the *UltraSPARC III Cu Processor User's Manual*; details specific to the UltraSPARC IV+ processor are described in this section.

After a power-on reset (POR), all fields of the DCUCR are set to 0. After a WDR, XIR, or SIR reset, all fields of the DCUCR except WE are set to 0 and WE is left unchanged.

The Data Cache Unit Control Register, as implemented in the UltraSPARC IV+ processor, is described in TABLE 9-3. In the table, bits are grouped by function rather than by strict bit sequence.

TABLE 9-3 DCU Control Register Access Data Format (ASI 45_{16}) *(1 of 2)*

Bits	Field	Description	R/W
[63:62]	*Reserved*	Reserved for DFT. (Design for Test).	RW
[61:56]	*Reserved*	Reserved for future implementation.	
[55]	WCE	Write-cache Coalescing Enable. If cleared, coalescing of store data for cacheable stores is disabled. The default setting for this bit is "1" (i.e., coalescing is enabled).	RW
[54]	PCM	P-cache Mode Rese to 0t: When weak prefetches (function code is 0x0, 0x1, 0x2, or 0x3) miss TLB, TLB miss - trap is not taken. When strong prefetches (function code is 0x14, 0x15, 0x16, or 0x17) miss TLB, TLB miss - trap is taken Weak prefetches are not recirculated if the prefetch queue is full. Strong prefetches are recirculated if the prefetch queue is full. Set to 1: When weak prefetches miss TLB, TLB miss - trap is not taken. When strong prefetches miss TLB, TLB miss - trap is taken. Weak prefetches are recirculated if the prefetch queue is full. Strong prefetches are recirculated if the prefetch queue is full.	RW
[53:52]	IPS	Programmable Instruction Prefetch Stride 00 - no prefetch 01 - 64 Bytes 10 - 128 Bytes 11 - 192 Bytes	RW
[51:50]	PPS	Programmable P-cache Prefetch Stride 00 - no prefetch 01 - 64 Bytes 10 - 128 Bytes 11 - 192 Bytes	RW
[49]	CP	The UltraSPARC IV+ processor implements this cacheability bit as described in the *UltraSPARC III Cu Processor User's Manual*.	RW
[48]	CV	The UltraSPARC IV+ processor implements this cacheability bit as described in the *UltraSPARC III Cu Processor User's Manual*	RW
[47]	ME	Non-cacheable Store Merging Enable. If cleared, no merging (or coalescing) of noncacheable, non-side-effect store data occurs. Each non-cacheable store generates a system bus (Fireplane) transaction.	RW
[46]	RE	RAW Bypass Enable. If cleared, no bypassing of data from the store queue to a dependent load instruction occurs. All load instructions will have their RAW predict field cleared.	RW
[45]	PE	P-cache Enable. If cleared, all references to the P-cache are handled as P-cache misses.	RW
[44]	HPE	P-cache hardware Prefetch Enable. If cleared, the P-cache does not generate any hardware prefetch requests to the L2-cache. Software prefetch instructions are not affected by this bit.	RW

TABLE 9-3 DCU Control Register Access Data Format (ASI 45_{16}) *(2 of 2)*

Bits	Field	Description	R/W
[43]	SPE	Software Prefetch Enable. If cleared, software prefetch instructions do not generate a request to the L2-cache/L3-cache or the system interface. They will continue to be issued to the pipeline, where they will be treated as NOPs.	RW
[42]	*Reserved*	Reserved for future implementation.	
[41]	WE	Write Cache Enable. If cleared, all W-cache references are handled as W-cache misses. Each store queue entry performs an RMW transaction to the L2-cache, and the W-cache is maintained in a clean state. Software is required to flush the W-cache (force it to a clean state) before setting this bit to 0.	RW
[40:33]	PM[7:0]	DCU Physical Address Data Watchpoint Mask. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[32:25]	VM[7:0]	DCU Virtual Address Data Watchpoint Mask. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[24], [23]	PR, PW	DCU Physical Address Data Watchpoint Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[22], [21]	VR, VW	DCU Virtual Address Data Watchpoint Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[20:4]	*Reserved*	Reserved for future implementation.	
[3]	DM	D-MMU Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[2]	IM	I-MMU Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[1]	DC	Data Cache Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW
[0]	IC	Instruction Cache Enable. Implemented as described in the *UltraSPARC III Cu Processor User's Manual.*	RW

9.4.2 Data Watchpoint Registers

The UltraSPARC IV+ processor supports a 43-bit physical address space. Software is responsible to write a zero-extended 64-bit address into the PA Data Watchpoint Register.

Note – Prefetch instructions do not generate *VA/PA_watchpoint* traps.

9.5 ScratchPad Registers (ASI_SCRATCHPAD_*n*_REG)

Each logical processor in the UltraSPARC IV+ processor implements eight ScratchPad registers (64-bit each, read/write accessible). The addresses of the ScratchPad registers are defined in TABLE 9-4 . The use of the ScratchPad registers is completely defined by software. The ScratchPad registers provide faster access than saving data in main memory and are not subject to register windowing. All ScratchPad registers in the UltraSPARC IV+ processor are equally as fast.

TABLE 9-4 ASI_SCRATCHPAD_n_REG Register

REGISTER NAME	ASI #	VA	SHARED?	ACCESS	SP
ASI_SCRATCHPAD_0_REG	0x4F	0x00	Private	RD/WR	NO
ASI_SCRATCHPAD_1_REG	0x4F	0x08	Private	RD/WR	NO
ASI_SCRATCHPAD_2_REG	0x4F	0x10	Private	RD/WR	NO
ASI_SCRATCHPAD_3_REG	0x4F	0x18	Private	RD/WR	NO
ASI_SCRATCHPAD_4_REG	0x4F	0x20	Private	RD/WR	NO
ASI_SCRATCHPAD_5_REG	0x4F	0x28	Private	RD/WR	NO
ASI_SCRATCHPAD_6_REG	0x4F	0x30	Private	RD/WR	NO
ASI_SCRATCHPAD_7_REG	0x4F	0x38	Private	RD/WR	NO

10

Address Space Identifiers

A SPARC V9 processor generates an address space identifier (ASI) with every address sent to memory. The ASI provides the following:

- The ability to distinguish different address spaces
- An attribute that is unique to an address space
- A map of the internal control and diagnostic registers within a processor

The UltraSPARC IV+ processor memory management hardware translates a 64-bit virtual address and an 8-bit ASI to a 43-bit physical address.

The UltraSPARC IV+ processor supports both big-endian and little-endian byte ordering. The default data access byte ordering after a power-on reset is big-endian. Instruction fetches are always big-endian.

Note – Programmers must not issue any memory operation with `ASI_PHYS_USE_EC` or `ASI_PHYS_USE_EC_LITTLE` to any bootbus address.

This chapter discusses the following sections:

Chapter Topics

10.1 I-TSB ASI Groupings

Internal ASIs (also called *non-translating ASIs*) are in the ranges of 30_{16} to $6F_{16}$, 72_{16} to 77_{16}, and $7A_{16}$ to $7F_{16}$. These ASIs are not translated by the MMU. Instead, these ASIs pass through their virtual addresses as physical addresses.

Note – Access to internal ASIs with invalid virtual addresses have undefined behavior. Invalid virtual addresses may or may not cause a *data_access_exception* trap, and may or may not alias onto a valid virtual address. Software should not rely on any specific behavior.

10.1.1 Fast Internal ASIs

In the UltraSPARC IV+ processor, internal ASIs are further classified into either fast or regular ASIs. Fast internal ASIs have a 3-cycle read latency (same as the regular load latency for a D-cache hit). Data for fast internal ASIs are returned in the M-stage of the pipeline without recirculating. Regular internal ASIs, on the other hand, have a longer read latency (approximately 20 cycles). The regular internal ASIs always recirculate once and return the data in the M-stage of the recirculated instruction. The latency of the regular internal ASIs depends on the ASI's address.

The ASIs listed in TABLE 10-1 are implemented as fast ASIs in the UltraSPARC IV+ processor. The balance of the internal ASIs are all regular ASIs.

TABLE 10-1 Fast Internal ASIs

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W
50_{16}	ASI_IMMU_TAG_TARGET	00_{16}	I-TSB Tag Target Register	R
50_{16}	ASI_IMMU_TAG_ACCESS	30_{16}	I-TLB Tag Access Register	RW
51_{16}	ASI_IMMU_TSB_8K_PTR_REG	00_{16}	I-TSB 8 KB Pointer Register	R
52_{16}	ASI_IMMU_TSB_64K_PTR_REG	00_{16}	I-TSB 64 KB Pointer Register	R
58_{16}	ASI_DMMU_TAG_TARGET	00_{16}	D-TSB Tag Target Register	R
58_{16}	ASI_DMMU_TAG_ACCESS	30_{16}	D-TLB Tag Access Register	RW
59_{16}	ASI_DMMU_TSB_8K_PTR_REG	00_{16}	D-TSB 8 KB Pointer Register	R
$5A_{16}$	ASI_DMMU_TSB_64K_PTR_REG	00_{16}	D-TSB 64 KB Pointer Register	R
$4F_{16}$	ASI_SCRATCHPAD_n_REG (n = 0-7)	00_{16} to 38_{16}	Scratchpad Registers	RW

10.1.2 Rules for Accessing Internal ASIs

All stores to internal ASIs are single-instruction group (SIG) instructions that basically have membar semantics before they can be issued. Stores to internal ASIs must be followed by a `MEMBAR #Sync`, `FLUSH`, `DONE`, or `RETRY` before the effects of the store are guaranteed to have taken effect. This includes ASI store to scratchpad registers which must have a `MEMBAR #Sync` separating a write from any subsequent read. Specifically,

- A `MEMBAR #Sync` is needed after an internal ASI store (other than MMU ASIs), before the point that side effects must be visible. This `MEMBAR` must precede the next load or non-internal store. To avoid data corruption, the `MEMBAR` must also be in or before the delay slot of a delayed control transfer instruction of any type.
- A `MEMBAR #Sync` is needed after an internal ASI store, before a load from any internal ASI. If a `MEMBAR` is not used, the load data returned for the internal ASI is not defined.
- A `FLUSH`, `DONE`, or `RETRY` is needed after an internal ASI store that affects instruction accesses, including the I/D-MMU ASIs (ASI 50_{16} to 52_{16}, 54_{16} to $5F_{16}$), the I-cache ASIs (66_{16} to $6F_{16}$), or the IC bit in the DCU Control Register (ASI 45_{16}), before the point that side effects must be visible. Stores to D-MMU registers (ASI 58_{16} to $5F_{16}$) other than the context ASIs (ASI 58_{16}, VA=8_{16}, 10_{16}) can use a `MEMBAR #Sync`. A `MEMBAR`, `FLUSH`, `DONE`, or `RETRY` instruction must precede the next load or non-internal store. The instruction must also be in, or before the delay slot of a delayed control transfer instruction to avoid corrupting data.

Note – For more predictable behavior, it is recommended to park the other logical processor when performing ASI accesses to a shared resource. If the other logical processor is not parked, it may perform operations making use of and/or modifying the shared resource being accessed.

10.1.3 Limitation of Accessing Internal ASIs

STXA to an internal ASI may corrupt the data returned for a subsequent LDXA from an internal ASI due to an exception which occurred prior to the protecting MEMBAR #Sync. For example, the following code sequence (where %asi is set to 0x5a) might cause this issue.

ASI Set To 0x5a.

```
init_mondo+24: stxa %o1, [%g0 + 50] %asi
init_mondo+28: stxa %o2, [%g0 + 60] %asi
init_mondo+2c: jmp %o7, 8, %g0
init_mondo+30: membar #Sync
```

In this particular case, a vector interrupt trap was taken on the JMP instruction. The interrupt trap handler executed an LDXA to the Interrupt Dispatch Register (ASI_INTR_DISPATCH_W, 0x77) which returned indeterminate data as the second STXA was still in progress (in this case, the data returned was the data written by the first STXA).

The reason the above case failed is that the JMP instruction took the interrupt before the MEMBAR #Sync semantics was invoked, thus leaving the interrupt trap handler unprotected. Besides interrupts, the JMP in this code sequence is also susceptible to the following traps:

- The trap, *mem_address_not_aligned*
- The trap, *illegal_instruction*
- Instruction breakpoint (debug feature which manifests itself as an illegal instruction, but is currently unsupported)
- I-MMU miss
- The trap, *instruction_access_exception*
- The trap, *instruction_access_error*
- The trap, *fast_ECC_error*

The specific problem observed can be avoided by using the following code sequence:

Jmp instruction interrupt before MEMBAR #Sync.

```
init_mondo+24: stxa %o1, [%g0 + 50] %asi
init_mondo+28: stxa %o2, [%g0 + 60] %asi
init_mondo+2c: membar #Sync
init_mondo+30: jmp %o7, 8, %g0
```

This approach works even though both the second STXA and the MEMBAR #Sync can take interrupts. STXA to an MMU Register and MEMBAR #Sync implicitly wait for all previous stores to complete before starting down the pipeline. Thus, if the second STXA or the MEMBAR takes an interrupt, it does so only at the end of the pipeline, after having made sure that all previous stores were complete. In the above case, the MEMBAR #Sync is still susceptible to all the traps noted above, except interrupts and *mem_address_not_aligned*:

- I-MMU miss
- *Illegal_instruction*
- Instruction breakpoint (debug feature which manifests itself as an illegal instruction, but is currently unsupported)
- *Instruction_access_exception*
- *Instruction_access_error*
- Fast ECC error

Note – DONE/RETRY can take a *privileged_opcode* trap if used in place of MEMBAR #Sync. This possibility is not considered since as any STXA that target internal ASIs must be done in privileged mode.

The second part of the code is to start any of the vulnerable trap handlers with a MEMBAR #Sync, an approach which has also been verified in the system, especially if they use LDXA instructions which target internal ASIs (ASI values 0x30 to 0x6f, 0x72 to 0x77, and 0x7a to 0x7f).

In the case of the I-MMU miss handler, this approach may result in unacceptable performance reduction. In such a case, it is recommended that both the STXA and the protecting MEMBAR #Sync (or FLUSH, DONE, or RETRY) are always on the same 8 KB page, thus eliminating the possibility of an intervening I-MMU miss (unless the code is otherwise guaranteed to not take an I-MMU miss, e.g., it was guaranteed to be locked down in the TLB).

The code described should be sufficient in all cases where an STXA to an internal ASI is either followed immediately by another such STXA or by one of the protecting instructions -- MEMBAR #Sync, FLUSH, DONE, or RETRY.

Note – In cases where other interruptable instructions are used after an STXA and before a protecting instruction, any exception handler which can be invoked would need similar protection. Such coding style is strongly discouraged and should only be done with great care when there are compelling performance reasons (e.g., in TLB miss handlers).

10.2 ASI Assignments

Please refer to the *UltraSPARC III Cu Processor User's Manual* for general information regarding the ASI assignments. The sections below discuss the UltraSPARC IV+ processor ASI assignments.

10.2.1 UltraSPARC IV+ Processor ASI Assignments

TABLE 10-2 defines all ASIs supported by the UltraSPARC IV+ processor that are not defined by either *The SPARC Architecture Manual*, Version 9 or are new. These can be used only with `LDXA`, `STXA`, or `LDDFA`, `STDFA` instructions, unless otherwise noted. Other-length accesses cause a *data_access_exception* trap.

