THE NETWORK IS THE COMPUTER™

OpenSPARC T2™ Supplement

to the *UltraSPARC Architecture 2007*

Draft D1.4.2, 01 Aug 2007

Privilege Levels: *Hyperprivileged, Privileged, and Nonprivileged*

Distribution: Public

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 650-960-1300

Part No: 950-5556-01
Revision: Draft 1.4, 01 Aug 2007

Contents

CHAPTER 1

OpenSPARC T2 Basics

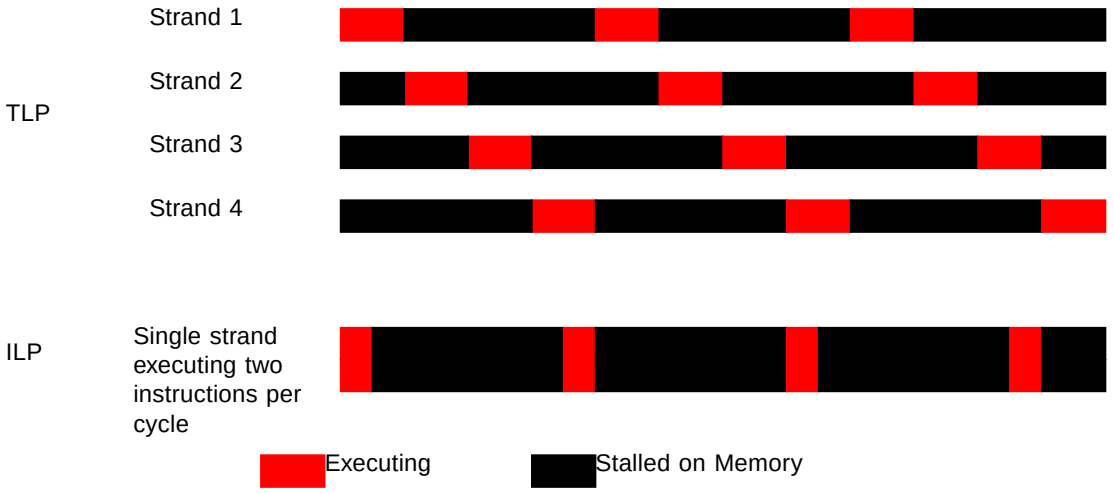
1.1 Background

OpenSPARC T2 is the follow-on chip multi-threaded (CMT) processor to the highly successful OpenSPARC T1 processor. The OpenSPARC T1 product line fully implements Sun's Throughput Computing initiative for the horizontal system space. Throughput Computing is a technique that takes advantage of the thread-level parallelism that is present in most commercial workloads. Unlike desktop workloads, which often have a small number of threads concurrently running, most commercial workloads achieve their scalability by employing large pools of concurrent threads.

Historically, microprocessors have been designed to target desktop workloads, and as a result have focused on running a single thread as quickly as possible. Single thread performance is achieved in these processors by a combination of extremely deep pipelines (over 20 stages in Pentium 4) and by executing multiple instructions in parallel (referred to as instruction-level parallelism or ILP). The basic tenet behind Throughput Computing is that exploiting ILP and deep pipelining has reached the point of diminishing returns, and as a result current microprocessors do not utilize their underlying hardware very efficiently. For many commercial workloads, the processor will be idle most of the time waiting on memory, and even when it is executing it will often be able to only utilize a small fraction of its wide execution width. So rather than building a large and complex ILP processor that sits idle most of the time, a number of small, single-issue processors that employ multithreading are built in the same chip area. Combining multiple processors on a single chip with

multiple strands per processor, allows very high performance for highly threaded commercial applications. This approach is called thread-level parallelism (TLP), and the difference between TLP and ILP is shown in the FIGURE 1-1.

FIGURE 1-1 Differences Between TLP and ILP



The memory stall time of one strand can often be overlapped with execution of other strands on the same processor, and multiple processors run their strands in parallel. In the ideal case, shown in FIGURE 1-1, memory latency can be completely overlapped with execution of other strands. In contrast, instruction-level parallelism simply shortens the time to execute instructions and does not help much in overlapping execution with memory latency.[1]

Given this ability to overlap execution with memory latency, why don't more processors utilize TLP? The answer is that designing processors is a mostly evolutionary process, and the ubiquitous deeply pipelined, wide ILP processors of today are the evolutionary outgrowth from a time when the processor was the bottleneck in delivering good performance. With processors capable of multiple GHz clocking, the performance bottleneck has shifted to the memory and I/O subsystems, and TLP has an obvious advantage over ILP for tolerating the large I/O and memory latency prevalent in commercial applications. Of course, every architectural technique has its advantages and disadvantages. The one disadvantage to employing TLP over ILP is that execution of a single thread will be slower on the TLP processor than an ILP processor. With processors running well over a GHz, a strand capable of executing only a single instruction per cycle is fully capable of completing tasks in the time required by the application, making this disadvantage a nonissue for nearly all commercial applications.

1. Processors that employ out-of-order ILP can overlap some memory latency with execution. However, this overlap is typically limited to shorter memory latency events such as L1 cache misses that hit in the L2 cache. Longer memory latency events such as main memory accesses are rarely overlapped to a significant degree with execution by an out-of-order processor.

1.2 OpenSPARC T2 Overview

OpenSPARC T2 is a single chip multi-threaded (CMT) processor. OpenSPARC T2 contains eight SPARC physical processor cores. Each SPARC physical processor core has full hardware support for eight strands, two integer execution pipelines, one floating-point execution pipeline, and one memory pipeline. The floating-point and memory pipelines are shared by all eight strands. The eight strands are hard-partitioned into two groups of four, and the four strands within a group share a single integer pipeline. While all eight strands run simultaneously, at any given time at most two strands will be active in the physical core, and those two strands will be issuing either a pair of integer pipeline operations, an integer operation and a floating-point operation, an integer operation and a memory operation, or a floating-point operation and a memory operation. Strands are switched on a cycle-by-cycle basis between the available strands within the hard-partitioned group of four using a least recently issued priority scheme. When a strand encounters a long-latency event, such as a cache miss, it is marked unavailable and instructions will not be issued from that strand until the long-latency event is resolved. Execution of the remaining available strands will continue while the long-latency event of the first strand is resolved.

Each SPARC physical core has a 16 KB, 8-way associative instruction cache (32-byte lines), 8 Kbytes, 4-way associative data cache (16-byte lines), 64-entry fully-associative instruction TLB, and 128-entry fully associative data TLB that are shared by the eight strands. In addition, each SPARC physical core has a cryptography (stream processing) unit that is controlled by processor loads and stores but executes as an independent coprocessor. The eight SPARC physical cores are connected through a crossbar to an on-chip unified 4 Mbyte, 16-way associative L2 cache (64-byte lines). The L2 cache is banked eight ways to provide sufficient bandwidth for the eight SPARC physical cores. The L2 cache connects to four on-chip DRAM controllers, which directly interface to a pair of fully buffered DIMM (FBD) channels. In addition, an on-chip PCI-EX controller, two 1-Gbit/10-Gbit Ethernet MACs, and several on-chip I/O-mapped control registers are accessible to the SPARC physical cores. Traffic from the PCI-EX port coherently interacts with the L2 cache.

A block diagram of the OpenSPARC T2 chip is shown in FIGURE 1-2.