TABLE 10-2 The UltraSPARC IV+ processor ASI Extensions *(1 of 5)*

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W	Private/ Shared
04_{16}	(*The SPARC Architecture Manual*, Version 9)				Private
$0C_{16}$	(*The SPARC Architecture Manual*, Version 9)				Private
10_{16} to 11_{16}	(*The SPARC Architecture Manual*, Version 9)				Private
14_{16} to 15_{16}	*Reserved*		Reserved for future implementation.		Private
18_{16} to 19_{16}	(*The SPARC Architecture Manual*, Version 9)				Private
$1C_{16}$ to $1D_{16}$	*Reserved*		Reserved for future implementation.		Private
24_{16}	*Reserved*		Reserved for future implementation.		Private
$2C_{16}$	*Reserved*		Reserved for future implementation.		Private
30_{16}	ASI_PCACHE_STATUS_DATA		P-cache data status RAM diagnostic access.	RW	Private
31_{16}	ASI_PCACHE_DATA		P-cache data RAM diagnostic access.	RW	Private
32_{16}	ASI_PCACHE_TAG		P-cache tag RAM diagnostic access.	RW	Private
33_{16}	ASI_PCACHE_SNOOP_TAG		P-cache snoop tag RAM diagnostic access.	RW	Private
34_{16}	ASI_QUAD_LDD_PHYS		128-bit atomic load.	R	Private
38_{16}	ASI_WCACHE_STATE	[10:6]	W-cache state array diagnostic access.	RW	Private
39_{16}	ASI_WCACHE_DATA	[10:6]	W-cache data RAM diagnostic access.	RW	Private
$3A_{16}$	ASI_WCACHE_TAG	[10:6]	W-cache tag RAM diagnostic access.	RW	Private
$3C_{16}$	ASI_QUAD_LDD_PHYS_L		128-bit atomic load, little-endian.	R	Private
$3F_{16}$	ASI_SRAM_FAST_INIT_SHARED		Fast initialize all SRAM in shared loop.	W	Shared
40_{16}	ASI_SRAM_FAST_INIT		Fast initialize all SRAM in logical processor.	W	Private
41_{16}	ASI_CORE_AVAILABLE	00_{16}	Bit mask of implemented logical processors.	R	Shared
41_{16}	ASI_CORE_ENABLED	10_{16}	Bit mask of enabled logical processors.	R	Shared

TABLE 10-2 The UltraSPARC IV+ processor ASI Extensions *(2 of 5)*

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W	Private/ Shared
41_{16}	ASI_CORE_ENABLE	20_{16}	Bit mask of logical processors to enable after next reset.	RW	Shared
41_{16}	ASI_XIR_STEERING	30_{16}	Cregs XIR steering register.	RW	Shared
41_{16}	ASI_CMP_ERROR_STEERING	40_{16}	Specify ID of which logical processor to trap on non-logical processor-specific error.	RW	Shared
41_{16}	ASI_CORE_RUNNING_RW	50_{16}	Bit mask to control which logical processors are active and which are parked. 1=active, 0=parked	RW	Shared
41_{16}	ASI_CORE_RUNNING_STATUS	58_{16}	Bit mask of logical processors that are currently active. 1=active, 0=parked.	R	Shared
41_{16}	ASI_CORE_RUNNING_W1S	60_{16}	logical processor parking register, write-one to set bit(s).	W	Shared
41_{16}	ASI_CORE_RUNNING_W1C	68_{16}	logical processor parking register, write-one to clear bit(s).	W	Shared
42_{16}	ASI_DCACHE_INVALIDATE		D-cache Invalidate diagnostic access.	W[1]	Private
43_{16}	ASI_DCACHE_UTAG		D-cache mTag diagnostic access.	RW	Private
44_{16}	ASI_DCACHE_SNOOP_TAG		D-cache snoop tag RAM diagnostic access.	RW	Private
45_{16}	*Reserved*		Reserved for future implementation.		Private
46_{16}	ASI_DCACHE_DATA		D-cache data RAM diagnostic access.	RW	Private
47_{16}	ASI_DCACHE_TAG		D-cache tag/valid RAM diagnostic access.	RW	Private
48_{16}	*Reserved*		Reserved for future implementation.		Private
49_{16}	*Reserved*		Reserved for future implementation.		Private
$4A_{16}$	ASI_FIREPLANE_CONFIG_REG	00_{16}	Sun Fireplane Interconnect config register.	RW	Shared
$4A_{16}$	ASI_FIREPLANE_CONFIG_REG	08_{16}	Sun Fireplane Interconnect address register.	RW	Shared
$4A_{16}$	ASI_FIREPLANE_CONFIG_REG_2	10_{16}	Sun Fireplane Interconnect config register II.	RW	Shared
$4A_{16}$	ASI_EMU_ACTIVITY_STATUS	18_{16}	EMU activity status register.	R	Shared
$4B_{16}$	ASI_L3STATE_ERROR_EN_REG	00_{16}	L3-cache error enable register.	RW	Shared
$4C_{16}$	ASI_ASYNC_FAULT_STATUS	00_{16}	Cregs async fault status register (AFSR).	RW	Private
$4C_{16}$	ASI_SECOND_ASYNC_FAULT_STATUS	08_{16}	Cregs secondary async fault status register (AFSR_2).	R	Private
$4C_{16}$	ASI_ASYNC_FAULT_STATUS_EXT	10_{16}	Cregs async fault status extension register (AFSR_EXT).	RW	Private

TABLE 10-2 The UltraSPARC IV+ processor ASI Extensions *(3 of 5)*

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W	Private/ Shared
$4C_{16}$	ASI_SECOND_ASYNC_FAULT_STATUS_ EXT	18_{16}	Cregs secondary async fault status extension register (AFSR_EXT_2).	R	Private
$4D_{16}$	*Reserved*		Reserved for future implementation.		Private
$4E_{16}$	ASI_L3CACHE_TAG	[26:6]	L3-cache tag state RAM data diagnostic access.	RW	Shared
$4F_{16}$	ASI_SCRATCHPAD_0_REG	00_{16}	Scratchpad register 0.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_1_REG	08_{16}	Scratchpad register 1.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_2_REG	10_{16}	Scratchpad register 2.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_3_REG	18_{16}	Scratchpad register 3.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_4_REG	20_{16}	Scratchpad register 4.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_5_REG	28_{16}	Scratchpad register 5.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_6_REG	30_{16}	Scratchpad register 6.	RW	Private
$4F_{16}$	ASI_SCRATCHPAD_7_REG	38_{16}	Scratchpad register 7.	RW	Private
50_{16}	*Reserved*	00_{16}, 18_{16}, 28_{16}, 30_{16}, 48_{16}, 58_{16}	Reserved for future implementation.		Private
50_{16}	ASI_IMMU_TAG_ACCESS_EXT	60_{16}	I-MMU Tag access extension register.	RW	Private
51_{16} to 54_{16}	*Reserved*		Reserved for future implementation.		Private
55_{16}	*Reserved*	00_{16} to $20FF8_{16}$	Reserved for future implementation.		Private
55_{16}	ASI_I-TLB_DIAG_REG	40000_{16} to $60FF8_{16}$	I-TLB diagnostic register	RW	Private
56_{16} to 57_{16}	*Reserved*		Reserved for future implementation.		Private
58_{16}	*Reserved*	00_{16}, 08_{16}, 10_{16}, 18_{16}, 20_{16}, 28_{16}, 30_{16}, 38_{16}, 40_{16}, 48_{16}, 50_{16}, 58_{16}	Reserved for future implementation.		Private
58_{16}	ASI_DMMU_TAG_ACCESS_EXT	60_{16}	D-MMU Tag access extension register.	RW	Private
59_{16} to $5C_{16}$	*Reserved*		Reserved for future implementation.		Private

TABLE 10-2 The UltraSPARC IV+ processor ASI Extensions *(4 of 5)*

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W	Private/ Shared
$5D_{16}$	*Reserved*	00_{16} to $30FF8_{16}$	Reserved for future implementation.		Private
$5D_{16}$	ASI_DTLB_DIAG_REG	40000_{16} to $70FF8_{16}$	D-TLB diagnostic register.	RW	Private
$5E_{16}$ to 60_{16}	*Reserved*		Reserved for future implementation.		Private
63_{16}	ASI_INTR_ID	00_{16}	Software assigned unique interrupt ID for logical processor.	RW	Private
63_{16}	ASI_CORE_ID	10_{16}	Hardware assigned ID for logical processor .	R	Private
63_{16}	ASI_CESR_ID	40_{16}	Software assigned CESR_ID value.	RW	Private
66_{16}	ASI_ICACHE_INSTR (ASI_IC_INSTR)		I-cache RAM diagnostic access.	RW	Private
67_{16}	ASI_ICACHE_TAG (ASI_IC_TAG)		I-cache tag/valid RAM diagnostic access.	RW	Private
68_{16}	ASI_ICACHE_SNOOP_TAG (ASI_IC_STAG)		I-cache snoop tag RAM diagnostic access.	RW	Private
69_{16}	ASI_IPB_DATA		I-cache prefetch buffer data RAM diagnostic access.	RW	Private
$6A_{16}$	ASI_IPB_TAG		I-cache prefetch buffer tag RAM diagnostic access.	RW	Private
$6B_{16}$	ASI_L2CACHE_DATA	[21:3]	L2-cache data RAM diagnostic access.	RW	Shared
$6C_{16}$	ASI_L2CACHE_TAG	[23:6]	L2-cache tag RAM diagnostic access.	RW	Shared
$6D_{16}$	ASI_L2CACHE_CTRL		L2-cache control diagnostic access.	RW	Shared
$6E_{16}$	ASI_BTB_DATA	[9:5]	Jump predict table RAM diagnostic access.	RW	Private
$6F_{16}$	ASI_BRANCH_PREDICTION_ARRAY		Branch Prediction RAM diagnostic access.	RW	Private
70_{16} to 71_{16}	*Reserved*		Reserved for future implementation.		Private
72_{16}	ASI_MCU_TIM1_CTRL_REG	00_{16}	Cregs memory control I register.	RW	Shared
72_{16}	ASI_MCU_TIM2_CTRL_REG	08_{16}	Cregs memory control II register.	RW	Shared
72_{16}	ASI_MCU_ADDR_DEC_REG0	10_{16}	Cregs memory decoding register. (bank 0)	RW	Shared
72_{16}	ASI_MCU_ADDR_DEC_REG1	18_{16}	Cregs memory decoding register. (bank 1)	RW	Shared
72_{16}	ASI_MCU_ADDR_DEC_REG2	20_{16}	Cregs memory decoding register. (bank 2)	RW	Shared
72_{16}	ASI_MCU_ADDR_DEC_REG3	28_{16}	Cregs memory decoding register. (bank 3)	RW	Shared
72_{16}	ASI_MCU_ADDR_CTRL_REG	30_{16}	Cregs memory address control register.	RW	Shared

TABLE 10-2 The UltraSPARC IV+ processor ASI Extensions *(5 of 5)*

Value	ASI Name (Suggested Macro Syntax)	VA	Description	R/W	Private/ Shared
72_{16}	ASI_MCU_TIM3_CTRL_REG	38_{16}	Cregs memory control III register.	RW	Shared
72_{16}	ASI_MCU_TIM4_CTRL_REG	40_{16}	Cregs memory control IV register.	RW	Shared
72_{16}	ASI_MCU_TIM5_CTRL_REG	48_{16}	Cregs memory control V register.	RW	Shared
74_{16}	ASI_L3CACHE_DATA	[5:3]	L3-cache data staging register.	RW	Shared
75_{16}	ASI_L3CACHE_CTRL		L3-cache control register.	RW	Shared
76_{16}	ASI_L3CACHE_W		L3-cache data RAM diagnostic write access.	W[1]	Shared
77_{16}	*Reserved*		Reserved for future implementation.		Private
78_{16} to 79_{16}	*Reserved*		Reserved for future implementation.		Private
$7E_{16}$	ASI_L3CACHE_R		L3-cache data RAM diagnostic read access.	R[1]	Shared
$7F_{16}$	*Reserved*		Reserved for future implementation.		Private
80_{16} to 83_{16}	(*The SPARC Architecture Manual*, Version 9)				Private
88_{16} to $8B_{16}$	(*The SPARC Architecture Manual*, Version 9)				Private
$C0_{16}$ to $C5_{16}$	*Reserved*		Reserved for future implementation.		Private
$C8_{16}$ to CD_{16}	*Reserved*		Reserved for future implementation.		Private
$D0_{16}$ to $D3_{16}$	*Reserved*		Reserved for future implementation.		Private
$D8_{16}$ to DB_{16}	*Reserved*		Reserved for future implementation.		Private
$E0_{16}$ to $E1_{16}$	*Reserved*		Reserved for future implementation.		Private
$F0_{16}$ to $F1_{16}$	*Reserved*		Reserved for future implementation.		Private
$F8_{16}$ to $F9_{16}$	*Reserved*		Reserved for future implementation.		Private

1.Read- or write-only access will cause a *data_access_exception* trap.

11

Sun Fireplane Interconnect and Processor Identification

This chapter describes the UltraSPARC IV+ processor system bus in the following sections:

Chapter Topics

11.1 Sun Fireplane Interconnect ASI Extensions

Note – When performing an ASI access to shared resources, it is important that the other logical processor first be parked. If the other logical processor is not parked, there is no guarantee that the ASI access will complete in a timely fashion, as normal transactions to shared resources from the other logical processor will have higher priority than ASI accesses.

Sun Fireplane Interconnect ASI extensions include:

- Sun Fireplane Interconnect Port ID Register
- Sun Fireplane Interconnect Configuration Register
- Sun Fireplane Interconnect Configuration Register 2
- Sun Fireplane Interconnect Address Register
- Cluster Error Status Register ID (`ASI_CESR_ID`) Register

Note – In the following registers, "*Reserved*" bits are read as '0's and ignored during a write.

11.1.1 Sun Fireplane Interconnect Port ID Register

The private FIREPLANE_PORT_ID Register can be accessed only from the Sun Fireplane Interconnect as a read-only, non-cacheable, slave access at offset 0 of the address space of the processor port.

This register indicates the capability of the processor module. See TABLE 11-1 for the state of this register after reset. The FIREPLANE_PORT_ID Register is described in TABLE 11-1.

TABLE 11-1 FIREPLANE_PORT_ID Register Format

Bits	Field	Description
[63:56]	FC_{16}	A one-byte field containing the value FC_{16}. This field is used by the OpenBoot PROM to indicate that no Fcode PROM is present.
[55:27]	*Reserved*	Reserved for future implementation.
[26:17]	AID	10-bit Sun Fireplane Interconnect Agent ID, read-only field, copy of AID field of the Sun Fireplane Interconnect Configuration Register 2.
[16]	M/S	Master or Slave bit. Indicates whether the agent is a Sun Fireplane Interconnect master or a Sun Fireplane Interconnect slave device: 1 = master, 0 = slave. This bit is hardwired to '0' in the UltraSPARC IV+ processor (master device).
[15:10]	MID	Manufacturer ID, read-only field. Refer to the UltraSPARC IV+ processor data sheet for the content of this register.
[9:4]	MT	Module Type, read-only field, copy of MT field of the Sun Fireplane Interconnect Configuration Register.
[3:0]	MR	Module revision, read-only field, copy of the MR field of the Sun Fireplane Interconnect Configuration Register.

11.1.1.1 Sun Fireplane Interconnect Configuration Register

The Sun Fireplane Interconnect Configuration register can be accessed at ASI $4A_{16}$, VA = 0. All fields except bits [26:17] (INTR_ID) are shared by both logical processors and are identical to the corresponding fields in the Sun Fireplane Interconnect Configuration Register 2. When any field except bits [26:17] (INTR_ID) is updated, the corresponding field in the Sun Fireplane Interconnect Configuration Register 2 will be updated as well. The bits [26:17] (INTR_ID) are logical processor-specific. See TABLE 11-2 for the state of this register after reset. The fields in the register are described in this table as well.

TABLE 11-2 FIREPLANE_CONFIG Register Format ***(1 of 3)***

Bits	Field	Description
[63]	CPBK_BYP	Copyback Bypass Enable. If set, it enables the copyback bypass mode.
[62]	SAPE	SDRAM Address Parity Enable. If set, it enables the detection of SDRAM address parity.
[61]	NXE	New SSM Transactions Enable. If set, it enables the 3 new SSM transactions: RTSU: ReadToShare and update MTag from gM to gS RTOU: ReadToOwn and update MTag to gI UGM: Update MTag to gM
[60:59]	DTL_6	DTL_6, DTL_5, DTL_4, DTL_3, DTL_2, DTL_1 [1:0]: DTL (Dynamic Termination Logic) termination mode. 00_{16}: *Reserved* 01_{16}: DTL-end – Termination pullup 02_{16}: DTL-mid – 25 ohm pulldown 03_{16}: DTL-2 – Termination pullup and 25 ohm pulldown See TABLE 11-3 for DTL pin configuration.
[58:57]	DTL_5	
[56:55]	DTL_4	
[54:53]	DTL_3	
[52:51]	DTL_2	
[50:49]	DTL_1	
[48:45]	MR[3:0]	Module revision. Written at boot time by the OpenBoot PROM (OBP) code, which reads it from the module serial PROM.

TABLE 11-2 FIREPLANE_CONFIG Register Format ***(2 of 3)***

Bits	Field	Description
[44:39]	MT[5:0]	Module type. Written at boot time by OBP code, which reads it from the module serial PROM.
[38]	TOF	Timeout Freeze mode. If set, all timeout counters are reset and stop counting.
[37:34]	TOL[3:0]	Timeout Log value. Timeout period is $2^{(10 + (2 \times TOL))}$ Sun Fireplane Interconnect cycles. Setting TOL ≥ 10 results in the max timeout period of 2^{30} Sun Fireplane Interconnect cycles. Setting TOL = 9 results in the Sun Fireplane Interconnect timeout period of 1.75 seconds. A TOL value of 0 should not be used since the timeout could occur immediately or as much as 2^{10} Sun Fireplane Interconnect cycles later. TOL / Timeout Period / TOL / Timeout Period 0 / 2^{10} / 8 / 2^{26} 1 / 2^{12} / 9 / 2^{28} 2 / 2^{14} / 10 / 2^{30} 3 / 2^{16} / 11 / 2^{31} 4 / 2^{18} / 12 / 2^{32} 5 / 2^{20} / 13 / $2^{32 + 31}$ 6 / 2^{22} / 14 / 2^{33} 7 / 2^{24} / 15 / $2^{33 + 32}$
[33]	CLK[3]	Processor to Sun Fireplane Interconnect clock ratio bit [3]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (Please refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292).
[32]	DEAD	0: Back-to-back Sun Fireplane Interconnect Request Bus Mastership. 1: Inserts a dead cycle in between bus masters on the Sun Fireplane Interconnect Request Bus.

TABLE 11-2 FIREPLANE_CONFIG Register Format ***(3 of 3)***

Bits	Field	Description
[31:30]	*Reserved*	Reserved for future implementation.
[29]	CLK[2]	Processor to Sun Fireplane Interconnect clock ratio bit[2]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292).
[28:27]	DBG[1:0]	Debug. 0: Up to 15 outstanding transactions allowed 1: Up to 8 outstanding transactions allowed 2: Up to 4 outstanding transactions allowed 3: One outstanding transaction allowed
[26:17]	INTR_ID[9:0]	Contains the lower 10 bits of the logical processor Interrupt ID (ASI_INTR_ID) for each logical processor on the processor. This field is logical processor-specific and is not shared.
[16:15]	CLK[1:0]	Processor to Sun Fireplane Interconnect clock ratio bits[1:0]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292).
[14:9]	CBND[5:0]	CBND address limit. Physical address bits [42:37] are compared to the CBND field. If PA[42:37] CBASE and PA[42:37] < CBND, then PA is in COMA space (Remote_WriteBack not issued in SSM mode).
[8:3]	CBASE[5:0]	CBASE address limit. Physical address bits [42:37] are compared to the CBASE field. If PA[42:37] CBASE and PA[42:37] < CBND, then PA is in COMA space (Remote_WriteBack not issued in SSM mode).
[2]	SLOW	If set, it expects snoop responses from other Fireplane agents, using the slow snoop response.
[1]	HBM	Hierarchical bus mode. If set, uses the Sun Fireplane Interconnect protocol for a multilevel transaction request bus. If cleared, the UltraSPARC IV+ processor uses the Sun Fireplane Interconnect protocol for a single-level transaction request bus.
[0]	SSM	If set, performs Sun Fireplane Interconnect transactions in accordance with the Sun Fireplane Interconnect Scalable Shared Memory model.

TABLE 11-3 DTL Pin Configurations

DTL GROUPS	Power Workstation	Midrange Server		Enterprise Server		Power-on Reset State
		mid	end	mid	end	
DTL_1 Group 0 COMMAND_L[1:0] ADDRESS_L[42:4] MASK_L[9:0] ATRANSID_L[8:0] Group 2 ADDRARBOUT_L ADDRARBIN_L[4:0] Group 8 ADDRPTY_L	3_{16}	2_{16}	1_{16}	2_{16}	1_{16}	2_{16}
DTL_2 Group 1 INCOMING_L PREREQIN_L	1_{16}	2_{16}	1_{16}	1_{16}	1_{16}	2_{16}
DTL_3 Group 3 PAUSEOUT_L MAPPEDOUT_L SHAREDOUT_L OWNEDOUT_L	3_{16}	1_{16}	1_{16}	1_{16}	1_{16}	2_{16}
DTL_4 Group 5 DTRANSID_L[8:0] DTARG_L DSTAT_L[1:0] Group 6 TARGID_L[8:0] TTRANSID_L[8:0]	3_{16}	2_{16}	1_{16}	1_{16}	1_{16}	2_{16}
DTL_5 Group 11 ERROR_L FREEZE_L FREEZEACK_L CHANGE_L	1_{16}	2_{16}	2_{16}	1_{16}	1_{16}	1_{16}
DTL_6 Group 4 PAUSEIN_L OWNEDIN_L SHAREDIN_L MAPPEDIN_L	1_{16}	2_{16}	1_{16}	1_{16}	1_{16}	1_{16}

Note – The Sun Fireplane Interconnect bootbus signals CAStrobe_L, ACD_L, and Ready_L have their DTL configuration programmable through two the UltraSPARC IV+ processor package pins. All other Sun Fireplane Interconnect DTL signals that do not have a programmable configuration are configured as DTL-end.

Several fields of the Sun Fireplane Interconnect Configuration Register do not take effect until after a soft POR is performed. If these fields are read before a soft POR, then the value last written will be returned. However, this value may not be the one currently being used by the processor. The fields that require a soft POR to take effect are HBM, SLOW, DTL, CLK, MT, MR, NXE, SAPE, CPBK_BYP.

Low power mode clock rate is NOT supported in the UltraSPARC IV+ processor. The field [31:30] are kept for backward compatibility and will be write ignore and read 0.

11.1.1.2 The UltraSPARC IV+ Processor System Bus Clock Ratio

The default UltraSPARC IV+ processor boot up system bus clock ratio is 8:1. The UltraSPARC IV+ processor supports system bus clock ratio from 8:1 to 16:1.

11.1.1.3 Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register

CLK[3], CLK[2], CLK[1:0]: bit[33], bit[29], bit[16:15] of the Sun Fireplane Interconnect Configuration Register.

Set the processor to system bus clock ratio. These fields may only be written during initialization before any System Bus transactions are initiated.

0, 0, 00 - reserved
0, 0, 01 - reserved
0, 0, 10 - reserved
0, 0, 11 - reserved
0, 1, 00 - 8:1 processor to system clock ratio (default)
0, 1, 01 - 9:1 processor to system clock ratio (new)
0, 1, 10 - 10:1 processor to system clock ratio (new)
0, 1, 11 - 11:1 processor to system clock ratio (new)
1, 0, 00 - 12:1 processor to system clock ratio (new)
1, 0, 01 - 13:1 processor to system clock ratio (new)
1, 0, 10 - 14:1 processor to system clock ratio (new)
1, 0, 11 - 15:1 processor to system clock ratio (new)
1, 1, 00 - 16:1 processor to system clock ratio (new)
1, 1, 01 - reserved
1, 1, 10 - reserved
1, 1, 11 - reserved

Note – The UltraSPARC IV+ processor supports 8:1, 9:1, 10:1, 11:1, 12:1, 13:1, 14:1, 15:1, 16:1 processor to system clock ratio.

11.1.1.4 Sun Fireplane Interconnect Configuration Register 2

The Sun Fireplane Interconnect Configuration Register 2 can be accessed at ASI $4A_{16}$, VA = 0x10. All fields are shared by both logical processors. All fields except bits[26:17] are identical to the corresponding fields in the Sun Fireplane Interconnect Configuration Register. When any field except bits [26:17] (AID) is updated, the corresponding field in the Sun Fireplane Interconnect Configuration Register will be updated too. The register is illustrated below and described in TABLE 11-4.

TABLE 11-4 FIREPLANE_CONFIG_2 Register Format *(1 of 2)*

Bits	Field	Description
[63]	CPBK_BYP	Copyback Bypass Enable. If set, it enables the copyback bypass mode.
[62]	SAPE	SDRAM Address Parity Enable. If set, it enables the detection of SDRAM address parity.
[61]	NXE	New SSM Transactions Enable. If set, it enables the 3 new SSM transactions: RTSU: ReadToShare and update MTag from gM to gS RTOU: ReadToOwn and update MTag to gI UGM: Update MTag to gM
[60:59]	DTL_6	DTL_* [1:0]: DTL termination mode. 0_{16}: *Reserved* 1_{16}: DTL-end – Termination pullup 2_{16}: DTL-mid – 25 ohm pulldown 3_{16}: DTL-2 – Termination pullup and 25 ohm pulldown See TABLE 11-3 for DTL pin configuration.
[58:57]	DTL_5	
[56:55]	DTL_4	
[54:53]	DTL_3	
[52:51]	DTL_2	
[50:49]	DTL_1	
[48:45]	MR[3:0]	Module revision. Written at boot time by the OpenBoot PROM (OBP) code, which reads it from the module serial PROM.
[44:39]	MT[5:0]	Module type. Written at boot time by OBP code, which reads it from the module serial PROM.
[38]	TOF	Timeout Freeze mode. If set, all timeout counters are reset and stop counting.
[37:34]	TOL[3:0]	Timeout Log value. Timeout period is $2^{(10 + (2 \times TOL))}$ Sun Fireplane Interconnect cycles. Setting TOL ≥ 10 results in the max timeout period of 2^{30} Sun Fireplane Interconnect cycles. Setting TOL = 9 results in the Sun Fireplane Interconnect timeout period of 1.75 seconds. A TOL value of 0 should not be used since the timeout could occur immediately or as much as 2^{10} Sun Fireplane Interconnect cycles later. TOL / Timeout Period: 0 → 2^{10}; 1 → 2^{12}; 2 → 2^{14}; 3 → 2^{16}; 4 → 2^{18}; 5 → 2^{20}; 6 → 2^{22}; 7 → 2^{24}; 8 → 2^{26}; 9 → 2^{28}; 10 → 2^{30}; 11 → 2^{31}; 12 → 2^{32}; 13 → 2^{32+31}; 14 → 2^{33}; 15 → 2^{33+32}
[33]	CLK[3]	Processor to Sun Fireplane Interconnect clock ratio bit [3]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (Please refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292).
[32]	DEAD	0: Back-to-back Sun Fireplane Interconnect Request Bus Mastership. 1: Inserts a dead cycle in between bus masters on the Sun Fireplane Interconnect Request Bus.

TABLE 11-4 FIREPLANE_CONFIG_2 Register Format *(2 of 2)*

Bits	Field	Description
[31:30]	*Reserved*	Reserved for future implementation.
[29]	CLK[2]	Processor to Sun Fireplane Interconnect clock ratio bit[2]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292)
[28:27]	DBG[1:0]	Debug. 0: Up to 15 outstanding transactions allowed 1: Up to 8 outstanding transactions allowed 2: Up to 4 outstanding transactions allowed 3: One outstanding transaction allowed
[26:17]	AID[9:0]	Contains the 10-bit Sun Fireplane Interconnect bus agent identifier for this processor. This field must be initialized on power-up before any Sun Fireplane Interconnect transactions are initiated.
[16:15]	CLK[1:0]	Processor to Sun Fireplane Interconnect clock ratio bits[1:0]. This field may only be written during initialization before any Sun Fireplane Interconnect transactions are initiated. (refer to *Additional CLK Encoding in the Sun Fireplane Interconnect Configuration Register* on page 292).
[14:9]	CBND[5:0]	CBND address limit. Physical address bits [42:37] are compared to the CBND field. If PA[42:37] CBASE and PA[42:37] < CBND, then PA is in COMA space (Remote_WriteBack not issued in SSM mode).
[8:3]	CBASE[5:0]	CBASE address limit. Physical address bits [42:37] are compared to the CBASE field. If PA[42:37] CBASE and PA[42:37] < CBND, then PA is in COMA space (Remote_WriteBack not issued in SSM mode).
[2]	SLOW	If set, it expects snoop responses from other Sun Fireplane Interconnect agents, using the slow snoop response.
[1]	HBM	Hierarchical bus mode. If set, uses the Sun Fireplane Interconnect protocol for a multilevel transaction request bus. If cleared, the UltraSPARC IV+ processor uses the Sun Fireplane Interconnect protocol for a single-level transaction request bus.
[0]	SSM	If set, performs Sun Fireplane Interconnect transactions in accordance with the Sun Fireplane Interconnect Scalable Shared Memory model.

Note – The Sun Fireplane Interconnect bootbus signals CAStrobe_L, ACD_L, and Ready_L have their DTL configuration programmable through the UltraSPARC IV+ processor package pins. All other Sun Fireplane Interconnect DTL signals that do not have a programmable configuration are configured as DTL-end.

Several fields of the Sun Fireplane Interconnect Configuration Register 2 do not take effect until after a soft POR is performed. If these fields are read before a soft POR, then the value last written will be returned. However, this value may not be the one currently being used by the processor. The fields that require a soft POR to take effect are HBM, SLOW, DTL, CLK, MT, MR, AID[4:0], NXE, SAPE, CPBK_BYP.

Low-power mode clock rate is NOT supported in the UltraSPARC IV+ processor. The field [31:30] are kept for backward compatibility and will be write-ignore and read 0.

11.1.1.5 Fireplane Interconnect Address Register

The Fireplane Interconnect Register can be accessed at ASI $4A_{16}$, VA = 08_{16}.

TABLE 11-5 describes the Address register

TABLE 11-5 Fireplane Address Register

Bit	Field	Description
[63:43]	*Reserved*	Reserved for future implementation.
[42:23]	Address	Address is the 20-bit physical address of the Fireplane-accessible registers (FIREPLANE_PORT_ID and Memory Controller registers). These address bits correspond to Sun Fireplane bus address bits PA[42:23].
[22:0]	*Reserved*	Reserved for future implementation.

11.1.1.6 Cluster Error Status Register ID (`ASI_CESR_ID`) Register

See TABLE 11-6 for the state of this register after reset. Refer to *CESR (Cluster Error Status Register) ID Register* on page 14 for further information.

11.2 RED_state and Reset Values

Reset values and RED_state for Sun Fireplane Interconnect-specific machines are listed in TABLE 11-6.

TABLE 11-6 Sun Fireplane Interconnect-Specific Machine State After Reset and RED_state

Name	Fields	Hard_POR	System Reset	WDR
FIREPLANE_PORT_ID	FC	FC_{16}		
	AID	0		
	MID	$3E_{16}$		
	MT	undefined[1]		
	MR	undefined[1]		
FIREPLANE_CONFIG	SSM	0	unchanged	unchanged
	HBM	0	updated[2]	unchanged
	SLOW	0	updated[2]	unchanged
	CBASE	0	unchanged	unchanged
	CBND	0	unchanged	unchanged
	CLK	0	updated[2]	unchanged
	MT	0	updated[2]	unchanged
	MR	0	updated[2]	unchanged
	INTR_ID	unknown	unchanged	unchanged
	DBG	0	unchanged	unchanged
	DTL	see TABLE 11-3	updated[2]	unchanged
	TOL	0	unchanged	unchanged
	TOF	0	unchanged	unchanged
	NXE	0	updated[2]	unchanged
	SAPE	0	updated[2]	unchanged
	CPBK_BYP	0	updated[2]	unchanged
FIREPLANE_CONFIG_2	SSM	0	unchanged	unchanged
	HBM	0	updated[3]	unchanged
	SLOW	0	updated[3]	unchanged
	CBASE	0	unchanged	unchanged
	CBND	0	unchanged	unchanged
	CLK	0	updated[3]	unchanged
	MT	0	updated[3]	unchanged
	MR	0	updated[3]	unchanged
	AID	0	[9:5] unchanged, [4:0] updated[3]	unchanged
	DBG	0	unchanged	unchanged
	DTL	see TABLE 11-3	updated[3]	unchanged
	TOL	0	unchanged	unchanged
	TOF	0	unchanged	unchanged
	NXE	0	updated[2]	unchanged
	SAPE	0	updated[2]	unchanged
	CPBK_BYP	0	updated[2]	unchanged
FIREPLANE_ADDRESS	all	unknown	unchanged	unchanged
ASI_CESR_ID	all	0	unchanged	unchanged

1.The state of Module Type (MT) and Module Revision (MR) fields of FIREPLANE_PORT_ID after reset is not defined. Typically, software reads a module PROM after a reset, then updates MT and MR.