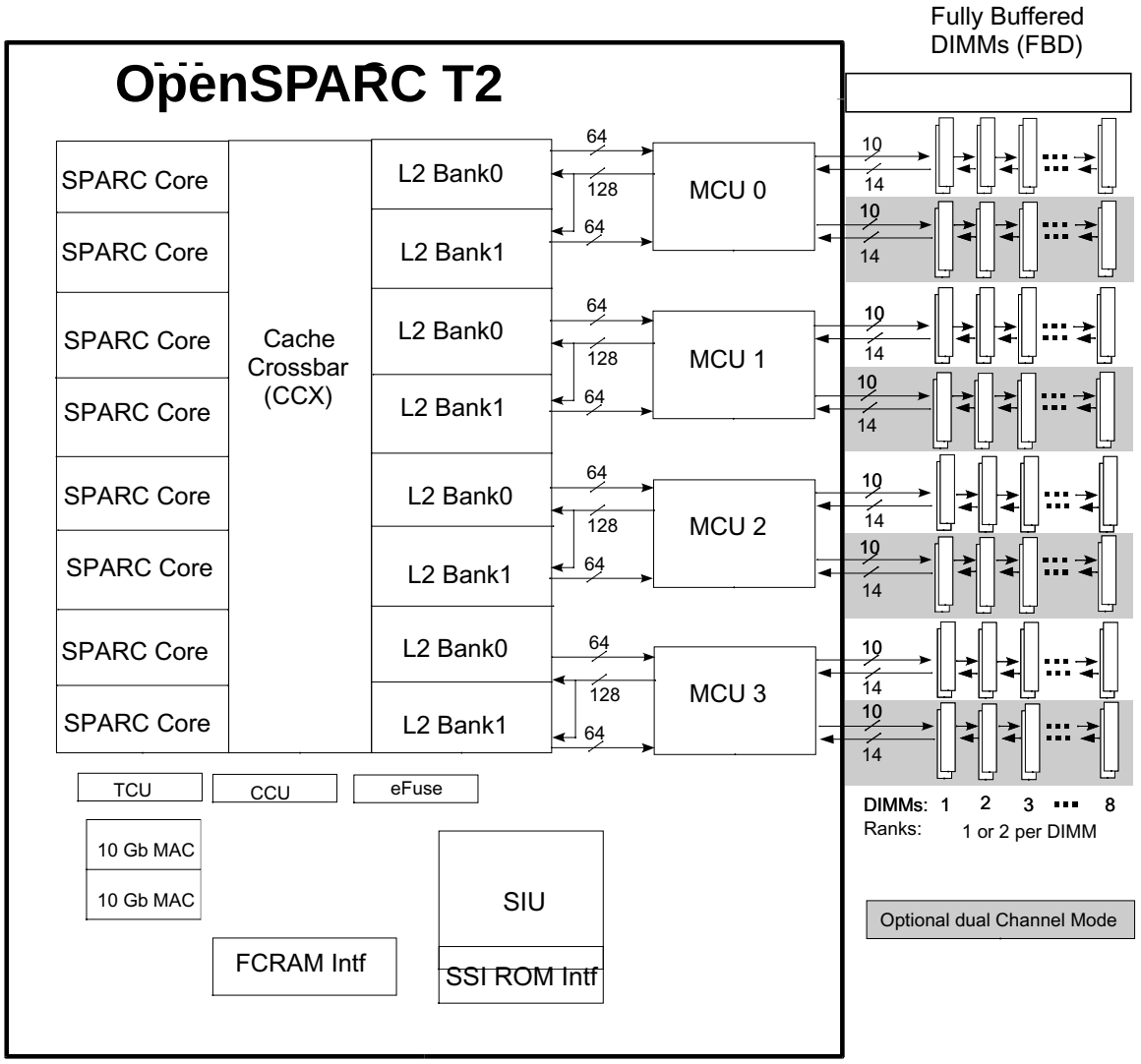


FIGURE 1-2 OpenSPARC T2 Chip Block Diagram

1.3 OpenSPARC T2 Components

This section describes each component in OpenSPARC T2.

1.3.1 SPARC Physical Core

Each SPARC physical core has hardware support for eight strands. This support consists of a full register file (with eight register windows) per strand, with most of the ASI, ASR, and privileged registers replicated per strand. The eight strands share the instruction and data caches and TLBs. An auto-demap feature is included with the TLBs to allow the multiple strands to update the TLB without locking.

A single floating-point unit is shared by all eight strands within a SPARC physical core. The shared floating-point unit is sufficient for most commercial applications which typically have less than 1% of their instructions being a floating-point operation.

1.3.2 L2 Cache

The L2 cache is banked eight ways. To provide for better partial-die recovery, OpenSPARC T2 can also be configured in 4-bank and 2-bank modes (with 1/2 and 1/4 the total cache size respectively). Bank selection based on physical address bits 8:6 for 8 banks, 7:6 for 4 banks, and 6 for 2 banks. The cache is 4 Mbytes, 16-way set associative. The line size is 64 bytes. Unloaded access time is 26 cycles for an L1 data cache miss and 24 cycles for an L1 instruction cache miss.

1.3.3 Memory Controller Unit (MCU)

OpenSPARC T2 has four MCUs, one for each memory branch with a pair of L2 banks interacting with exactly one DRAM branch. The branches are interleaved based on physical address bits 7:6, and support 1–16 DDR2 DIMMs. Each memory branch is two FBD channels wide. A branch may use only one of the FBD channels in a reduced power configuration.

Each DRAM branch operates independently and can have a different memory size and a different kind of DIMM (for example, a different number of ranks or different CAS latency). Software should not use address space larger than four times the lowest memory capacity in a branch because the cache lines are interleaved across branches. The DRAM controller frequency is the same as that of the DDR (Double Data Rate) data buses, which is twice the DDR frequency. The FBDIMM links run at six times the frequency of the DDR data buses.

1.3.4 Noncacheable Unit (NCU)

The NCU performs an address decode on I/O-addressable transactions and directs them to the appropriate block (for example, DMU, CCU). In addition, the NCU maintains the register status for external interrupts.

1.3.5 System Interface Unit (SIU)

The System Interface Unit connects the DMU and L2 Cache. SIU is the L2 Cache access point for the Network subsystem. The SIU-L2 Cache interface is also the ordering point for PCI-Express ordering rule.

1.3.6 SSI ROM Interface (SSI)

OpenSPARC T2 has a 50 Mb/s serial interface (SSI), which connects to an external field-programmable gate array (FPGA) that interfaces to the boot ROM. In addition, the SSI supports PIO accesses across the SSI, thus supporting optional Control and Status registers (CSRs) or other interfaces within the FPGA.

CHAPTER 2

Data Formats

Data formats supported by OpenSPARC T2 are described in the *UltraSPARC Architecture 2007* specification.

CHAPTER 3

Registers

3.1 Ancillary State Registers (ASRs)

This chapter discusses the OpenSPARC T2 ancillary state registers. TABLE 3-1 summarizes and defines these registers.