2.This field of the FIREPLANE_CONFIG register is not immediately updated when written by software. Writes to this field of the FIREPLANE_CONFIG register have no visible effect until a reset occurs.

3.This field of the FIREPLANE_CONFIG_2 register is not immediately updated when written by software. Writes to this field of the FIREPLANE_CONFIG register have no visible effect until a reset occurs.

11.3 The UltraSPARC IV+ Processor Identification

11.3.1 Version Register

The 64-bit Version (VER) register is only accessible via the SPARC V9 `RDPR` instruction. It is not accessible through an ASI or ASR. In SPARC assembly language, the instruction is "`rdpr %ver, %rd`."

The VER.mask field consists of two 4-bit subfields: the "major" mask number (VER bits[31:28]) and the "minor" mask number (VER bits[27:24]). Please refer to TABLE 11-7.

TABLE 11-7 VER Register Encoding in Panther

Bit	Field	Value
[63:48]	manuf	$003E_{16}$
[47:32]	impl	0019_{16}
[31:24]	mask	starts from 10_{16}
[23:16]	*Reserved*	0
[15:8]	maxtl	05_{16}
[7:5]	*Reserved*	0
[4:0]	maxwin	07_{16}

11.3.2 FIREPLANE_PORT_ID MID Field

The 6-bit MID field in the FIREPLANE_PORT_ID register contains the six least significant bits ($3E_{16}$) of Sun's JEDEC code.

11.3.3 Speed Data Register

The Speed Data register is a low-power mode programmable 64-bit register. It is programmed to hold the clock frequency information of the processor after final testing. The value stored in the register is the clock frequency in MHz divided by 25. This register is read accessible from the ASI bus using ASI 53_{16}, VA = 20_{16}. The data format for the Speed Data register is shown in TABLE 11-8.

TABLE 11-8 Speed Data Register Bits

Bit (Fuse)	Description
[63:8]	Mandatory value.
[7:0]	Speed data (clock frequency in MHz/25).

11.3.4 Program Version Register

The Program Version register is a low-power mode 64-bit register. It is programmed to hold the test program version used to test the processor. This register is read accessible from the ASI bus using ASI 53_{16}, VA = 30_{16}. The data format for the Speed Data register is shown in TABLE 11-9.

TABLE 11-9 Program Version Register Bits

Bit (Fuse)	Description
[63:8]	Mandatory value.
[7:0]	Program version.

12

Interrupt Handling

For general information, please refer to the *UltraSPARC III Cu Processor User's Manual*. This chapter describes the UltraSPARC IV+ processor implementation-specific information about interrupt handling in the following sections:

Chapter Topics

12.1 Interrupt ASI Registers

12.1.1 Interrupt Vector Dispatch Register

The UltraSPARC IV+ processor interprets all 10 bits of VA[38:29] when the Interrupt Vector Dispatch Register is written.

12.1.2 Interrupt Vector Dispatch Status Register

In the UltraSPARC IV+ processor, 32 BUSY/NACK pairs are implemented in the Interrupt Vector Dispatch Status Register.

12.1.3 Interrupt Vector Receive Register

The UltraSPARC IV+ processor sets all 10 physical module ID (MID) bits in the SID_U and SID_L fields of the Interrupt Vector Receive Register. UltraSPARC IV+ processor obtains SID_U from VA[38:34] of the interrupt source and SID_L from VA[33:29] of the interrupt source.

12.1.4 Logical Processor Interrupt ID Register

ASI 0x63, VA[63:0] == 0x00

Name: `ASI_INTR_ID`

Read/Write, Privileged access, per-logical processor register

Please refer to *LP Interrupt ID Register (ASI_INTR_ID)* on page 13.

12.2 CMT Related Interrupt Behavior

12.2.1 Interrupt to Disabled Logical Processor

If an interrupt is issued to a disabled logical processor, the target processor in which the disabled logical processor resides will not assert a Mapped Out signal for the interrupt transaction. The TO bit in AFSR will be asserted in the logical processor issuing the interrupt. The busy bit in the Interrupt Vector Dispatch Status Register will be cleared for the interrupt.

12.2.2 Interrupt to Parked Logical Processor

If an interrupt is issued to a parked logical processor, the target processor in which the parked logical processor resides will assert a snoop response and log the interrupt receive register for the incoming interrupt transaction the same way as a running logical processor. If the incoming interrupt is not “NACK”ed for a parked logical processor, the pipeline will process the interrupt after the logical processor is unparked.

13

Instruction and Data Memory Management Unit

The Instruction Memory Management Unit (I-MMU) conforms to the requirements set forth in the *UltraSPARC III Cu Processor User's Manual*. In particular, the I-MMU supports a 64-bit virtual address space, software TLB-miss processing (no hardware page table walk), simplified protection encoding, and multiple page sizes.

This chapter describes the Instruction Memory Management Unit, as seen by the operating system software, in these sections:

Chapter Topics

13.1 Instruction Memory Management Unit

13.1.1 Virtual Address Translation

The UltraSPARC IV+ processor supports a 64-bit virtual address (VA) space, with 43 bits of physical address (PA). The I-MMU's two Translation Lookaside Buffers (TLBs) and their respective page sizes are described in TABLE 13-1. Replacement pages not maintaining the T512 page size are covered by the T16 TLB.

TABLE 13-1 I-MMU TLBs

TLB name	TLB ID	Translating Page Size	Description
T16	0	8 KB, 64 KB, 512 KB, 4 MB	16-entry fully-associative, both for locked and unlocked pages
T512	2	8 KB, 64 KB	512-entry 2-way set-associative (256 entries per way), for unlocked pages

13.1.2 Larger & Programmable I-TLB

The UltraSPARC IV+ processor has two I-TLBs. The original 2-way set-associative I-TLB of the UltraSPARC III processor has been changed to 512-entries and is now known as the T512 I-TLB. The 16-entry fully-associative T16 I-TLB is the same as found in the UltraSPARC III processor, with the exception of now supporting 8 KB unlocked pages. When an instruction memory access is issued, its VA, Context, and PgSz are presented to the I-MMU. Both I-TLBs (T512 and T16) are accessed in parallel.

Note – Unlike in the UltraSPARC III processor, the UltraSPARC IV+ processor's T16 can support ***unlocked*** 8 KB pages. This is necessary in the case where the T512 is programmed to a non-8 KB page size, to ensure that the I-TLB's unlocked 8 KB pages will not get dropped.

The T512 page size (pgsz) is programmable, one PgSz per context (primary/nucleus). Software can set the PgSz fields in ASI_PRIMARY_CONTEXT_REG as described in TABLE 13-2.

TABLE 13-2 I-MMU and D-MMU Primary Context Register

Bit	Field	Description
[63:61]	N_pgsz0	
[60:58]	N_pgsz1	
[57:55]	N_pgsz_I	Nucleus context's page size for T512 in I-TLB
[54:23]	*Reserved*	Reserved for future implementation.
[24:22]	P_pgsz0	
[21:19]	P_pgsz1	
[18:16]	P_pgsz_I	Primary context's page size for T512 in I-TLB
[15:13]	*Reserved*	Reserved for future implementation.
[12:0]	PContext	Context identifier for the primary address space

Page size bit encoding (Two most significant bits are reserved to 0 in I-MMU):
000 = 8 KB
001 = 64 KB

Note – The ASI_PRIMARY_CONTEXT_REG resides in D-MMU. There is no I-MMU Secondary Context Register.

When changing page size on primary or nucleus context for the T512, the code must reside in a T16 page.

A FLUSH must be executed after programming a page size change that effects the I-TLB. This is to ensure that instructions fetched afterward the change are translated correctly.

13.1.2.1 I-TLB Access Operation

When an instruction memory access is issued, its VA, Context, and PgSz are presented to the I-MMU. Both I-TLBs (T512 and T16) are accessed in parallel. The fully associative T16 only needs the VA and the Context to CAM-match and output an entry (1 out of 16). The proper VA bits are compared based on the page size bits of each T16 entry (fast 3-bit encoding is used to define 8 KB, 64 KB, 512 KB, and 4 MB pages).

Since the T512 is not fully-associative, indexing the T512 array requires knowledge of the page size to properly select the VA bits (8 bits) to be used as the index as shown below:

if 8 KB page is selected, array index = VA[20:13]
if 64 KB page is selected, array index = VA[23:16]

The Context bits are used after the indexed entry comes out of each array bank/way, to qualify the context hit.

There are 3 possible Context numbers active in the processor, but only two are relevant to the I-TLB:

- primary (PContext field in ASI_PRIMARY_CONTEXT_REG)
- nucleus (default to Context = 0)

Determining which Context register to send to the I-MMU is based on the ASI encoding of primary/nucleus of the instruction memory access.

Since both I-TLBs are accessed in parallel, software must guarantee that there are no duplicate (stale) entries. Most of this responsibility lies in operating system software with the hardware providing some assistance to support full software control. The rules of I-TLB replacement, demap and context switching, must be followed to maintain consistent and correct behavior.

13.1.2.2 I-TLB Parity Protection

The T512 I-TLB supports parity protection for both tag and data arrays, however, the T16 does not support parity protection. The I-MMU generates an odd-parity for the tag array from a 58-bit parity-tree, and odd-parity for the data array from a 33-bit parity-tree upon I-TLB replacement.

Parities are calculated as follows:

Tag Parity = XOR(Size[0],Global,VA[63:21], Context[12:0])
Data Parity = XOR(PA[42:13],CP,CV,P)

Note – The I-TLB tag parity calculation can ignore Size[2:1] as this bits are always 0. The I-TLB data parity calculation can ignore NFO, IE, E, and W bits as these also are always 0. Due to physical implementation constraint, parity error will still be reported if these bits are flipped in the T512.

The CV bit is included in the I-TLB data parity calculation as this bit is maintained in the UltraSPARC IV+ processor. In the UltraSPARC III processor, the CV bit was read as zero and writes to it were ignored.

Parity bits are available in the same cycle that the tag and data are sent to the I-TLB. The tag parity is written to bit[60] of the tag array while the data parity is written to bit[35] of the data array.

During the I-TLB translation, the I-TLB generates parities on both tag and data, and then checks them against the parity bits previously written during replacement. The T512 has 4 parity trees, 2 per way, with one for the tag and one for the data arrays. Parity checking takes place in parallel with tag comparison. A tag and/or data parity error is reported as an *instruction_access_exception* with the fault status recorded in the I-SFSR register (Fault Type = 20_{16}).

Note – Both tag and data parities are checked even for invalid I-TLB entries.

When a trap is taken on an I-TLB parity error, software needs to invalidate the corresponding entry and write the entry with the good parity.

The I-TLB tag and data parity errors are masked off with the following conditions:

- If there is a translation hit in the T16.
- If the I-TLB parity checking is disabled. The I-TLB parity enable is controlled by bit[16] of the Dispatch Control Register.

- If the I-MMU enable bit is off. The I-MMU enable is controlled by bit[2] (IM) of the Data cache Unit Control Register.
- The T512 hit signal and the I-TLB tag parity error signal generation share many common bits including VA[63:21]. If one of these common bits is flipped, a miss trap (*fast_instruction_access_MMU_miss*) will take place instead of a parity error trap (*instruction_access_exception*) as *fast_instruction_access_MMU_miss* has a higher priority than *instruction_access_exception*. The parity error in this case will be indirectly corrected when the TLB entry gets replaced by the *fast_instruction_access_MMU_miss* trap handler.

The T512 also checks both tag and data parities during the I-TLB demap operation. If a parity error is found during the demap operation, the corresponding entry will be invalidated by the hardware regardless of whether it was a hit or miss. This will only clear the valid bit, but not the parity error. No *instruction_access_exception* will be reported during the demap operation.

When writing to the I-TLB using the Data In Register (`ASI_ITLB_DATA_IN_REG`) during a replacement, or using the Data Access Register (`ASI_ITLB_DATA_ACCESS_REG`), both tag and data parities are calculated in the I-MMU and sent to the I-TLB to be stored into the respective arrays. When writing to the I-TLB using the I-TLB Diagnostic Register (`ASI_ITLB_DIAG_REG`), both tag and data parities are explicitly specified in the to-be-stored data, with bit[46] being the tag parity and bit[47] being the data parity (see *D-MMU Synchronous Fault Status Registers (D-SFSR)* on page 334). When using `ASI_SRAM_FAST_INIT`, all tag and data parity bits will be cleared.

When reading the I-TLB using the Data Access Register (`ASI_ITLB_DATA_ACCESS_REG`), or the I-TLB Diagnostic Register (`ASI_ITLB_DIAG_REG`), both tag and data parities are available in bit[46] and bit[47] of the I-TLB Diagnostic Register.

During bypass mode, both the tag and the data bypass the I-TLB structure and no parity errors are generated. Parity error generation is also suppressed during ASI reads.

TABLE 13-3 summarizes the UltraSPARC IV+ processor I-MMU parity error behavior.

TABLE 13-3 I-MMU parity error behavior

Operation	I-MMU Enable (DCU Control register bit[2])	Parity Enable (DCR register bit[16])	T16	T512			Trap taken
			hit	TLB hit	Data parity error	Tag parity error	
Translation	0	x	x	x	x	x	no trap taken
	1	0	0	0	x	x	fast_instruction_access_MMU
	1	0	0	1	x	x	no trap taken
	1	1	0	0	0	0	fast_instruction_access_MMU
	1	1	0	0	1	x	fast_instruction_access_MMU; instruction_access_exception on retry if accessing T512 and the other way has the parity error.
	1	1	0	0	x	1	fast_instruction_access_MMU; instruction_access_exception on retry if accessing T512 and the other way has the parity error.
	1	1	0	1	1	x	instruction_access_exception
	1	1	0	1	x	1	instruction_access_exception
	1	1	1	0	1	x	no trap taken
	1	1	1	0	x	1	no trap taken
Demap	x	x	x	x	1	x	no trap taken
	x	x	x	x	x	1	no trap taken

NOTE: An “x” in the table represents don’t-cares.

13.1.3 I-TLB Automatic Replacement

The I-TLB-miss fast trap handler utilizes the automatic (hardware) replacement write using store `ASI_ITLB_DATA_IN_REG`.

When an I-TLB misses, an *instruction_access_exception*, or protection trap is detected, with the hardware automatically saving the missing VA and context to the Tag Access Register (`ASI_IMMU_TAG_ACCESS`). To facilitate indexing of the T512 when the TTE data is presented (via `STXA ASI_ITLB_DATA_IN_REG`), the missing page size information of the T512 is captured into a new extension register, called `ASI_IMMU_TAG_ACCESS_EXT` which is described in *I-TLB Tag Access Extension Register* on page 313.

The hardware I-TLB replacement algorithm is as follows:

Note – “PgSz” below is `ASI_IMMU_TAG_ACCESS_EXT[24:22]` bits.

See below for the I-TLB Replacement Algorithm:

```
if (TTE to fill is a locked page, (L bit is set))
{
   fill TTE to T16;
 }
else
 {
     if (TTE's Size == PgSz)
      {
         if (one of the 2 same-index entries is invalid)
          {
             fill TTE to that invalid entry;
           }
          else if (no entry is valid | both entries are valid)
          {
               case (LFSR[0])
                {
                     0: fill TTE to T512 way0;
                     1: fill TTE to T512 way1;
                 }
           }
        }
        else
           {
                fill TTE to T16;
            }
 }
```

13.1.3.1 Direct I-TLB Data Read/Write

As described in the *UltraSPARC III Cu Processor User's Manual*, each I-TLB can be directly written using the store ASI_ITLB_DATA_ACCESS_REG instruction. Software typically uses this method for initialization and diagnostic.