TABLE 3-1 Summary of OpenSPARC T2 Ancillary State Registers

Address	ASR Name	Access	priv	Replicated per-Strand	Description
00_{16}	Y	RW	N	Y	Y Register
01_{16}	*Reserved*	—			Any access causes a *illegal_instruction* trap
02_{16}	CCR	RW	N	Y	Condition Code register
03_{16}	ASI	RW	N	Y	ASI register
04_{16}	TICK	RW	Y[1]	Partially	TICK register
05_{16}	PC	RO[2]	N	Y	Program counter
06_{16}	FPRS	RW	N	Y	Floating-Point Registers Status register
07_{16}–$0E_{16}$	*Reserved*	-		—	Any access causes an *illegal_instruction* trap
$0F_{16}$	(MEMBAR, STBAR, SIR)	—	N	—	Instruction opcodes only, not an actual ASR.
10_{16}	PCR	RW	Y[3]	Y	Performance counter control register
11_{16}	PIC	RW	Y[4]	Y	Performance instrumentation counter
12_{16}	*Reserved*	—			Any access causes an *illegal_instruction* trap
13_{16}	GSR	RW	N	Y	General Status register

TABLE 3-1 Summary of OpenSPARC T2 Ancillary State Registers *(Continued)*

Address	ASR Name	Access	priv	Replicated per-Strand	Description
14_{16}	SOFTINT_SET	W	Y[5]	Y	Set bit in Soft Interrupt register
15_{16}	SOFTINT_CLR	W	Y[5]	Y	Clear bit in Soft Interrupt register
16_{16}	SOFTINT	RW	Y[3]	Y	Soft Interrupt register
17_{16}	TICK_CMPR	RW	Y[3]	Y	TICK Compare register
18_{16}	STICK	RW	Y[6]	Partially	System Tick register
19_{16}	STICK_CMPR	RW	Y[3]	Y	System TICK Compare register
$1A_{16}$–$1F_{16}$	*Reserved*	—		—	Any access causes an *illegal_instruction* trap

Notes:

1. Nonprivileged software may read this register if the npt bit is 0. An attempt to read this register by nonprivileged software with npt = 1 causes a *privileged_action* trap. An attempted write by privileged software causes an *illegal_instruction* trap. An attempted write by nonprivileged software causes a *privileged_opcode* trap.
2. An attempted write to this register causes an *illegal_instruction* trap.
3. An attempted access in nonprivilged mode causes a *privileged_opcode* trap.
4. An attempted access in nonprivilged mode with PCR.priv = 1 causes a *privileged_action* trap.
5. Read accesses cause an *illegal_instruction* trap. An attempted write access in nonprivilged mode causes a *privileged_opcode* trap.
6. Nonprivileged software may read this register if the npt bit is 0. An attempt to read this register by nonprivileged software with npt = 1 causes a *privileged_action* trap. A write by privileged or user software causes an *illegal_instruction* trap.

3.1.1 Tick Register (TICK)

The TICK register contains two fields: npt and counter. The npt field is replicated per strand, while the counter field is shared by the eight strands on a physical core. Hyperprivileged software on any strand can write the TICK register. A write of the TICK register will update both the shared counter as well as the writing strand's npt field (the npt fields for other strands will be unaffected). The counter increments each processor core clock that `ASI_CMT_TICK_ENABLE.tick_enable` is set to 1. See *ASI_CMT_TICK_ENABLE* on page 181 for more details. On a warm reset, npt is set to

1, but counter continues counting (if ASI_CMT_TICK_ENABLE.tick_enable is 1) or remains unchanged (if ASI_CMT_TICK_ENABLE.tick_enable is 0). On all other resets, TICK does not change (other than the normal counting of counter).

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.2 Program Counter (PC)

Each strand has a read-only program counter register. The PC contains a 48-bit virtual address and VA{63:48} is sign-extended from VA{47}. The format of this register is shown in TABLE 3-2.

TABLE 3-2 Program Counter – PC (ASR 05_{16})

Bit	Field	Initial (POR) Value	R/W	Description
63:48	va_high	$FFFF_{16}$[1]	RO	Sign-extended from VA{47}.
47:2	va	$3FFFFC000008_{16}$[1]	RO	Virtual address contained in the program counter.
1:0	—	0	RO	The lower 2 bits of the program counter always read as 0.

1. Initial value listed is when ASI_RST_VEC_MASK.VEC_MASK = 0. If ASI_RST_VEC_MASK.VEC_MASK = 1, the initial value for the register is 20_{16}. See Section 29.1.4, *ASI_RST_VEC_MASK*, on page 454 for more details.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.3 Floating-Point State Register (FSR)

Each virtual processor has a Floating-Point State register. This register follows the UltraSPARC Architecture 2007 specification, with the ver field permanently set to 0 and the qne field permanently set to 0 (OpenSPARC T2 does not support a FQ).

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.4 General Status Register (GSR)

Each virtual processor has a nonprivileged general status register (GSR). When PSTATE.pef or FPRS.fef is zero, accesses to this register cause an *fp_disabled* trap.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.5 Software Interrupt Register (SOFTINT)

Each virtual processor has a privileged software interrupt register. Nonprivileged accesses to this register cause a *privileged_opcode* trap. The TICK_CMPR register contains three fields: sm, int_level, and tm. Note that while setting the sm (bit 16), tm (bit 0), and SOFTINT{14} bits all generate *interrupt_level_14*, these bits are considered completely independent of each other. Thus a STICK compare will only set bit 16 and generate *interrupt_level_14*, not also set bit 14.

TABLE 3-3 specifies how *interrupt_level_14* will be shared between SOFTINT writes, STICK compares, and TICK compares.

TABLE 3-3 Sharing of *interrupt_level_14*

Event	tm	SOFTINT{14}	sm	Action
STICK compare when sm = 0	Unchanged	Unchanged	1	*interrupt_level_14* if PSTATE.ie = 1 and PIL < 14
Set sm = 1 when sm = 0	Unchanged	Unchanged	1	*interrupt_level_14* if PSTATE.ie = 1 and PIL < 4
Set SOFTINT{14} = 1 when SOFTINT{14} = 0.	Unchanged	1	Unchanged	*interrupt_level_14* if PSTATE.ie = 1 and PIL < 4
TICK compare when tm = 0	1	Unchanged	Unchanged	*interrupt_level_14* if PSTATE.ie = 1 and PIL < 4
Set tm=1 when tm = 0	1	Unchanged	Unchanged	*interrupt_level_14* if PSTATE.ie = 1 and PIL < 4

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.6 Tick Compare Register (TICK_CMPR)

Each virtual processor has a privileged Tick compare register. Nonprivileged accesses to this register cause a *privileged_opcode* trap. The TICK_CMPR register contains two fields: int_dis and tick_cmpr. A full 63-bit tick_cmpr field is implemented in the register, but the bottom seven bits are ignored when comparing against the TICK counter field. The int_dis bit controls whether a TICK *interrupt_level_14* interrupt is posted in the SOFTINT register when tick_cmpr bits 62:7 match TICK bits 62:7.

Caution To reliably create *interrupt_level_14* interrupts using the tick compare register, software should ensure that the value written to bits 62:7 of the Tick Compare Register is larger than the value subsequently read from bits 62:7 of the TICK Register.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.7 System Tick Register (STICK)

STICK and TICK are derived from the same register. Writes to STICK affect TICK and vice versa.

Writes by user-level code to TICK generate a *privileged_opcode* trap, while writes by user-level code to STICK generate an *illegal_instruction* trap.

Reads of STICK.counter{6:0} are tied to $7F_{16}$. This prevents software from setting the System Tick Compare Register or Hyperprivileged System Tick Compare Register to a value that should cause a subsequent interrupt but that would not be detected due to the System Tick Compare Register and Hyperprivileged System Tick Compare implementation. The compare registers are not continuously compared to STICK, but are compared periodically (at least once every 128 cycles).

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.1.8 System Tick Compare Register (STICK_CMPR)

Each virtual processor has a privileged System Tick Compare (STICK_CMPR register. Nonprivileged accesses to this register cause a *privileged_opcode* trap. STICK_CMPR contains two fields: int_dis and stick_cmpr. A full 63-bit stick_cmpr field is implemented in the register, but the bottom seven bits are ignored when comparing against the STICK counter field. To assist software in reliably creating interrupt_level_14 interrupts using the system tick compare register, OpenSPARC T2 always returns reads of the system tick register with bits 6:0 set to 7'h7F. This ensures that if software writes a value to the system tick compare register that is greater than the value subsequently read from the system tick register that a match will occur in the future.