Page size information is determined by bit[48], [62:61] of TTE data (store data of STXA ASI_ITLB_DATA_ACCESS_REG). Direct accessing the I-TLB by properly selecting the TLB ID and TLB entry fields of the ASI virtual address is explained in the *D-TLB Tag Access Extension Registers* on page 331.

It is not required to write the Tag Access Extension Register (ASI_IMMU_TAG_ACCESS_EXT) with page size information since ASI_ITLB_DATA_ACCESS_REG gets the page size from the TTE data. But it is recommended that software writes proper page size information to ASI_IMMU_TAG_ACCESS_EXT before writing to ASI_ITLB_DATA_ACCESS_REG.

13.1.3.2 I-TLB Tag Read Register

The UltraSPARC IV+ processor's behavior on read to ASI_ITLB_TAG_READ_REG (ASI 56_{16}) is as follows:

- For the T16, the 64-bit data read is the same as in the UltraSPARC III processor, and is backward compatible (see the *UltraSPARC III Cu Processor User's Manual*).
- For the T512, the bit positions of VPN (virtual page number) within bits[63:13] change as follows:

bit[63:21] = VA[63:21] (for 8 KB/64 KB page)

bit[20:13] = I-TLB index (VA[23:16] for 64 KB page)

bit[20:13] = I-TLB index (VA[20:13] for 8 KB page)

bit[12:0] = Context[12:0]

The I-TLB tag array can be written during BIST mode to support back-to-back ASI writes and reads.

13.1.3.3 Demap Operation

For the demap-page in the large I-TLB, the page size used to index the I-TLB is derived based on the Context bits (primary/nucleus). The hardware will automatically select the proper PgSz bits based on the "Context" field (primary/nucleus) defined in `ASI_IMMU_DEMAP` (ASI 57_{16}). These two PgSz fields are used to properly index the T512.

Demap operations in the T16 are single cycle operations. All matching entries in the T16 are demapped in one cycle, however, Demap operations in the T512 are multi-cycle operations - demap one entry at a time.

13.1.3.4 I-TLB SRAM/BIST Mode

Back-to-back ASI writes and reads to the I-TLB data array is done using `ASI_ITLB_DATA_ACCESS_REG` (ASI 0x55). Back-to-back ASI writes and reads to the I-TLB tag array are done using `ASI_ITLB_TAG_READ_REG` (ASI 0x56).

13.1.3.5 I-TLB Access Summary

TABLE 13-4 lists the I-MMU TLB Access Summary.

Note – There is no I-MMU Synchronous Fault Address Register (SFAR). A missed VA is found in the TPC Register.

TABLE 13-4 I-MMU TLB Access Summary

Software Operation		Effect on MMU Physical Registers				
Load/ Store	Register	TLB tag array	TLB data array	Tag Access	Tag Access Ext	SFSR
Load	Tag Read	Contents returned. From entry specified by LDXA's access	No effect	No effect	No effect	No effect
	Tag Access	No effect	No effect	Contents returned	Contents returned	No effect
	Data In	Trap with instruction_access_exception *(see I-MMU Synchronous Fault Status Register (I-SFSR)* on page 316*)*				
	Data Access	No effect	Contents returned. From entry specified by LDXA's access	No effect	No effect	No effect
	SFSR	No effect	No effect	No effect	No effect	Contents returned.
Store	Tag Read	Trap with instruction_access_exception				
	Tag Access	No effect	No effect	Written with store data	No effect	No effect
	Data In	TLB entry determined by replacement policy written with contents of Tag Access Register	TLB entry determined by replacement policy written with store data	No effect	No effect	No effect
	Data Access	TLB entry specified by STXA address written with contents of Tag Access Register	TLB entry specified by STXA address written with store data	No effect	No effect	No effect
	SFSR	No effect	No effect	No effect	No effect	Written with store data
TLB miss		No effect	No effect	Written with VA and context of access	Written with miss page size	Written with fault status of faulting instruction and page sizes at faulting context for two 2 way set associative TLB

13.1.4 Translation Table Entry (TTE)

The Translation Table Entry (TTE) holds information for a single page mapping. The TTE is divided into two 64-bit words representing the tag and data of the translation. Just as in a hardware cache, the tag is used to determine whether there is a hit in the TSB; if there is a hit, then the data is fetched by software.

The configuration of the TTE is described in TABLE 13-5 (see also the *UltraSPARC III Cu Processor User's Manual*).

Note – All bootbus addresses must be mapped as side-effect pages with the TTE E bit set.

TABLE 13-5 Translation Table Entry (TTE) ***(1 of 2)***

Bit	Field	Description
[63]	V	See the *UltraSPARC III Cu Processor User's Manual*
[62:61]	Size[1:0]	Bit[62:61] are the encoded page size bits for I-TLB. 00: 8 KB page 01: 64 KB page 10: 512 KB page 11: 4 MB page
[60]	NFO	See the *UltraSPARC III Cu Processor User's Manual*
[59]	IE	See the *UltraSPARC III Cu Processor User's Manual*
[58:50]	Soft2	See the *UltraSPARC III Cu Processor User's Manual*
[49]	*Reserved*	Reserved for future implementation.

TABLE 13-5 Translation Table Entry (TTE) *(2 of 2)*

Bit	Field	Description
[48]	Size[2]	Bit 48 is the most significant bit of the page size and is concatenated with bits [62:61]. Size[2] is always 0 for I-TLB.
[47:43]	*Reserved*	Reserved for future implementation.
[42:13]	PA	The physical page number. Page offset bits for larger page sizes (PA[15:13], PA[18:13], and PA[21:13] for 64 KB, 512 KB, and 4 MB pages, respectively) are stored in the TLB and returned for a Data Access read but are ignored during normal translation. When page offset bits for larger page sizes (PA[15:13], PA[18:13], and PA[21:13] for 64 KB, 512 KB, and 4 MB pages, respectively) are stored in the TLB on the UltraSPARC IV+ processor, the data returned from those fields by a Data Access read are the data previously written to them.
[12:7]	Soft	See the *UltraSPARC III Cu Processor User's Manual*
[6]	L	If the lock bit is set, then the TTE entry will be locked down when it is loaded into the TLB; that is, if this entry is valid, it will not be replaced by the automatic replacement algorithm invoked by an ASI store to the Data In Register. The lock bit has no meaning for an invalid entry. Arbitrary entries can be locked down in the TLB. Software must ensure that at least one entry is not locked when replacing a TLB entry; otherwise, a locked entry will be replaced. Since the 16-entry, fully associative TLB is shared for all locked entries as well as for 4 MB and 512 KB pages, the total number of locked pages is limited to less than or equal to 15. In the UltraSPARC IV+ processor, the TLB lock bit is only implemented in the D-MMU 16-entry, fully- associative TLB, and the I-MMU 16-entry, fully associative TLB. In the TLBs dedicated to 8 KB page translations (D-MMU 512-entry, 2-way associative TLB and I-MMU 512-entry, 2-way associative TLB), each TLB entry's lock bit reads as 0 and writes to it are ignored. The lock bit set for 8 KB page translation in both I-MMU and D-MMU is read as 0 and ignored when written.
[5], [4]	CP, CV	The cacheable-in-physically-indexed-cache bit and cacheable-in-virtually-indexed-cache bit determine the placement of data in the caches. UltraSPARC IV+ processor fully implements the CV bit. The following table describes how CP and CV control cacheability in specific UltraSPARC IV+ processor caches. **TABLE 13-6 Meaning of TTE** **Cacheable (CP, CV)** / **Meaning of TTE when placed in the I-TLB** 00, 01 — Non-cacheable 10 — Cacheable L2/L3-cache & I-cache, but not installed in I-cache 11 — Cacheable L2/L3-cache & I-cache The MMU does not operate on the cacheable bits but merely passes them through to the cache subsystem. In the UltraSPARC IV+ processor, the CV bit is maintained in I-TLB.
[3]	E	See the *UltraSPARC III Cu Processor User's Manual*.
[2]	P	See the *UltraSPARC III Cu Processor User's Manual*.
[1]	W	See the *UltraSPARC III Cu Processor User's Manual*.
[0]	G	See the *UltraSPARC III Cu Processor User's Manual*.

13.1.5 Hardware Support for TSB Access

The MMU hardware provides services to allow the TLB-miss handler to efficiently reload a missing TLB entry for either an 8 KB or 64 KB page.

These services include:

- Formation the TSB Pointers, based on the missing virtual address and address space
- Formation of the TTE Tag Target used for the TSB tag comparison
- Efficient atomic write of a TLB entry with a single store ASI operation
- Alternate globals on MMU-signaled traps

Please refer to the *UltraSPARC III Cu Processor User's Manual* for additional details.

13.1.5.1 Typical I-TLB Miss/Refill Sequence

A typical TLB-miss and TLB-refill sequence:

1. An I-TLB miss causes a *fast_instruction_access_MMU_miss* exception.
2. The appropriate TLB-miss handler loads the TSB Pointers and the TTE Tag Target with loads from the MMU registers.
3. Using this information, the TLB miss handler checks to see if the desired TTE exists in the TSB. If so, the TTE data are loaded into the TLB Data In Register to initiate an atomic write of the TLB entry chosen by the replacement algorithm.
4. If the TTE does not exist in the TSB, then the TLB-miss handler jumps to the more sophisticated, and slower, TSB miss handler.

The virtual address used in the formation of the pointer addresses comes from the Tag Access Register, which holds the virtual address and context of the load or store responsible for the MMU exception. See *Translation Table Entry (TTE)* on page 328.

Note – There are no separate physical registers in hardware for the pointer registers; rather virtual registers are implemented through dynamic reordering of the data stored in the Tag Access and the TSB registers.

The hardware provides pointers for the most common cases of 8 KB and 64 KB page miss processing. These pointers give the virtual addresses where the 8 KB and 64 KB TTEs are stored if either is present in the TSB.

The TSB_Size field, *n*, is defined to have a value ranging from 0 to 7. Note that the TSB_Size refers to the size of each TSB when the TSB is split. The ▯ symbol designates concatenation of bit vectors and ⊕ indicates an exclusive-or operation.

For a shared TSB (TSB register split field = 0):

8K_PTR = TSB_Base[63:13+*n*] ⊕ TSB_Extension[63:13+*n*] ▯ VA[21+*n*:13] ▯ 0000

64K_PTR = TSB_Base[63:13+*n*] ⊕ TSB_Extension[63:13+*n*] ▯ VA[24+*n*:16] ▯ 0000

For a split TSB (TSB register split field = 1):

8K_PTR = TSB_Base[63:14+*n*] ⊕ TSB_Extension[63:14+*n*] ▯ 0 ▯ VA[21+*n*:13] ▯ 0000

$$64K\_PTR = TSB\_Base[63:14+n] \oplus TSB\_Extension[63:14+n] \;\square\; 1 \;\square\; VA[24+n:16] \;\square\; 0000$$

The TSB Tag Target is formed by aligning the missing access VA (from the Tag Access Register) and the current context to positions found above in the description of the TTE tag, allowing a simple `XOR` instruction to detect a TSB hit.

13.1.6 Faults and Traps

On a *mem_address_not_aligned* trap that occurs during the execution of a `JMPL` or `RETURN` instruction, the UltraSPARC IV+ processor updates the D-SFSR register with the FT field set to 0 and updates the D-SFAR register with the fault address. For details, please refer to the *UltraSPARC III Cu Processor User's Manual*.

13.1.7 Reset, Disable, and RED_state Behavior

Please refer to the *UltraSPARC III Cu Processor User's Manual* for general details.

When the I-MMU is disabled, it truncates all instruction accesses to the physical address size (implementation dependent) and passes the physically cacheable bit (Data Cache Unit Control Register CP bit) to the cache system. The access does not generate an *instruction_access_exception* trap.

13.1.8 Internal Registers and ASI Operations

Please refer to the *UltraSPARC III Cu Processor User's Manual* for details.

13.1.9 I-TLB Tag Access Extension Register

ASI 50_{16}, VA[63:0] == 60_{16},
Name: `ASI_IMMU_TAG_ACCESS_EXT`
Access: RW

Tag Access Extension Register keeps the missed page sizes of T512. The format of the I-TLB Tag Access Extension Register is described in TABLE 13-7.

TABLE 13-7 I-TLB Tag Access Extension Register

Bit	Field	Description
[63:25]	*Reserved*	Reserved for future implementation.
[24:22]	pgsz	Page size of I-TLB miss context (primary/nucleus) in the T512.
[21:0]	*Reserved*	Reserved for future implementation.

Note – Bit 24 and 23 are hardwired to zero since only one bit is required to decode 8 KB and 64 KB page sizes.

With the saved page sizes, the hardware pre-computes in the background the index to the T512 for a TTE fill. When the TTE data arrives, only one write enable to the T512 and the T16 will be activated.

13.1.9.1 I-TLB Data In, Data Access, and Tag Read Registers

Data In Register

ASI 54_{16}, VA[63:0] == 0016,
Name: `ASI_ITLB_DATA_IN_REG`
Access: W

Writes to the TLB Data In register requires the virtual address to be set to 0.

Note – The I-TLB Data In Register is used when the *fast_instruction_access_MMU_miss* trap is taken to fill I-TLB based on replacement algorithm. Other than the *fast_instruction_access_MMU_miss* trap, an ASI store to I-TLB Data in register may replace an unexpected entry in the I-TLB even if the entry is locked. The entry that gets updated depends on the state of the Tag Access Extension Register, LFSR bits and the TTE page size in the store data. If a nested *fast_instruction_access_MMU_miss* happens, the I-TLB Data in register will not work.

Data Access Register

ASI 55_{16}, VA[63:0] == 0016 - 20FF816,
Name: `ASI_ITLB_DATA_ACCESS_REG`
Access: RW

Virtual Address Format

The virtual address format of the I-TLB Data Access register is described in TABLE 13-8.

TABLE 13-8 I-TLB Data Access Register Virtual Address Format Description

Bit	Field	Description	R/W
[63:19]	*Reserved*	Reserved for future implementation.	
[18]	Mandatory value	Should be 0.	
[17:16]	TLB ID	The TLB to access, as defined below. 0 : T16 2 : T512	RW
[15:12]	*Reserved*	Reserved for future implementation.	
[11:3]	TLB Entry	The TLB entry number to be accessed, in the range 0–511. Not all TLBs will have all 512 entries. All TLBs regardless of size are accessed from 0 to $N - 1$, where N is the number of entries in the TLB. For the T512, bit[11] is used to select either way0 or way1, and bit[10:3] is used to access the specified index. For the T16, only bit[6:3] is used to access one of 16 entries.	RW
[2:0]	Mandatory value	Should be 0.	

Data Format

The Data Access Register uses the TTE data format with the addition of parity information in the T512 as described in TABLE 13-5.

Note – Size bits are stored in the tag array of the T512 in the UltraSPARC IV+ processor to correctly select bits for different page sizes. Page size bits are returned by all operations.

When writing to the T512, the two most significant size bits, Size[2:1](bit[48],bit[62]), are masked and forced to zero by the hardware. The Data Parity and Tag Parity bits, DP (bit[47]) and TP (bit[46]), are masked by the hardware during writes. The parity bits are calculated by the hardware and written to the corresponding I-TLB entry when replacement occurs as mentioned in *I-TLB Parity Protection* on page 304.

Tag Read Register

ASI 56_{16}, VA[63:0] == 0016 - 20FF816,
Name: `ASI_ITLB_TAG_READ_REG`
Access: RW

Virtual Address Format

The virtual address format of the I-TLB Tag Read Register.

Note – Bit[2:0] is 0.

Data Format

The data format of Tag Read Register is described in TABLE 13-9 and TABLE 13-10.

TABLE 13-9 Tag Read Register Data Format for T512

Bit	Field	Description	R/W
[63:21]	VA (T512)	VA[63:21] for both 8 KB and 64 KB pages.	R
[20:13]	Index (T512)	VA[20:13]for 8 KB pages, VA[23:16] for 64 KB pages.	R
[12:0]	Context	The 13-bit context identifier.	R

TABLE 13-10 Tag Read Register Data Format for T16

Bit	Field	Description	R/W
[63:13]	VA (T16)	The 51-bit virtual page number. In the fully associative TLB, page offset bits for larger page sizes are stored in TLB; that is VA[15:13], VA[18:13], and VA[21:13] for 64 KB, 512 KB, and 4 MB pages, respectively. These values are ignored during normal translation.	R
[12:0]	Context	The 13-bit context identifier.	R

Note – An ASI load from the I-TLB Diagnostic Register initiates an internal read of the data portion of the specified I-TLB. If any instruction that misses the I-TLB is followed by a diagnostic read access (`LDXA` from `ASI_ITLB_DATA_ACCESS_REG`, i.e., ASI 0x55) from the fully associative TLBs, and the target TTE has the page size set to 64 KB, 512 KB, or 4 MB, then the data returned from the TLB will be incorrect. This issue can be overcome by reading the fully associative TLB TTE twice, back to back. The first access may return incorrect data if the above conditions are met, however the second access will return correct data.

13.1.9.2 I-MMU TSB (I-TSB) Base Register

ASI 50_{16}, VA[63:0] == 2816,
Name: `ASI_IMMU_TSB_BASE`
Access: RW

The I-MMU TSB Base Register follows the same format as the D-MMU TSB. Please refer to *Tag Read Register* on page 333.

13.1.9.3 I-MMU TSB (I-TSB) Extension Registers

ASI 50_{16}, VA[63:0] == 4816,
Name: `ASI_IMMU_TSB_PEXT_REG`
Access: RW

ASI 50_{16}, VA[63:0] == 5816,
Name: `ASI_IMMU_TSB_NEXT_REG`
Access: RW

Please refer to the *UltraSPARC III Cu Processor User's Manual* for information on the TSB Extension Registers. In the UltraSPARC IV+ processor, TSB_Hash (bits [11:3] of the extension registers) are exclusive-ORed with the calculated TSB offset to provide a "hash" into the TSB. Changing the TSB_Hash field on a per-process basis minimizes the collision of TSB entries between different processes.

13.1.9.4 I-MMU Synchronous Fault Status Register (I-SFSR)

ASI 50_{16}, VA[63:0] == 1816,
Name: `ASI_IMMU_SFSR`
Access: RW

I-MMU SFSR is described in TABLE 13-11.

TABLE 13-11 I-SFSR Bit Description

Bit	Field	Description	R/W
[63:25]	*Reserved*	Reserved for future implementation.	
[24]	NF	Non-faulting load bit. Hardwired to zero in I-SFSR.	
[23:16]	ASI	Records the 8-bit ASI associated with the faulting instruction. This field is valid for all traps in which the FV bit is set.	RW
[15]	TM	I-TLB miss.	RW
[14:13]	*Reserved*	Reserved for future implementation.	
[12:7]	FT	Specifies the exact condition that caused the recorded fault, according to TABLE 13-12 following this table. In the I-MMU, the Fault Type field is valid only for instruction_access_exception faults; there is no ambiguity in all other MMU trap cases. Note that the hardware does not priority-encode the bits set in the fault type register; that is, multiple bits can be set.	RW
[6]	E	Side-effect bit. Hardwired to zero in I-SFSR.	
[5:4]	CT	Context Register selection. The context is set to 112 when the access does not have a translating ASI.	RW
[3]	PR	Privilege bit. Set if the faulting access occurred while in privileged mode. This field is valid for all traps in which FV bit is set.	RW
[2]	W	Write bit. Hardwired to zero in I-SFSR.	
[1]	OW	Overwrite bit. When the I-MMU detects a fault, the Overwrite bit is set to 1 if the FV bit has not been cleared from a previous fault; otherwise, it is set to 0.	RW
[0]	FV	Fault Valid bit. Set when the I-MMU detects a fault; it is cleared only on an explicit ASI write of 0 to SFSR. When the FV bit is not set, the values of the remaining fields in the SFSR and SFAR are undefined for traps.	RW

TABLE 13-12 FT[5:0]

I/D	FT[5:0]	Description
I	01_{16}	Privileged violation.
I	20_{16}	I-TLB tag or data parity error

Note – FT[4:1] are hardwired to zero. Bit[18] and bit[2:0] are set to 1 and 0 respectively.

Data Format

The Diagnostic Register format for the T16 is described below in TABLE 13-13 . Three new bits are added to the UltraSPARC IV+ processor in the T512 Diagnostic Register described in TABLE 13-14.

An ASI store to the I-TLB Diagnostic Register initiates an internal atomic write to the specified TLB entry. The Tag portion is obtained from the Tag Access Register, while the data portion is obtained from the to-be-stored data.

TABLE 13-13 I-TLB Diagnostic Register

Bit	Field	Description	R/W
[63:7]	*Reserved*	Reserved for future implementation.	
[6]	LRU	The LRU bit in the CAM, read-write.	RW
[5:3]	CAM SIZE	The 3-bit page size field from the RAM, read-only.	R
[2:0]	RAM SIZE	The 3-bit page size field from the CAM, read-only.	R

TABLE 13-14 T512 Diagnostic Register

Bit	Field	Description	R/W
[63]	V	See the *UltraSPARC III Cu Processor User's Manual*.	
[62:61]	Size[1:0]	Encode page size bits.	
[60]	NFO	See the *UltraSPARC III Cu Processor User's Manual*.	
[59]	IE	See the *UltraSPARC III Cu Processor User's Manual*.	
[58:50]	Soft2	See the *UltraSPARC III Cu Processor User's Manual*.	
[49]	*Reserved*	Reserved for future implementation.	
[48]	Size[2]	Always 0	
[47]	DP	Data parity bit.	RW
[46]	TP	Tag parity bit.	RW
[45:43]	*Reserved*	Reserved for future implementation.	
[42:13]	PA	Physical page number.	
[12:7]	Soft	See the *UltraSPARC III Cu Processor User's Manual*.	
[6]	L	Lock bit.	
[5], [4]	CP, CV	Cacheable bit.	
[3]	E	See the *UltraSPARC III Cu Processor User's Manual*.	
[2]	P	See the *UltraSPARC III Cu Processor User's Manual*.	
[1]	W	See the *UltraSPARC III Cu Processor User's Manual*.	
[0]	G	See the *UltraSPARC III Cu Processor User's Manual*.	

Note – See TABLE 13-5 for a detailed description of the fields.

An ASI load from the I-TLB Diagnostic Register initiates an internal read of the data portion of the specified I-TLB. If any instruction that misses the I-TLB is followed by a diagnostic read access (`LDXA` from `ASI_ITLB_DATA_ACCESS_REG`, i.e., ASI 0x55) from the fully associative TLBs, and the target TTE has the page size set to 64 KB, 512 KB, or 4 MB, then the data returned from the TLB will be incorrect. This issue can be overcome by reading the fully associative TLB TTE twice, back to back. The first access may return incorrect data if the above conditions are met, however the second access will return correct data.

13.1.10 Write Access Limitation of I-MMU Registers

If an STXA instruction that targets an I-MMU register (e.g., ASI_IMMU_SFSR, ASI =0x50, VA=0x18) is executed and an I-MMU miss is taken prior to the programmer visible state being modified, it is possible for both the STXA and the I-MMU miss to attempt to update the targeted register at the exact same instant. An example is shown below:

0x25105e21ffc: stxa %g0, [%i0]0x50

0x25105e22000: IMMU miss

In this case, if the STXA instruction takes priority over the I-MMU miss, it will cause stale data to be stored in the I-MMU register.

A FLUSH, DONE, or RETRY is a special case needed after an internal ASI store that affects instruction accesses, (see *Instruction Memory Management Unit* on page 302). The usage is that the FLUSH, DONE, or RETRY should immediately follow the STXA.

There can be a case where the programmer may have inserted such an instruction after the STXA instruction, but the instruction was not processed prior to the I-MMU miss.

There are two ways of ensuring such a case does not occur:

1. Any code which modifies the I-MMU state should be locked down in the I-TLB to prevent the possibility of intervening TLB misses.
2. If suggestion (1) is not possible, the STXA and the subsequent FLUSH, DONE, or RETRY should be kept on the same 8 KB page, again preventing an intervening I-TLB miss.

Note – An *instruction_access_execption* by I-MMU parity error may cause this problem.

13.2 Data Memory Management Unit

The Data Memory Management Unit (D-MMU) conforms to the requirements set forth in the *UltraSPARC III Cu Processor User's Manual*. In particular, the D-MMU supports a 64-bit virtual address space, simplified protection encoding, multiple page sizes, and software D-TLB-miss processing. There is no hardware page table walk.

13.2.1 Virtual Address Translation

A 64-bit virtual address (VA) space is supported, with 43 bits of physical address (PA).

The UltraSPARC IV+ processor D-MMU consists of three Translation Lookaside Buffers (TLBs) as described in TABLE 13-15. There is no support for locked pages of sizes 32 MB or 256 MB. An attempt to install a 32 MB or 256 MB locked page will result in undefined and potentially erroneous translations.

TABLE 13-15 D-MMU TLBs

TLB name	TLB ID	Translating Page Size	Remark
T16	0	8 KB, 64 KB, 512 KB, 4 MB	16-entry fully-associative, both for locked and unlocked pages
T512_0	2	8 KB, 64 KB, 512 KB, 4 MB	First large D-TLB. 512-entry 2-way set-associative (256 entries per way), for unlocked pages
T512_1	3	8 KB, 64 KB, 32 MB, 256 MB	Second large D-TLB. 512-entry 2-way set-associative (256 entries per way), for unlocked pages

13.2.2 Two D-TLBs with Large Page Support

The UltraSPARC IV+ processor has three D-TLBs. The first 16-entry fully-associative D-TLB, the T16 remains the same as in the UltraSPARC III processor, with the exception of supporting 8 KB unlocked pages. All supported page sizes are described in TABLE 13-15. When a memory access is issued, its VA, Context, and PgSz are presented to the D-MMU. All three D-TLBs, T512_0, T512_1, and T16 are accessed in parallel.

Note – Unlike the UltraSPARC III processor, the UltraSPARC IV+ processor's T16 can support ***unlocked*** 8 KB pages to prevent dropping of a D-TLB fill of an unlocked 8 KB page when the large D-TLBs are programmed to non unlocked 8 KB pages.

When both large D-TLBs are configured with same page size, the behavior is like a single D-TLB with 1024-entry 4-way set-associative (256 entries per way).

Each T512's page size (PgSz) is programmable independently, one PgSz per context (primary/secondary/nucleus). Software can set the PgSz fields in `ASI_PRIMARY_CONTEXT_REG` and `ASI_SECONDARY_CONTEXT_REG` as described in TABLE 13-16 and TABLE 13-17.

TABLE 13-16 I-MMU and D-MMU Primary Context Register

Bit	Field	Description
[63:61]	N_pgsz0	Nucleus context's page size at the first large D-TLB (T512_0)
[60:58]	N_pgsz1	Nucleus context's page size at the second large D-TLB (T512_1)
[57:55]	N_pgsz_I	Nucleus context's page size at the first large I-TLB (iT512)
[54:23]	*Reserved*	Reserved for future implementation.
[24:22]	P_pgsz0	Primary context's page size at the first large D-TLB (T512_0)
[21:19]	P_pgsz1	Primary context's page size at the second large D-TLB (T512_1)
[18:16]	P_pgsz_I	Primary context's page size at the first large I-TLB (iT512)
[15:13]	*Reserved*	Reserved for future implementation.
[12:0]	PContext	Context identifier for the primary address space

TABLE 13-17 D-MMU Secondary Context Register

Bit	Field	Description
[63:22]	*Reserved*	Reserved for future implementation.
[21:19]	S_pgsz1	Secondary context's page size at the second large D-TLB (T512_1)
[18:16]	S_pgsz_0	Secondary context's page size at the first large D-TLB (T512_0)
[15:13]	*Reserved*	Reserved for future implementation.
[12:0]	SContext	Context identifier for the secondary address space

Page size bit encoding:
000 = 8 KB
001 = 64 KB
010 = 512 KB
011 = 4 MB
100 = 32 MB
101 = 256 MB

T512_0 undefined page size bit encoding: 100, 101
T512_1 undefined page size bit encoding: 010, 011

Note – Due to different page size support in the T512_0 and T512_1, the following bits in the D-MMU Primary Context Register and Secondary Context Register are reserved as 0: N_pgsz1[1], N_pgsz0[2], P_pgsz1[1], P_pgsz0[2], N_pgsz_I[2:1], S_pgsz1[1], S_pgsz0[2].

It is illegal to program primary and/or secondary context registers with page sizes that are not supported by the two T512 D-TLBs. Such incorrect page size programming can result in unexpected D-TLB parity error (*data_access_exception* trap) or *fast_data_access_MMU_miss* trap. It is also possible that these traps can recur and in some cases will lead to RED mode. A load operation (`ldxa` to appropriate ASI) targetting an illegally programmed context register will not return the actual value; instead, it will return one of the legal values. The only way to find out if illegal page sizes are programmed in the context register is to read the `ASI_DMMU_TAG_ACCESS_EXT` register on a *fast_data_access_MMU_miss* or a *data_access_exception* trap and inspect `ASI_DMMU_ACCSES_EXT` bits[21:19] for T512_1 page size and bits[18:16] for T512_0 page size.

13.2.2.1 D-TLB Access Operation

When a memory access instruction is issued, its VA, Context, and PgSz are used to access all 3 D-TLBs (T512_0, T512_1, and T16) in parallel. The fully-associative T16 only needs the VA and Context to CAM-match and output an entry, 1 out of 16. The proper VA bits are compared based on the page size bits of each T16 entry. Fast 3-bit encoding is used to define the 8 KB, 64 KB, 512 KB, and 4 MB page sizes.

Since the T512s are not fully-associative, indexing the T512 arrays require knowledge of the page size to properly select the VA bits to be used as the index as shown below:

if an 8 KB page is selected, array index = VA[20:13]
if a 64 KB page is selected, array index = VA[23:16]
if a 512 KB page is selected, array index = VA[26:19]

if a 4 MB page is selected, array index = VA[29:22]
if a 32 MB page is selected, array index = VA[32:25]
if a 256 MB page is selected, array index = VA[35:28]

The Context bits are used after the indexed entry comes out of each array bank/way, to qualify the context hit.

There are 3 possible Context numbers active in the processor, i.e., the primary (PContext field in `ASI_PRIMARY_CONTEXT_REG`), secondary (SContext field in `ASI_SECONDARY_CONTEXT_REG`), and nucleus (default to Context = 0). The Context register used to send to data to the D-MMU is determined based on the load/store's ASI encoding of primary/secondary/nucleus.

Since all 3 D-TLBs are accessed in parallel, software must guarantee that there are no duplicate (stale) entry hits. Most of this responsibility lies with the operating system software with the hardware providing some assistance to support full software control. A set of rules on the D-TLB replacement, demap and context switch must be followed to maintain consistent and correct behavior.

13.2.2.2 D-TLB Parity Protection

Both the T512_0 and T512_1 support parity protection for both the tag and data arrays, however, the T16 does not support parity protection. The D-MMU generates an odd-parity for the tag from a 60-bit parity-tree, and an odd-parity for the data from a 37-bit parity-tree upon a D-TLB replacement. The parities are calculated as follows:

Tag Parity = XOR(Size[2:0],Global,VA[63:21],Context[12:0])
Data Parity = XOR(NFO,IE,PA[42:13],CP,CV,E,P,W)

Note – The Valid bit is not included in the tag parity calculation.

The parity bits are available during the same cycle that the tag and data are sent to the D-TLB. The tag parity is written to bit[60] of the tag array while the data parity is written to bit[35] of the data array.

During the D-TLB translation, set-associative TLBs, T512_0 and T512_1, check the tag and data parities previously written by replacement. The tag and data parity errors are reported as a *data_access_exception* with the fault status recorded in the D-SFSR register. Fault Type 20_{16} (D-SFSR bit[12]) is valid when tag or data parity errors occur.

Note – The tag and data parities are checked even for invalid D-TLB entries.

When a trap is taken on a D-TLB parity error, the software needs to invalidate the corresponding entry and write that entry with good parity.