The int_dis bit controls whether a STICK *interrupt_level_14* interrupt is posted in the SOFTINT register when stick_cmpr bits 62:7 match STICK bits 62:7.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.2 Privileged PR State Registers

TABLE 3-4 lists the privileged registers.

TABLE 3-4 Privileged Registers

Register	Register Name	Access	Replicated per-Strand	Description
00_{16}	TPC	RW	Y	Trap PC[1]
01_{16}	TNPC	RW	Y	Trap Next PC[1]
02_{16}	TSTATE	RW	Y	Trap State
03_{16}	TT	RW	Y	Trap Type
04_{16}	TICK	RW	Partially	Tick
05_{16}	TBA	RW	Y	Trap Base Address[1]
06_{16}	PSTATE	RW	Y	Process State
07_{16}	TL	RW	Y	Trap Level
08_{16}	PIL	RW	Y	Processor Interrupt Level
09_{16}	CWP	RW	Y	Current Window Pointer
$0A_{16}$	CANSAVE	RW	Y	Savable Windows
$0B_{16}$	CANRESTORE	RW	Y	Restorable Windows
$0C_{16}$	CLEANWIN	RW	Y	Clean Windows
$0D_{16}$	OTHERWIN	RW	Y	Other Windows
$0E_{16}$	WSTATE	RW	Y	Window State
10_{16}	GL	RW	Y	Global Level

1. OpenSPARC T2 only implements bits 47:0 of the TPC, TNPC, and TBA registers. Bits 63:48 are always sign-extended from bit 47.

3.2.1 Trap State Register (TSTATE)

Each virtual processor has *MAXTL* (6) Trap State registers. These registers hold the state values from the previous trap level. The format of one element the TSTATE register array (corresponding to one trap level) is shown in TABLE 3-5.

TABLE 3-5 Trap State Register

Bit	Field	Initial (POR) Value	R/W	Description
63:42	—	0	RO	*Reserved*.
41:40	gl	0	RW	Global level at previous trap level
39:32	ccr	0	RW	CCR at previous trap level
31:24	asi	0	RW	ASI at previous trap level
23:21	—	0	RO	*Reserved*
20	pstate tct	0	RW	PSTATE.tct at previous trap level
19:18	—	0	RO	*Reserved* (corresponds to bits 11:10 of PSTATE)
17	pstate cle	0	RW	PSTATE.cle at previous trap level
16	pstate tle	0	RW	PSTATE.tle at previous trap level
15:13	—	0	RO	*Reserved* (corresponds to bits 7:5 of PSTATE)
12	pstate pef	0	RW	PSTATE.pef at previous trap level
11	pstate am	0	RW	PSTATE.am at previous trap level
10	pstate priv	0	RW	PSTATE.priv at previous trap level
9	pstate ie	0	RW	PSTATE.ie at previous trap level
8	—	0	RO	*Reserved* (corresponds to bit 0 of PSTATE)
7:3	—	0	RO	*Reserved*
2:0	cwp	0	RW	CWP from previous trap level

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.2.2 Processor State Register (PSTATE)

Each virtual processor has a Processor State register. More details on PSTATE can be found in the UltraSPARC Architecture 2007 specification. The format of this register is shown in TABLE 3-6; note that the memory model selection field (mm) mentioned in UltraSPARC Architecture 2007 is not implemented in OpenSPARC T2.

TABLE 3-6 Processor State Register

Bit	Field	Initial (POR) Value	R/W	Description
63:13	—	0	RO	*Reserved*
12	tct	0	RW	Trap on control transfer
11:10	—	0	RO	*Reserved*
9	cle	0	RW	Current little endian
8	tle	0	RW	Trap little endian

TABLE 3-6 Processor State Register

Bit	Field	Initial (POR) Value	R/W	Description
7:6	—	0	RO	*Reserved* (mm; not implemented in OpenSPARC T2)
5	—	0	RO	*Reserved* (was red)
4	pef	1	RW	Enable floating-point
3	am	0	RW	Address mask
2	priv	1	RW	Privileged mode
1	ie	0	RW	Interrupt enable
0	—	0	RO	*Reserved* (was ag)

Implementation Note Traps to hyperprivileged space will set PSTATE.priv to 0. PSTATE.priv could be set to either a 0 or 1 for this case, as HPSTATE.hpriv being a 1 overrides the setting in PSTATE.priv.

Programming Note Hyperprivileged changes to translation in delay slots of delayed control transfer instructions should be avoided; see Section 12.3.2, *Real-to-Physical Address Mapping and Speculative Instruction Fetch*, on page 109.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.2.3 Trap Level Register (TL)

Each virtual processor has a Trap Level register. Writes to this register saturate at *MAXPTL* (2) when in privileged mode and at *MAXTL* (6) in hyperprivileged mode. This saturation is based on bits 2:0 of the write data; bits 63:3 of the write data are ignored.

Note Hyperprivileged software can set TL to greater than *MAXPTL* for user or supervisor code by writing to TSTATE followed by a DONE/RETRY, doing a JMPL/WRHPR pair, etc. Operation of the OpenSPARC T2 chip when HPSTATE.hpriv = 0 and TL > *MAXPTL* follows UltraSPARC Architecture, and while in this state all traps destined for privileged level will instead be delivered to hyperprivileged level using the guest watchdog vector.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.2.4 Current Window Pointer (CWP) Register

Since *N_REG_WINDOWS* = 8 on OpenSPARC T2, the CWP register in each virtual processor is implemented as a 3-bit register.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.2.5 Global Level Register (GL)

Each virtual processor has a Global Level register, which controls which set of four global register windows is in use. The maximum global level (*MAXGL*) for OpenSPARC T2 is 3, so GL is implemented as a 2-bit register on OpenSPARC T2. GL is restricted to be less than or equal to *MAXPTL* (2) for privileged and nonprivileged code. On a trap, GL is set to min(GL + 1, *MAXGL*) for traps to hyperprivileged mode and to **min**(GL + 1,*MAXPTL*) for traps to privileged mode. On a DONE or RETRY, if executed with HTSTATE[TL].HPSTATE.hpriv = 1 (so that the DONE or RETRY places the virtual processor in hyperprivileged mode), the value of GL is restored from TSTATE[TL].gl.

Writes to the GL register saturate at *MAXPTL* when in privileged mode, and *MAXGL* in hyperprivileged mode. This saturation is based on bits 3:0 of the write data; bits 63:4 of the write data are ignored.

The format of the GL register is shown in TABLE 3-7.

TABLE 3-7 Global Level Register

Bit	Field	Initial (POR) Value	R/W	Description
63:2	—	0	RO	*Reserved*
1:0	gl	3	RW	Global level.

Note | Hyperprivileged software can still set GL to greater than *MAXPTL* for nonprivileged or privileged code (although this is ***not*** recommended, except in diagnostic code) by doing a JMPL/WRHPR pair when GL > *MAXPTL*. The OpenSPARC T2 chip allows software normal access to the global registers when HPSTATE.hpriv = 0 and GL > *MAXPTL*.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.3 Hyperprivileged Registers

TABLE 3-8 shows the format of the OpenSPARC T2 hyperprivileged registers.

TABLE 3-8 Hyperprivileged Registers

Register	Register Name	Access	Replicated by Strand	Description
00_{16}	HPSTATE	RW	Y	Hypervisor Processor State register
01_{16}	HTSTATE	RW	Y	Hypervisor Trap State register
03_{16}	HINTP	RW	Y	Hypervisor Interrupt Pending register
05_{16}	HTBA	RW	Y	Hypervisor Trap Base Address register[1]
06_{16}	HVER	RO	N	Version register
$1E_{16}$	HALT	RW	Y	Halt instruction
$1F_{16}$	HSTICK_CMPR	RW	Y	Hypervisor System Tick Compare register

1. OpenSPARC T2 only implements bits 47:14 of the tba field. Bits 63:48 are always sign-extended from bit 47.

3.3.1 Hypervisor Processor State Register (HPSTATE)

Each virtual processor has a Hypervisor Processor State register, HPSTATE.

Full documentation on the Hypervisor Processor State register can be found in the UltraSPARC Architecture 2007 specification.

Note The tlz bit retains its current value when a trap is taken, which is different from the UltraSPARC Architecture specification, which specifies it is cleared when any trap is taken.

Programming Note Hyperprivileged changes to translation in delay slots of delayed control transfer instructions should be avoided; see Section 12.3.2, *Real-to-Physical Address Mapping and Speculative Instruction Fetch*, on page 109.

3.3.2 Hypervisor Trap State Register (HTSTATE)

Each virtual processor has a set of Hypervisor Trap State registers, one per trap level. These registers hold the hyperprivileged state values from the previous trap level. Full documentation on this register can be found in the UltraSPARC Architecture specification.

3.3.3 Hypervisor Interrupt Pending Register (HINTP)

Each virtual processor has a Hypervisor Interrupt Pending register. Full documentation on this register can be found in the UltraSPARC Architecture specification.

3.3.4 Hypervisor Trap Base Address Register (HTBA)

Each virtual processor has a Hypervisor Trap Base Address Register. Full documentation on this register can be found in the UltraSPARC Architecture specification.

Note | OpenSPARC T2 only implements bits 47:14 of the tba field of HTBA. Bits 63:48 are always sign-extended from bit 47.

3.3.5 Hyperprivileged Version Register (HVER)

The strands on a physical core share a read-only Version register. Writes to this register generate an *illegal_instruction* trap.

3.3.6 Hyperprivileged System Tick Compare Register (HSTICK_CMPR)

Each virtual processor has a Hyperprivileged System Tick Compare register. HSTICK_CMPR register contains two fields: int_dis and hstick_cmpr.

In the OpenSPARC T2 implementation, a full 63-bit hstick_cmpr field is implemented in the register but the bottom seven bits are ignored when comparing to the STICK counter field. To assist software in reliably creating *hstick_match* traps using the hyperprivileged system tick compare register, OpenSPARC T2 always returns reads of the system tick register with bits 6:0 set to $7F_{16}$ (all ones). This ensures that if software writes a value to HSTICK_CMPR that is greater than the value subsequently read from the system tick register, a match will occur in the future.

For more information on this register, see the UltraSPARC Architecture 2007 specification.

3.3.7 Halt

OpenSPARC T2 provides an iimplementation-specific "halt" pseudo-instruction that can place the virtual processor (strand) executing it into the `halt` state. The "halt" pseudo-instruction is encoded as a write (via WRHPR) to HPR $1E_{16}$ (the "halt" psuedo-register) The virtual strand enters the `halt` state when:
(a) hyperprivileged software writes to HPR $1E_{16}$ and
(b) there are no pending interrupts or modes (as described below) that would prevent entering the `halt` state.

A RDHPR of HPR $1E_{16}$ returns zero.

The format of the "halt" pseudo-register is shown in Table 3-9 .

TABLE 3-9 "Halt" Pseudo-Register

Bit	Field	Initial (POR) Value	R/W	Description
63:0	—	0	RW	*Reserved.Reads return zero and write data is ignored.*

The operation of the Halt pseudo-instruction is as follows. The virtual processor can be parked, disabled, running, or halted. A virtual processor can be enabled or disabled by writing to ASI_CORE_ENABLE (see 14.1.3 on page 180). A virtual processor, when enabled, can either be parked or unparked, by writing to the ASI_CORE_RUNNING_RW register (see 14.1.7 on page 182). When enabled and unparked, the virtual processor is normally running. A running virtual core may be halted by writing to the Halt register. Once halted, the virtual processor remains halted until an interrupt arrives. When the interrupt arrives, the processor transitions back to running. It resumes execution at the NPC of the Halt instruction.

When halted, the virtual processor consumes no execution resource. It is similar to the parked state except that it awakens upon an interrupt.

If an interrupt arrives coincident with the execution of the Halt instruction, the virtual processor remains running and takes the interrupt.

The following interrupts transition a halted virtual processor back to the running state:

1. The virtual processor receives an interrupt from another virtual processor via the Interrupt Vector Dispatch Register(see 7.3.3 on page 55).
2. The virtual processor receives an XIR.
3. The virtual processor receives any disrupting or deferred error leading to a *software_recoverable_error* trap, *hardware_corrected_error* trap, or a deferred *store_error* trap. Note: ASI_SETER masking is not applied, so even if the virtual processors ASI_SETER masks the error trap, the virtual processor transitions to the running state.

4. The virtual processor receives a *modular_arithmetic_interrupt*.

5. The virtual processor receives a *control_word_queue_interrupt.*

6. The virtual processor sets softint[16] (softint.sm). Only stick_match can do this while in halted state. If softint[16] is 1 when the Halt instruction executes, the virtual processor remains in running state. Masking via PIL is not applied, so even if PIL masks the exception, the virtual processor transitions to running state.

7. The virtual processor sets softint[0] (softint.tm). Only tick_match can do this while in halted state. If softint[0] is 1 when the Halt instruction executes, the virtual processor remains in running state. Masking via PIL is not applied, so even if PIL masks the exception, the strand transitions to running state

8. The virtual processor receives an *hstick_match_interrupt*. If hintp is 1 when the Halt instruction executes, the strand remains in running state.

9. The virtual processor receives an *interrupt_vector* exception.

10. The virtual processor receives a park request, as a result of another virtual processor writing to the ASI_CORE_RUNNING_RW or ASI_CORE_RUNNING_W1C registers. In this case, the virtual processor transitions from halted to running to parked.

11. Entry to Single Step mode

12. Entry to Disable Overlap mode

Note: If the virtual processor is executing in Single Step or Disable Overlap mode and executes a Halt instruction, the virtual processor remains running in that mode. It does not enter halt state.

CHAPTER 4

Instruction Format

Instruction formats are described in the UltraSPARC Architecture 2006 specification.

CHAPTER 5

Instruction Definitions

5.1 Instruction Set Summary

The OpenSPARC T2 CPU implements the UltraSPARC Architecture 2007*UltraSPARC Architecture 2007* instruction set.

TABLE 5-1 lists the complete OpenSPARC T2 instruction set supported in hardware. All instructions are documented in the *UltraSPARC Architecture 2007* specification.