The tag or data parity errors are masked with the following conditions:

- Fully-associative TLB T16 is hit.
- The D-TLB parity enable bit is off. The D-TLB parity enable is controlled by bit[17] (DTPE) of the Data Cache Unit Control Register.
- The D-MMU enable bit is off. The D-MMU enable is controlled by bit[3] (DM) of the Data Cache Unit Control Register.

During the D-TLB demap operation, set-associative TLBs (T512_0 and T512_1) also check the tag and data parities. If a parity error is detected, the corresponding entry will be invalidated by the hardware regardless of whether a hit or miss has occurred. The demap operation will only clear the valid bit, but not the parity error. No *data_access_exception* is generated during demap operation.

While writing to the D-TLB using Data In Register (`ASI_DTLB_DATA_IN_REG`), the tag and data parities are generated by hardware. While writing to the D-TLB using the Data Access Register (`ASI_DTLB_DATA_ACCESS_REG`), the tag and data parities are generated by the hardware. While writing to the D-TLB using the D-TLB Diagnostic Register (`ASI_DTLB_DIAG_REG`), the tag and data parities can be supplied by the stored data. When using `ASI_SRAM_FAST_INIT`, all tag and data parity bits will be cleared.

While reading the D-TLB using the Data Access Register (`ASI_DTLB_DATA_ACCESS_REG`) or the D-TLB Diagnostic Register (`ASI_DTLB_DIAG_REG`), the tag and data parities are available in bit[46] and bit[47] of the D-TLB Diagnostic Registers.

During bypass ASIs, the D-TLB does not flag parity errors.

TABLE 13-18 summarizes the UltraSPARC IV+ processor D-MMU parity error behavior.

TABLE 13-18 D-MMU parity error behavior

Operation	Parity Enable (DCR register bit[17])	D-MMU Enable (DCU Ctrl register bit[3])	T512_0		T512_1		T16	Trap taken
			Data parity error	Tag parity error	Data parity error	Tag parity error	hit	
Translation	0	x	x	x	x	x	x	no trap taken
	1	0	x	x	x	x	x	no trap taken
	1	1	0	0	0	0	x	no trap taken
	1	1	1	x	x	x	1	no trap taken
	1	1	x	1	x	x	1	no trap taken
	1	1	x	x	1	x	1	no trap taken
	1	1	x	x	x	1	1	no trap taken
	1	1	1	x	x	x	0	data_access_exception
	1	1	x	1	x	x	0	data_access_exception
	1	1	x	x	1	x	0	data_access_exception
	1	1	x	x	x	1	0	data_access_exception
Demap	x	x	1	x	x	x	x	no trap taken
	x	x	x	1	x	x	x	no trap taken
	x	x	x	x	1	x	x	no trap taken
	x	x	x	x	x	1	x	no trap taken
Read	x	x	1	x	x	x	x	no trap taken
	x	x	x	1	x	x	x	no trap taken
	x	x	x	x	1	x	x	no trap taken
	x	x	x	x	x	1	x	no trap taken
Translation	0	x	x	x	x	x	x	no trap taken
	1	0	x	x	x	x	x	no trap taken

NOTE: An “x” in the table represents don’t-cares.

13.2.2.3 Rules for Setting the Same Page Size on Both the T512_0 and T512_1

When both the T512s are programmed to have identical page sizes, they would behave is as if both T512s were a *single* 4-way 1024-entry T512. During the TTE fill, if there is no invalid TLB entry to take, then the T512 selection and way selection are determined based on a random replacement policy using a new 10-bit LFSR (Linear Feedback Shift Register) pseudo-random generator. For 4-way pseudo-random selection, LFSR[1:0] bits (two least significant bits of LFSR) will be used. Please refer to TABLE 13-19.

TABLE 13-19 LFSR Bit Setting

LFSR[1:0]	TTE Fill To:
00	T512_0, way 0
01	T512_0, way 1
10	T512_1, way 0
11	T512_1, way 1

Note – All LFSRs (in the D-cache, I-cache, and TLBs) are initialized on power-on reset (Hard_POR) and system reset (Soft_POR).

This single LFSR is also shared by both the T512_0 and T512_1 when they have different page sizes. The least significant bit (LFSR[0]) is used for the entry replacement. It selects the same bank of both T512s, but only one of the T512 write-enables is eventually asserted at the TTE fill time.

The Demap Context is needed when the *same* context changes PgSz. During context-in, if the operating system software decides to change any PgSz setting of the T512_0 or T512_1 *differently* from last context-out of the *same* Context (e.g., was both 8 KB at context out, now 8 KB & 256 MB at context in), then the operating system software will perform a Demap Context operation first. This avoids remnant entries in the T512_0 or T512_1, which could cause a duplicate, possibly stale, hit.

13.2.2.4 D-TLB Automatic Replacement

A D-TLB-miss fast trap handler utilizes the automatic, hardware, replacement write using store `ASI_DTLB_DATA_IN_REG`.

When a D-TLB miss, *data_access_exception*, or *fast_data_access_protection* is detected, the hardware automatically saves the missing VA and context to the Tag Access Register (`ASI_DMMU_TAG_ACCESS`). The missing page size information of the T512_0 and T512_1 is captured into the Tag Access Extension Register, `ASI_DMMU_TAG_ACCESS_EXT` (see *D-TLB Tag Access Extension Registers* on page 331). This information is used during replacement. The hardware D-TLB replacement algorithm is as follows:

Note – “PgSz0” below is `ASI_DMMU_TAG_ACCESS_EXT[18:16]` bits corresponding to the page size of T512_0, and “PgSz1” is `ASI_DMMU_TAG_ACCESS_EXT[21:19]` bits corresponding to the page size of T512_1.

See below for the D-TLB Replacement Pseudocode:

```
if (TTE to fill is a locked page) //L bit is set
{
           fill TTE to T16;
} else if (both T512s have same page size)  //PgSz0 == PgSz1
  {
       if (TTE's Size != PgSz0)
           {
                fill TTE to T16;
           } else {
                if (one of the 4 same-index entries is invalid)
                {
                    fill TTE to an invalid entry with selection order of
                    (T512_0 way0, T512_0 way1, T512_1 way0, T512_1 way1)
                 } else
                   {
                      case (LFSR[1:0]) {
                        00: fill TTE to T512_0 way0;
                        01: fill TTE to T512_0 way1;
                        10: fill TTE to T512_1 way0;
                        11: fill TTE to T512_1 way1;
                      }
                }
           }
        } else {
            if (TTE's Size == PgSz0) {
                if (one of the 2 same-index entries is invalid) {
                    fill TTE to an invalid entry with selection order of
                    (T512_0 way0, T512_0 way1)
                } else {
                    case (LFSR[0]) {
                        0: fill TTE to T512_0 way0;
                        1: fill TTE to T512_0 way1;
                    }
                }
            } else if (TTE's Size == PgSz1) {
                if (one of the 2 same-index entries is invalid) {
                    fill TTE to an invalid entry with selection order of
                    (T512_1 way0, T512_1 way1)
                } else {
                    case (LFSR[0]) {
                        0: fill TTE to T512_1 way0;
                        1: fill TTE to T512_1 way1;
                    }
                }
            } else {
                fill TTE to T16;
            }
        }
```

13.2.2.5 Direct D-TLB Data Read/Write

As described in the *UltraSPARC III Cu Processor User's Manual*, each D-TLB can be directly written to using store `ASI_DTLB_DATA_ACCESS_REG` (ASI $5D_{16}$) instruction. Software typically uses this method for initialization and diagnostic.

Page size information is determined by bit[48], [62:61] of the TTE data (store data of `STXA ASI_DTLB_DATA_ACCESS_REG`). Direct access to the D-TLB is achieved by properly selecting the TLB ID and TLB entry fields of ASI virtual address as explained in *D-TLB Tag Access Extension Registers* on page 331.

It is not required to write the D-TLB Tag Access Extension Register (ASI_DMMU_TAG_ACCESS_EXT) with page size information since ASI_DTLB_DATA_ACCESS_REG gets the page size from the TTE data. But it is recommended that software writes the proper page size information to ASI_DMMU_TAG_ACCESS_EXT before writing to ASI_DTLB_DATA_ACCESS_REG.

13.2.2.6 D-TLB Tag Read Register

See *Tag Read Register* on page 333 for details about ASI_DTLB_TAG_READ_REG (ASI $5E_{16}$).

13.2.2.7 Demap Operation

For a demap-page in the large D-TLBs, the page size used to index the D-TLBs is derived based on the Context bits (primary/secondary/nucleus). The hardware will automatically select the proper PgSz bits based on the “Context” field (primary/secondary/nucleus) defined in ASI_DMMU_DEMAP (ASI $5F_{16}$). These two PgSz fields are used to properly index the T512_0 and T512_1.

Demap operations in the T16 are single cycle operations - all matching entries are demapped in one cycle. The Demap operations in the T512_0 and T512_1 are multi-cycle operations. The demap operation is done in parallel for both TLBs - one entry at a time for all 512 entries.

Note – When global pages are used (G = 1), any active page in a given T512 must have the same page size. When pages with G = 1 in a T512 have variety of page sizes, the T512 cannot index and locate the page correctly when trying to match the VA tag without the context number as a qualifier.

13.2.2.8 D-TLB Access Summary

TABLE 13-20 lists the D-MMU TLB Access Summary.

TABLE 13-20 D-MMU TLB Access Summary

Software Operation		Effect on MMU Physical Registers				
Load/ Store	**Register**	**TLB tag array**	**TLB data array**	**Tag Access**	**SFSR**	**SFAR**
Load	Tag Read	Contents returned. From entry specified by LDXA's/LDDFA's access	No effect	No effect	No effect	No effect
	Tag Access	No effect	No effect	Contents returned	No effect	No effect
	Data In	Trap with data_access_exception				
	Data Access	No effect	Contents returned. From entry specified by LDXA's/LDDFA's access	No effect	No effect	No effect
	SFSR	No effect	No effect	No effect	Contents returned	No effect
	SFAR	No effect	No effect	No effect	No effect	Contents returned
Store	Tag Read	Trap with data_access_exception				
	Tag Access	No effect	No effect	Written with store data	No effect	No effect
	Data In	TLB entry determined by replacement policy written with contents of Tag Access Register	TLB entry determined by replacement policy written with store data	No effect	No effect	No effect
	Data Access	TLB entry specified by STXA's/STDFA's address written with contents of Tag Access Register	TLB entry specified by STXA's/ STDFA's address written with store data	No effect	No effect	No effect
	SFSR	No effect	No effect	No effect	Written with store data	No effect
	SFAR	No effect	No effect	No effect	No effect	Written with store data
TLB miss		No effect	No effect	Written with VA and context of access	Written with fault status of faulting instruction and page sizes at faulting context for two 2 way set associative TLB	Written with virtual address of faulting instruction

13.2.3 Translation Table Entry (TTE)

The Translation Table Entry (TTE) holds the information for a single page mapping. The TTE is divided into two 64-bit words representing the tag and data of the translation. Just as in a hardware cache, the tag is used to determine whether there is a hit in the TSB; if there is a hit, then the data is fetched by software.

The configuration of the TTE is described in TABLE 13-21 (see also the *UltraSPARC III Cu Processor User's Manual*).

Note – All bootbus addresses must be mapped as side-effect pages with the TTE E bit set.

TABLE 13-21 TTE Data Field Descriptions ***(1 of 2)***

Bit	Field	Description
[63]	V	See the *UltraSPARC III Cu Processor User's Manual*.
[62:61]	Size[1:0]	Bit [62:61] represent the least significant 2 bits of the page size. Size[2] concatenated with Size[1:0] encodes the page size for D-TLB as follows: 000 = 8 KB 001 = 64 KB 010 = 512 KB 011 = 4 MB 100 = 32 MB 101 = 256 MB
[60]	NFO	See the *UltraSPARC III Cu Processor User's Manual*.
[59]	IE	See the *UltraSPARC III Cu Processor User's Manual*.
[58:50]	Soft2	See the *UltraSPARC III Cu Processor User's Manual*.
[49]	*Reserved*	Reserved for future implementation.

TABLE 13-21 TTE Data Field Descriptions *(2 of 2)*

Bit	Field	Description
[48]	Size[2]	Bit[48] is the most significant bit of the page size and is concatenated with bits [62:61].
[47:43]	*Reserved*	Reserved for future implementation.
[42:13]	PA	The physical page number. Page offset bits for larger page sizes (PA[15:13], PA[18:13], PA[21:13], PA[24:13] and PA[27:13] for 64 KB, 512 KB, 4 MB, 32 MB, and 256 MB pages, respectively) are stored in the TLB and returned for a Data Access read but are ignored during normal translation. When page offset bits for larger page sizes are stored in the TLB on UltraSPARC IV+ processor, the data returned from those fields by a Data Access read are the data previously written to them.
[12:7]	Soft	See the *UltraSPARC III Cu Processor User's Manual*.
[6]	L	If the lock bit is set, then the TTE entry will be locked down when it is loaded into the TLB; that is, if this entry is valid, it will not be replaced by the automatic replacement algorithm invoked by an ASI store to the Data In Register. The lock bit has no meaning for an invalid entry. Arbitrary entries can be locked down in the TLB. Software must ensure that at least one entry is not locked when replacing a TLB entry; otherwise, a locked entry will be replaced. Only the 16-entry, fully-associative TLB (T16) can hold locked pages. In the UltraSPARC IV+ processor, 32 MB and 256 MB can't be locked. Set-associative TLB (T512_0 and T512_1) can hold unlocked pages of all sizes.
[5], [4]	CP, CV	The cacheable-in-physically-indexed-cache bit and cacheable-in-virtually-indexed-cache bit determine the placement of data in the caches. The UltraSPARC IV+ processor fully implements the CV bit. The following table describes how CP and CV control cacheability in specific UltraSPARC IV+ processor caches. **TABLE 13-22 Meaning of TTE** **Cacheable (CP, CV)** / **Meaning of TTE when placed in: D-TLB (Data Cache VA-indexed)** 00 — Non-cacheable 01 — Cacheable P-cache 10 — Cacheable all caches except D-cache 11 — Cacheable all caches The MMU does not operate on the cacheable bits but merely passes them through to the cache subsystem. Note: When defining alias page attributes, care should be taken to avoid the following (CP, CV) combinations: Set 1: VA = VA1, PA = PA1, CP = 1, CV = 1 Set 2: VA = VA2, PA = PA1, CP = 1, CV = 0 Aliasing with the above attributes will result in stale value in the D-cache for VA1 after a write to VA2. A write to VA2 will only update the W-cache and not the D-cache.
[3]	E	See the *UltraSPARC III Cu Processor User's Manual*.
[2]	P	See the *UltraSPARC III Cu Processor User's Manual*.
[1]	W	See the *UltraSPARC III Cu Processor User's Manual*.
[0]	G	See the *UltraSPARC III Cu Processor User's Manual*.

13.2.4 Hardware Support for TSB Access

The MMU hardware provides services to allow the TLB-miss handler to efficiently reload a missing TLB entry for an 8 KB or 64 KB page. These services include:

- Formation of TSB Pointers, based on the missing virtual address and address space
- Formation of the TTE Tag Target used for the TSB tag comparison
- Efficient atomic write of a TLB entry with a single store ASI operation
- Alternate globals on MMU-signaled traps

Please refer to the *UltraSPARC III Cu Processor User's Manual* for additional details.

13.2.4.1 Typical TLB Miss/Refill Sequence

A typical TLB-miss and TLB-refill sequence is the following:

1. A D-TLB miss causes a *fast_data_access_MMU_miss* exception.
2. The appropriate TLB-miss handler loads the TSB Pointers and the TTE Tag Target with loads from the MMU registers.
3. Using this information, the TLB miss handler checks to see if the desired TTE exists in the TSB. If so, the TTE data are loaded into the TLB Data In Register to initiate an atomic write of the TLB entry chosen by the replacement algorithm.
4. If the TTE does not exist in the TSB, then the TLB-miss handler jumps to the more sophisticated, and slower, TSB miss handler.

The virtual address used in the formation of the pointer addresses comes from the Tag Access Register, which holds the virtual address and context of the load or store responsible for the MMU exception. See *Translation Table Entry (TTE)* on page 328.