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(1 of 6)*

Opcode	Description
ADD (ADDcc)	Add (and modify condition codes)
ADDC (ADDCcc)	Add with carry (and modify condition codes)
ALIGNADDRESS	Calculate address for misaligned data access
ALIGNADDRESSL	Calculate address for misaligned data access (little-endian)
ALLCLEAN	Mark all windows as clean
AND (ANDcc)	And (and modify condition codes)
ANDN (ANDNcc)	And not (and modify condition codes)
ARRAY{8,16,32}	3-D address to blocked byte address conversion
Bicc	Branch on integer condition codes
BMASK	Writes the GSR.mask field
BPcc	Branch on integer condition codes with prediction
BPr	Branch on contents of integer register with prediction
BSHUFFLE	Permutes bytes as specified by the GSR.mask field
CALL[1]	Call and link
CASA	Compare and swap word in alternate space
CASXA	Compare and swap doubleword in alternate space
DONE	Return from trap
EDGE{8,16,32}{L}{N}	Edge boundary processing {little-endian} {non-condition-code altering}

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(2 of 6)*

Opcode	Description
FABS(s,d)	Floating-point absolute value
FADD(s,d)	Floating-point add
FALIGNDATA	Perform data alignment for misaligned data
FANDNOT1{s}	Negated *src1* **and** *src2* (single precision)
FANDNOT2{s}	*Src1* **and** negated *src2* (single precision)
FAND{s}	Logical **and** (single precision)
FBPfcc	Branch on floating-point condition codes with prediction
FBfcc	Branch on floating-point condition codes
FCMP(s,d)	Floating-point compare
FCMPE(s,d)	Floating-point compare (exception if unordered)
FCMPEQ{16,32}	Four 16-bit / two 32-bit compare: set integer dest if *src1* = *src2*
FCMPGT{16,32}	Four 16-bit / two 32-bit compare: set integer dest if *src1* > *src2*
FCMPLE{16,32}	Four 16-bit / two 32-bit compare: set integer dest if *src1* ≤ *src2*
FCMPNE{16,32}	Four 16-bit / two 32-bit compare: set integer dest if *src1* ≠ *src2*
FDIV(s,d)	Floating-point divide
FEXPAND	Four 8-bit to 16-bit expand
FiTO(s,d)	Convert integer to floating-point
FLUSH	Flush instruction memory
FLUSHW	Flush register windows
FMOV(s,d)	Floating-point move
FMOV(s,d)cc	Move floating-point register if condition is satisfied
FMOV(s,d)R	Move floating-point register if integer register contents satisfy condition
FMUL(s,d)	Floating-point multiply
FMUL8SUX16	Signed upper 8- x 16-bit partitioned product of corresponding components
FMUL8ULX16	Unsigned lower 8- x 16-bit partitioned product of corresponding components
FMUL8X16	8- x 16-bit partitioned product of corresponding components
FMUL8X16AL	Signed lower 8- x 16-bit lower α partitioned product of four components
FMUL8X16AU	Signed upper 8- x 16-bit lower α partitioned product of four components
FMULD8SUX16	Signed upper 8- x 16-bit multiply → 32-bit partitioned product of components
FMULD8ULX16	Unsigned lower 8- x 16-bit multiply → 32-bit partitioned product of components
FNAND{s}	Logical **nand** (single precision)
FNEG(s,d)	Floating-point negate
FNOR{s}	Logical **nor** (single precision)
FNOT1{s}	Negate (1's complement) *src1* (single precision)
FNOT2{s}	Negate (1's complement) *src2* (single precision)
FONE{s}	One fill (single precision)
FORNOT1{s}	Negated *src1* **or** *src2* (single precision)

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(3 of 6)*

Opcode	Description
FORNOT2{s}	*src1* **or** negated *src2* (single precision)
FOR{s}	Logical **or** (single precision)
FPACKFIX	Two 32-bit to 16-bit fixed pack
FPACK{16,32}	Four 16-bit/two 32-bit pixel pack
FPADD{16,32}{s}	Four 16-bit/two 32-bit partitioned add (single precision)
fPMERGE	Two 32-bit to 64-bit fixed merge
FPSUB{16,32}{s}	Four 16-bit/two 32-bit partitioned subtract (single precision)
FsMULd	Floating-point multiply single to double
FSQRT(s,d)	Floating-point square root
FSRC1{s}	Copy *src1* (single precision)
FSRC2{s}	Copy *src2* (single precision)
F(s,d)TO(s,d)	Convert between floating-point formats
F(s,d)TOi	Convert floating point to integer
F(s,d)TOx	Convert floating point to 64-bit integer
FSUB(s,d)	Floating-point subtract
FXNOR{s}	Logical **xnor** (single precision)
FXOR{s}	Logical **xor** (single precision)
FxTO(s,d)	Convert 64-bit integer to floating-point
FZERO{s}	Zero fill (single precision)
ILLTRAP	Illegal instruction
INVALW	Mark all windows as CANSAVE
JMPL	Jump and link
LDBLOCKF	64-byte block load
LDDF	Load double floating-point
LDDFA	Load double floating-point from alternate space
LDF	Load floating-point
LDFA	Load floating-point from alternate space
LDFSR	Load floating-point state register lower
LDSB	Load signed byte
LDSBA	Load signed byte from alternate space
LDSH	Load signed halfword
LDSHA	Load signed halfword from alternate space
LDSTUB	Load-store unsigned byte
LDSTUBA	Load-store unsigned byte in alternate space
LDSW	Load signed word
LDSWA	Load signed word from alternate space
LDTW	Load twin words

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(4 of 6)*

Opcode	Description
LDTWA	Load twin words from alternate space
LDUB	Load unsigned byte
LDUBA	Load unsigned byte from alternate space
LDUH	Load unsigned halfword
LDUHA	Load unsigned halfword from alternate space
LDUW	Load unsigned word
LDUWA	Load unsigned word from alternate space
LDX	Load extended
LDXA	Load extended from alternate space
LDXFSR	Load extended floating-point state register
MEMBAR	Memory barrier
MOVcc	Move integer register if condition is satisfied
MOVr	Move integer register on contents of integer register
MULScc	Multiply step (and modify condition codes)
MULX	Multiply 64-bit integers
NOP	No operation
NORMALW	Mark other windows as restorable
OR (ORcc)	Inclusive-or (and modify condition codes)
ORN (ORNcc)	Inclusive-or not (and modify condition codes)
OTHERW	Mark restorable windows as other
PDIST	Distance between 8 8-bit components
POPC	Population count
PREFETCH	Prefetch data
PREFETCHA	Prefetch data from alternate space
PST	Eight 8-bit/4 16-bit/2 32-bit partial stores
RDASI	Read ASI register
RDASR	Read ancillary state register
RDCCR	Read condition codes register
RDFPRS	Read floating-point registers state register
RDHPR	Read hyperprivileged register
RDPC	Read program counter
RDPR	Read privileged register
RDTICK	Read TICK register
RDY	Read Y register
RESTORE	Restore caller's window
RESTORED	Window has been restored
RETRY	Return from trap and retry

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(5 of 6)*

Opcode	Description
RETURN	Return
SAVE	Save caller's window
SAVED	Window has been saved
SDIV (SDIVcc)	32-bit signed integer divide (and modify condition codes)
SDIVX	64-bit signed integer divide
SETHI	Set high 22 bits of low word of integer register
SIAM	Set interval arithmetic mode
SIR	Software-initiated reset
SLL	Shift left logical
SLLX	Shift left logical, extended
SMUL (SMULcc)	Signed integer multiply (and modify condition codes)
SRA	Shift right arithmetic
SRAX	Shift right arithmetic, extended
SRL	Shift right logical
SRLX	Shift right logical, extended
STB	Store byte
STBA	Store byte into alternate space
STBAR	Store barrier
STBLOCKF	64-byte block store
STDF	Store double floating-point
STDFA	Store double floating-point into alternate space
STF	Store floating-point
STFA	Store floating-point into alternate space
STFSR	Store floating-point state register
STH	Store halfword
STHA	Store halfword into alternate space
STTW	Store twin words
STTWA	Store twin words into alternate space
STW	Store word
STWA	Store word into alternate space
STX	Store extended
STXA	Store extended into alternate space
STXFSR	Store extended floating-point state register
SUB (SUBcc)	Subtract (and modify condition codes)
SUBC (SUBCcc)	Subtract with carry (and modify condition codes)
SWAP	Swap integer register with memory
SWAPA	Swap integer register with memory in alternate space