Note – There are no separate physical registers in hardware for the pointer registers; rather, they are implemented through a dynamic reordering of the data stored in the Tag Access and the TSB registers.

The hardware provides pointers for the most common cases of either 8 KB and 64 KB page miss processing. These pointers give the virtual addresses where the 8 KB and 64 KB TTEs are stored if either is present in the TSB.

The TSB_Size field, *n*, of the TSB register, ranges from 0 to 7. Note that TSB_Size refers to the size of each TSB when the TSB is split. The symbol designates concatenation of bit vectors and ⊕ indicates an exclusive-or operation.

For a shared TSB (TSB register split field = 0):

$$8K\_PTR = TSB\_Base[63:13+n] \oplus TSB\_Extension[63:13+n] \,\|\, VA[21+n:13] \,\|\, 0000$$

$$64K\_PTR = TSB\_Base[63:13+n] \oplus TSB\_Extension[63:13+n] \,\|\, VA[24+n:16] \,\|\, 0000$$

For a split TSB (TSB register split field = 1):

$$8K\_PTR = TSB\_Base[63:14+n] \oplus TSB\_Extension[63:14+n] \,\|\, 0 \,\|\, VA[21+n:13] \,\|\, 0000$$

$$64K\_PTR = TSB\_Base[63:14+n] \oplus TSB\_Extension[63:14+n] \,\|\, 1 \,\|\, VA[24+n:16] \,\|\, 0000$$

The TSB Tag Target is formed by aligning the missing access VA (from the Tag Access Register) and the current context to positions found above in the description of the TTE tag, allowing a simple `XOR` instruction for TSB hit detection.

13.2.5 Faults and Traps

On a *mem_address_not_aligned* trap that occurs during a `JMPL` or `RETURN` instruction, the UltraSPARC IV+ processor updates the D-SFSR register with the FT field set to 0 and updates the D-SFAR register with the fault address. For details, please refer to the *UltraSPARC III Cu Processor User's Manual*.

13.2.6 Reset, Disable, and RED_state Behavior

Please refer to the *UltraSPARC III Cu Processor User's Manual* for general details.

Note – While the D-MMU is disabled and the default CV bit in the Data Cache Unit Control Register is set to 0, data in the D-cache can be accessed only through the load and store alternates to the internal D-cache access ASI. Normal loads and stores bypass the D-cache. Data in the D-cache cannot be accessed by load or store alternates that use `ASI_PHYS_EC` or `ASI_PHYS_EC_LITTLE`. Other caches are physically indexed or are still accessible.

13.2.7 Internal Registers and ASI Operations

Please refer to the *UltraSPARC III Cu Processor User's Manual* for details.

13.2.7.1 D-TLB Tag Access Extension Registers

Tag Access Extension Register

ASI 58_{16}, VA[63:0]==60_{16},
Name: `ASI_DMMU_TAG_ACCESS_EXT`
Access: RW

The D-TLB Tag Access Extension Register saves the missed page size information in the T512_0 and T512_1.

TABLE 13-23 DTLB Tag Access Extension Register Description

Bit	Field	R/W	Description
[21:19]	pgsz1	RW	Page size of T512_1, pgsz1[1] is reserved as 0.
[18:16]	pgsz0	RW	Page size of T512_0, pgsz0[2] is reserved as 0.

Note – With the saved page sizes, hardware pre-computes in the background the index to the T512_0 and T512_1 for the TTE fill. When the TTE data arrive, only one write is enabled to the T512_0, T512_1, or T16.

13.2.7.2 D-TLB Data In, Data Access, and Tag Read Registers

Data In Register

ASI $5C_{16}$, VA[63:0]==00_{16},
Name: `ASI_DTLB_DATA_IN_REG`
Access: W

Writes to the TLB Data In register require the virtual address to be set to 0.

Note – DTLB Data In register is used when fast_data_access_MMU_miss trap is taken to fill DTLB based on replacement algorithm. Other than fast_data_access_MMU_miss trap, an ASI store to DTLB Data In register may replace an unexpected entry in the DTLB even if the entry is locked. The entry that gets updated depends on the state of the Tag Access Extension Register, LFSR bits and the TTE page size in the store data. If nested fast_data_access_MMU_miss happens, DTLB Data In register will not work.

Data Access Register

ASI $5D_{16}$, VA[63:0]==00_{16} - $30FF8_{16}$,
Name: `ASI_DTLB_DATA_ACCESS_REG`
Access: RW

Virtual Address Format

The virtual address format of the D-TLB Data Access register is described in TABLE 13-24

TABLE 13-24 D-TLB Data Access register

Bit	Field	Description	R/W
[63:19]	*Reserved*	Reserved for future implementation.	
[18]	Mandatory value	Should be 0.	
[17:16]	TLB ID	The TLB to access as defined below. 0 : T16 2 : T512_0 3 : T512_1	RW
[15:12]	*Reserved*	Reserved for future implementation.	
[11:3]	TLB Entry	The TLB entry number to be accessed, in the range 0–511. Not all TLBs will have all 512 entries. All TLBs regardless of size are accessed from 0 to $N - 1$, where N is the number of entries in the TLB. For the T512s, bit[11] is used to select either way0 or way1, and bit[10:3] is used to access the specified index. For the T16, only bit[6:3] is used to access one of 16 entries.	RW
[2:0]	Mandatory value	Should be 0.	

Data Format

The D-TLB Data Access Register uses the TTE data format with the addition of parity information in the T512s as described in TABLE 13-21 for the T16 fields and TABLE 13-31 for the T512 fields.

Note – When writing to the T512_0 and T512_1, Size[2](bit[48]) for the T512_0 and Size[1](bit[62]) for the T512_1 are masked and forced to zero by the hardware. The Data Parity and Tag Parity bits, DP(bit[47]) and TP(bit[46]), are also masked by the hardware during writes. The parity bits are calculated by the hardware and written to the corresponding D-TLB entry when replacement occurs as mentioned in *D-TLB Parity Protection* on page 322.

Tag Read Register

ASI $5E_{16}$, VA[63:0]==00_{16} - $20FF8_{16}$,
Name: `ASI_DTLB_TAG_READ_REG`
Access: R

Virtual Address Format

Note – Bit[2:0] is 0.

Data Format

The data format of Tag Read Register is described in TABLE 13-25 and TABLE 13-26.

TABLE 13-25 Tag Read Register Data Format Description for T16

Bit	Field	Description	R/W
[63:13]	VA	The 51-bit virtual page number. In the fully associative TLB, page offset bits for larger page sizes are stored in TLB; that is VA[15:13], VA[18:13], and VA[21:13] for 64 KB, 512 KB, and 4 MB pages, respectively. These values are ignored during normal translation.	R
[12:0]	Context	The 13-bit context identifier.	R

TABLE 13-26 Tag Read Register Data Format Description for T512

Bit	Field	Description	R/W
[63:21]	VA	VA[63:21] stored in T512 Tag array.	R
[20:13]	Index	Accessed entry index which is given by data access address[10:3].	R
[12:0]	Context	The 13-bit context identifier.	R

Note – If any memory access instruction that misses the D-TLB is followed by a diagnostic read access (`LDXA` from `ASI_DTLB_DATA_ACCESS_REG`, i.e., ASI 0x5d) from fully associative TLBs and the target TTE has page size set to 64 KB, 512 KB, or 4 MB, the data returned from the TLB will be incorrect.

13.2.7.3 D-MMU TSB (D-TSB) Base Registers

ASI 58_{16}, VA[63:0]==28_{16},
Name: ASI_DMMU_TSB_BASE
Access: RW

The D-MMU TSB Base Register is described in TABLE 13-27.

TABLE 13-27 TSB Base Register Description

Bit	Field	Description	R/W
[63:13]	TSB_Base	See the *UltraSPARC III Cu Processor User's Manual*.	RW
[12]	Split	See the *UltraSPARC III Cu Processor User's Manual*.	RW
[11:3]	*Reserved*	Reserved for future implementation.	
[2:0]	TSB_Size	The UltraSPARC IV+ processor implements a 3-bit TSB_Size field. The number of entries in the TSB (or each TSB if split) = 512×2^3.	RW

13.2.7.4 D-MMU TSB (D-TSB) Extension Registers

ASI 58_{16}, VA[63:0]==48_{16},
Name: ASI_DMMU_TSB_PEXT_REG
Access: RW

ASI 58_{16}, VA[63:0]==50_{16},
Name: ASI_DMMU_TSB_SEXT_REG
Access: RW

ASI 58_{16}, VA[63:0]==58_{16},
Name: ASI_DMMU_TSB_NEXT_REG
Access: RW

Please refer to the *UltraSPARC III Cu Processor User's Manual* for information on the TSB Extension Registers. The TSB registers are defined as follows:

In the UltraSPARC IV+ processor, the TSB_Hash (bits [11:3] of the extension registers) are exclusive-ORed with the calculated TSB offset to provide a "hash" into the TSB. Changing the TSB_Hash field on a per-process basis minimizes the collision of TSB entries between different processes.

13.2.7.5 D-MMU Synchronous Fault Status Registers (D-SFSR)

ASI 58_{16}, VA[63:0]==18_{16},
Name: ASI_DMMU_SFSR
Access: RW

D-MMU SFSR is described in TABLE 13-28.

TABLE 13-28 D-SFSR Bit Description

Bit	Field	Description	R/W
[63:25]	*Reserved*	Reserved for future implementation.	
[24]	NF	Set if the faulting instruction is a nonfaulting load (a load to `ASI_NOFAULT`)	RW
[23:16]	ASI	Records the 8-bit ASI associated with the faulting instruction. This field is valid for all traps in which the FV bit is set.	RW
[15]	TM	D-TLB miss.	RW
[14]	*Reserved*	Reserved for future implementation.	
[13:7]	FT	Specifies the exact condition that caused the recorded fault, according to following this table. In the D-MMU, the Fault Type field FT[5:0] is valid only for data_access_exception faults and invalid for fast_data_access_MMU_miss; FT[6] is valid only for fast_data_access_MMU_miss and invalid for data_access_exception. There is no ambiguity in all other MMU trap cases. The hardware does not priority-encode the bits set in the fault type register; that is, multiple bits can be set. Note that, when a D-TLB parity error occurs (FT[5] = 1), the other FT bits (FT[6] and FT[4:0]) are undefined.	RW
[6]	E	Side-effect bit. Associated with the faulting data access or FLUSH instruction. Set by translating ASI accesses that are mapped by the TLB with the E bit set and bypass ASIs 15_{16} and $1D_{16}$. The E-bit is undefined after a D-TLB parity error. All other cases that update the SFSR (including bypass or internal ASI accesses) set the E bit to 0.	RW
[5:4]	CT	Context Register selection. The context is set to 11_2 when the access does not have a translating ASI.	RW
[3]	PR	Privilege bit. Set if the faulting access occurred while in privileged mode. This filed is valid for all traps in which FV bit is set.	RW
[2]	W	Write bit. Set if the faulting access indicated a data write operation (a store or atomic load/store instruction).	RW
[1]	OW	Overwrite bit. When the D-MMU detects a fault, the Overwrite bit is set to 1 if the FV bit has not been cleared from a previous fault; otherwise, it is set to 0.	RW
[0]	FV	Fault Valid bit. Set when the D-MMU detects a fault; it is cleared only on an explicit ASI write of 0 to SFSR. When the FV bit is not set, the values of the remaining fields in the SFSR and SFAR are undefined for traps.	RW

TABLE 13-29 D-MMU Synchronous Fault Status Register FT (Fault Type) Field

I/D	FT[6:0]	Description
D	01_{16}	Privileged violation.
D	02_{16}	Speculative Load instruction to page marked with E bit. This bit is 0 for internal ASI access.
D	04_{16}	Atomic (including 128-bit atomic load) to page marked non-cacheable.
D	08_{16}	Illegal LDA/STA ASI value, VA,RW, or size. Does not include cases where 02_{16} and 04_{16} are set.
D	10_{16}	Access other than nonfaulting load to page marked NFO. This bit is 0 for internal ASI accesses.
D	20_{16}	D-TLB tag or data parity error.
D	40_{16}	D-TLB miss with prefetch instruction.

Note – The NF, ASI, TM, FT, E, CT, PR, and W fields of the D-SFSR are undefined for *privileged_action* trap caused by an `rd` instruction.

13.2.7.6 D-TLB Diagnostic Register

ASI $5D_{16}$, VA[63:0]==40000_{16} - $70FF8_{16}$,
Name: `ASI_DTLB_DIAG_REG`
Access: RW

Virtual Address Format

Note – Bit[18] and bit[2:0] are set to 1 and 0 respectively.

Data Format

The format for the Diagnostic Register of the T16 is described below TABLE 13-30. The format for the Diagnostic Register of the T512s, as described in TABLE 13-31, uses the TTE data format with addition of parity information.

TABLE 13-30 TTE Data Format

Bit	Field	Description	R/W
[63:7]	*Reserved*	Reserved for future implementation.	
[6]	LRU	The LRU bit in the CAM, read-write.	RW
[5:3]	CAM SIZE	The 3-bit page size field from the CAM, read-only.	R
[2:0]	RAM SIZE	The 3 bit page size field from the RAM, read-only.	R

TABLE 13-31 D-TLB Diagnostic Register of T512_0 and T512_1 ***(1 of 2)***

Bit	Field	Description	R/W
[63]	V	See the *UltraSPARC III Cu Processor User's Manual*.	
[62:61]	Size[1:0]	Encode page size bits.	
[60]	NFO	See the *UltraSPARC III Cu Processor User's Manual*.	
[59]	IE	See the *UltraSPARC III Cu Processor User's Manual*.	
[58:50]	Soft2	See the *UltraSPARC III Cu Processor User's Manual*.	
[49]	*Reserved*	Reserved for future implementation.	
[48]	Size[2]	Always 0.	
[47]	DP	Data parity bit.	RW

TABLE 13-31 D-TLB Diagnostic Register of T512_0 and T512_1 *(2 of 2)*

Bit	Field	Description	R/W
[46]	TP	Tag parity bit.	RW
[45:43]	*Reserved*	Reserved for future implementation.	
[42:13]	PA	Physical page number.	
[12:7]	Soft	See the *UltraSPARC III Cu Processor User's Manual*.	
[6]	L	Lock bit.	
[5], [4]	CP, CV	Cacheable bit.	
[3]	E	See the *UltraSPARC III Cu Processor User's Manual*.	
[2]	P	See the *UltraSPARC III Cu Processor User's Manual*.	
[1]	W	See the *UltraSPARC III Cu Processor User's Manual*.	
[0]	G	See the *UltraSPARC III Cu Processor User's Manual*.	

Note – See TABLE 13-5 for a detailed description of the fields.

An ASI store to the D-TLB Diagnostic Register initiates an internal atomic write to the specified TLB entry. The Tag portion is obtained from the Tag Access Register (`ASI_DMMU_TAG_ACCESS`), the Data portion is obtained from the store data.

An ASI load from the TLB Diagnostic Register (`ASI_DTLB_DIAG_REG`) initiates an internal read of the Tag and Data. The Tag portion is discarded, the Data portion is returned.

Note – If any memory access instruction that misses the D-TLB is followed by a diagnostic read access (`LDXA` from `ASI_DTLB_DATA_ACCESS_REG`, i.e., ASI 0x5d) from fully associative TLBs and the target TTE has page size set to 64 KB, 512 KB, or 4 MB, the data returned from the TLB will be incorrect.

13.2.8 Translation Lookaside Buffer Hardware

13.2.8.1 TLB Replacement Policy

On an automatic replacement write to the TLB, the D-MMU picks the entry to write. The rules used for picking the entry to write are given in detail in *D-TLB Automatic Replacement* on page 324. If replacement is directed to the fully associative TLB, then the following alternatives are evaluated:

a. The first invalid entry is replaced (measuring from entry 0). If there is no invalid entry, then:

b. The first unused, unlocked (LRU, but clear) entry will be replaced (measuring from entry 0). If there is no unused unlocked entry, then:

c. All used bits are reset, and the process is repeated from Step b.
The replacement operation is undefined if all entries in the fully associative TLB have their lock bit set.

For the 2-way set associative TLBs, a pseudo-random replacement algorithm is used to select a way.

INDEX