TABLE 5-1 Complete OpenSPARC T2 Hardware-Supported Instruction Set *(6 of 6)*

Opcode	Description
TADDcc (TADDccTV)	Tagged add and modify condition codes (trap on overflow)
TSUBcc (TSUBccTV)	Tagged subtract and modify condition codes (trap on overflow)
Tcc	Trap on integer condition codes (with 8-bit sw_trap_number, if bit 7 is set trap to hyperprivileged)
UDIV (UDIVcc)	Unsigned integer divide (and modify condition codes)
UDIVX	64-bit unsigned integer divide
UMUL (UMULcc)	Unsigned integer multiply (and modify condition codes)
WRASI	Write ASI register
WRASR	Write ancillary state register
WRCCR	Write condition codes register
WRFPRS	Write floating-point registers state register
WRHPR	Write hyperprivileged register
WRPR	Write privileged register
WRY	Write Y register
XNOR (XNORcc)	Exclusive-nor (and modify condition codes)
XOR (XORcc)	Exclusive-or (and modify condition codes)

1. The PC format saved by the CALL instruction is the same as the format of the PC register specified in Section 3.1.2, *Program Counter (pc)*, on page 11.

TABLE 5-2 lists the SPARC V9 and sun4v instructions that are not directly implemented in hardware by OpenSPARC T2, and the exception that occurs when an attempt is made to execute it.

TABLE 5-2 UltraSPARC Architecture 2007 Instructions Not Directly Implemented by OpenSPARC T2 Hardware *(1 of 2)*

Opcode	Description	Exception
FABSq	Floating-point absolute value quad	*illegal_instruction*
FADDq	Floating-point add quad	*illegal_instruction*
FCMPq	Floating-point compare quad	*illegal_instruction*
FCMPEq	Floating-point compare quad (exception if unordered)	*illegal_instruction*
FDIVq	Floating-point divide quad	*illegal_instruction*
FdMULq	Floating-point multiply double to quad	*illegal_instruction*
FiTOq	Convert integer to quad floating-point	*illegal_instruction*
FMOVq	Floating-point move quad	*illegal_instruction*
FMOVqcc	Move quad floating-point register if condition is satisfied	*illegal_instruction*

TABLE 5-2 UltraSPARC Architecture 2007 Instructions Not Directly Implemented by OpenSPARC T2 Hardware *(2 of 2)*

Opcode	Description	Exception
FMOVqr	Move quad floating-point register if integer register contents satisfy condition	*illegal_instruction*
FMULq	Floating-point multiply quad	*illegal_instruction*
FNEGq	Floating-point negate quad	*illegal_instruction*
FSQRTq	Floating-point square root quad	*illegal_instruction*
F(s,d,q)TO(q)	Convert between floating-point formats to quad	*illegal_instruction*
FQTOI	Convert quad floating point to integer	*illegal_instruction*
FQTOX	Convert quad floating point to 64-bit integer	*illegal_instruction*
FSUBq	Floating-point subtract quad	*illegal_instruction*
FxTOq	Convert 64-bit integer to floating-point	*illegal_instruction*
IMPDEP1 (not listed in TABLE 5-1)	Implementation-dependent instruction	*illegal_instruction*
IMPDEP2 (not listed in TABLE 5-1)	Implementation-dependent instruction	*illegal_instruction*
LDQF	Load quad floating-point	*illegal_instruction*
LDQFA	Load quad floating-point into alternate space	*illegal_instruction*
STQF	Store quad floating-point	*illegal_instruction*
STQFA	Store quad floating-point into alternate space	*illegal_instruction*

5.2 OpenSPARC T2-Specific Instructions

5.3 Block Load and Store Instructions

See the LDBLOCKF and STBLOCKF instruction descriptions in the *UltraSPARC Architecture 2007* specification for the standard definitions of these instructions.

Block loads are not allowed to IO space (which is indicated on OpenSPARC T2 by PA{39} = 1).

A block load to IO space generates a *DAE_nc_page* trap.

Block stores to IO space are permitted.

Block store commits in UltraSPARC T2 do NOT force the data to be written to memory as specified in the *UltraSPARC Architecture 2007* specification. Block store commits are implemented the same as block stores in UltraSPARC T2. As with all stores, block stores and block store commits will maintain coherency with all I-caches, but will not flush any modified instructions executing down a pipeline. Flushing those instructions requires the pipeline to execute a FLUSH instruction.

Notes

If LDBLOCKF is used with an `ASI_BLK_COMMIT_{P,S}` and a destination register number rd is specified which is not a multiple of 8 (a misaligned `rd`), OpenSPARC T2 generates an *illegal_instruction* exception (impl. dep. #255-U3-Cs10).

If LDBLOCKF is used with an `ASI_BLK_COMMIT_{P,S}` and a memory address is specified with less than 64-byte alignment, OpenSPARC T2 generates a *mem_address_not_aligned* exception (impl. dep. #256-U3)

These instructions are used for transferring large blocks of data (more than 256 bytes); for example, `bcopy()` and `bfill()`. On OpenSPARC T2, a block load forces a miss in the primary cache and will not allocate a line in the primary cache, but does allocate in L2.

OpenSPARC T2 treats block loads as interlocked with respect to following instructions. That is, all floating-point registers are updated before any subsequent instruction issues.

STBLOCKF source data registers are interlocked against completion of previous instructions, including block load instructions.

LDBLOCKF does not follow memory model ordering with respect to stores. In particular, read-after-write hazards to overlapping addresses are not detected. The side-effect bit associated with the access is ignored (see *Translation Table Entry (TTE)* on page 99). If ordering with respect to earlier stores is important (for example, a block load that overlaps previous stores), then there must be an intervening MEMBAR #`StoreLoad` or stronger MEMBAR. If the LDBLOCKF overlaps a previous store and there is no intervening MEMBAR or data reference, the LDBLOCKF may return data from before or after the store.

Compatibility Note

Prior UltraSPARC implementations may have provided the first two registers at the same time. If code depends upon this unsupported behavior it must be modified for OpenSPARC T2.

STBLOCKF does not follow memory model ordering with respect to loads, previous block stores, or subsequent stores. (OpenSPARC T2 orders block stores with respect to previous nonblock stores). In particular, read-after-write hazards to overlapping addresses are not detected. The side-effects bit associated with the access is ignored.

If ordering with respect to later loads is important then there must be an intervening MEMBAR instruction. If the STBLOCKF overlaps a later load and there is no intervening MEMBAR `#StoreLoad` instruction, the contents of the block are undefined.

Compatibility Notes

Block load and store operations do not obey the ordering restrictions of the currently selected processor memory model (TSO, PSO, or RMO); block operations always execute under an RMO memory ordering model. In general, explicit MEMBAR instructions are required to order block operations among themselves or with respect to normal loads and stores. In addition, block operations do not generally conform to dependence order on the issuing virtual processor; that is, no read-after-write or write-after-read checking occurs between block loads and stores. Explicit MEMBARs are required to enforce dependence ordering between block operations that reference the same address. However, OpenSPARC T2 partially orders some block operations.

TABLE 5-3 describes the synchronization primitives required in OpenSPARC T2, if any, to guarantee TSO ordering between various sequences of memory reference operations. The first column contains the reference type of the first or earlier instruction; the second column contains the reference type of the second or the later instruction. OpenSPARC T2 orders loads and block loads against all subsequent instructions.

TABLE 5-3 OpenSPARC T2 Synchronization Requirements for Memory Reference Operations

First reference	Second reference	Synchronization Required
Load	Load	—
	Block load	—
	Store	—
	Block store	—
Block load	Load	—
	Block load	—
	Store	—
	Block store	—

TABLE 5-3 OpenSPARC T2 Synchronization Requirements for Memory Reference Operations

First reference	Second reference	Synchronization Required
Store	Load	—
	Block load	MEMBAR #StoreLoad or #Sync
	Store	—
	Block store	—
Block store	Load	MEMBAR #StoreLoad or #Sync
	Block load	MEMBAR #StoreLoad or #Sync
	Store	MEMBAR #Sync
	Block store	MEMBAR #Sync

Block Initializing Store ASIs

Instruction	imm_asi	ASI Value	Operation
ST[B,H,W,TW,X]A	ASI_ST_BLKINIT_AS_IF_USER_PRIMARY (ASI_STBI_AIUP)	22_{16}	64-byte block initialing store to primary address space, user privilege
	ASI_ST_BLKINIT_AS_IF_USER_SECONDARY (ASI_STBI_AIUS)	23_{16}	64-byte block initialing store to secondary address space, user privilege
	ASI_ST_BLKINIT_NUCLEUS (ASI_STBI_N)	27_{16}	64-byte block initialing store to nucleus address space
	ASI_ST_BLKINIT_AS_IF_USER_PRIMARY_LITTLE (ASI_STBI_AIUPL)	$2A_{16}$	64-byte block initialing store to primary address space, user privilege, little-endian
	ASI_ST_BLKINIT_AS_IF_USER_SECONDARY_LITTLE (ASI_STBI_AIUS_L)	$2B_{16}$	64-byte block initialing store to secondary address space, user privilege, little-endian
	ASI_ST_BLKINIT_NUCLEUS_LITTLE (ASI_STBI_NL)	$2F_{16}$	64-byte block initialing store to nucleus address space, little-endian
	ASI_ST_BLKINIT_PRIMARY (ASI_STBI_P)	$E2_{16}$	64-byte block initialing store to primary address space

Instruction	imm_asi	ASI Value	Operation
	ASI_ST_BLKINIT_SECONDARY (ASI_STBI_S)	$E3_{16}$	64-byte block initialing store to secondary address space
	ASI_ST_BLKINIT_PRIMARY_LITTLE (ASI_STBI_PL)	EA_{16}	64-byte block initialing store to primary address space, little-endian
	ASI_ST_BLKINIT_SECONDARY_LITTLE (ASI_STBI_SL)	EB_{16}	64-byte block initialing store to secondary address space, little-endian

Description

Block initializing store instructions are selected by using one of the block initializing store ASIs with integer store instructions. These ASIs allow block initializing stores to be performed to the same address spaces as normal stores. Little-endian ASIs access data in little-endian format, otherwise the access is assumed to be big-endian.

Integer stores of all sizes (to alternate space) are allowed to use these ASIs

All stores to these ASIs operate under relaxed memory ordering (RMO), regardless of the PSTATE.mm setting, and software must follow a sequence of these stores with a MEMBAR #Sync to ensure ordering with respect to subsequent loads and stores. Stores to these ASIs where the least-significant 6 bits of the address are non-zero (that is, not the first word in the cache line) behave the same as a normal RMO store. A store to these ASIs where the least-significant 6 bits are zero will load a cache line in the L2 cache with either all zeros or the existing data, and then update that line with the new store data. This special store will make sure the line maintains coherency when it is loaded into the cache, but will not generally fetch the line from memory (initializing it with zeros instead). Stores using these ASIs to a noncacheable address (PA{39} = 1) will behave the same as a normal store.

Note These instructions are used for transferring large blocks of data (more than 256 bytes); for example, `bcopy()` and `bfill()`. On OpenSPARC T2, a quad load forces a miss in the primary cache and will not allocate a line in the primary cache, but does allocate in L2.

Access to these ASIs by a floating-point store (STFA, STDFA) will result in a *DAE_invalid_ASI* trap (or *mem_address_not_aligned* trap if not properly aligned for the store size).

The following pseudocode shows how these ASIs can be used to do a quadword aligned (on both source and destination) copy of N quadwords from *A* to *B* (where N > 3). Note that the final 64 bytes of the copy is performed using normal stores, guaranteeing that all initial zeros in a cache line are overwritten with copy data.

```
%l0 ← [A]
%l1 ← [B]
prefetch [%l0]
for (i = 0; i < N-4; i++) {
```

```
    if (!(i % 4)) {
       prefetch [%l0+64]
    }
    ldtxa [%l0] #ASI_TWINX_P, %l2
    add %l0, 16, %l0
    stxa %l2, [%l1] #ASI_ST_BLKINIT_PRIMARY
    add %l1, 8, %l1
    stxa %l3, [%l1] #ASI_ST_BLKINIT_PRIMARY
    add %l1, 8, %l1
}
for (i = 0; i < 4; i++) {
    ldtxa [%l0] #ASI_TWINX_P, %l2
    add %l0, 16, %l0
    stx %l2, [%l1]
    stx %l3,d [%l1+8]
    add %l1, 16, %l1
}
membar #Sync
```

Programming Notes

These ASIs are specific to OpenSPARC T2 to provide a high-performance bcopy alternative to block load and store (which fetch the lined stored to from memory to the L2 cache, requiring three memory operations for `bcopy()` and two memory operations for a `bfill()`). These ASIs are of Class "N" and are only allowed in dynamically linked, platform-specific, OS-enabled libraries.

These ASIs provide a higher-performance `bcopy()` or `bfill()` than LDBLOCKF and STBLOCKF, due to their ability to avoid the unnecessary fetch from memory of the data that is overwritten by the store.

5.3.1 Load Twin Extended Word

Load Twin Extended Word instructions are not allowed to access IO space (indicated by PA{39} = 1 on OpenSPARC T2). An LDTXA to IO space generates a *DAE_nc_page* trap.

CHAPTER 6

Traps

6.1 Trap Levels

Each virtual processor supports six trap levels (*MAXTL* = 6). Traps to privileged mode (supervisor software) while in privileged mode when TL = *MAXPTL* (2) will trap instead to the hyperprivileged mode (hypervisor software), using the guest watchdog vector in the hyperprivileged trap table. TL will be incremented, but the processor will not enter `RED_state` and the trap type will be set to that of the trap that caused the event, not the watchdog trap type.

6.2 Trap Behavior

TABLE 6-1 specifies the codes used in the tables below.

TABLE 6-1 Table Codes

Code	Meaning
H	Trap is taken via the Hyperprivileged trap table, in Hyperprivileged mode (HSTATE.hpriv = 1)
P	Trap is taken via the Privileged trap table, in Privileged mode (PSTATE.priv = 1)

TABLE 6-1 Table Codes

Code	Meaning
H^U	Trap is taken via the Hyperprivileged trap table, in Hyperprivileged mode (HSTATE.hpriv = 1). However, the trap is *unexpected*. While hardware can legitimately generate this trap, it should not do so unless there is a programming error or some other error. Therefore, occurrence of this trap indicates an actual error to hyperprivileged software.
-x-	Not possible. Hardware cannot generate this trap in the indicated running mode. For example, all privileged instructions can be executed in both privileged and hyperprivileged modes, therefore a *privileged_opcode* trap cannot occur in privileged or hyperprivileged mode.
—	This trap can only legitimately be generated by hyperprivileged software, not by the CPU hardware. So, for the purposes of sun4v, the trap vector has to be correct, but for a hardware CPU implementation these trap types are not generated by the hardware, therefore the resultant running mode is irrelevant.

For example, trap 1 ("power on reset") in TABLE 6-2, if delivered in any running mode, results in a delivery directly to the hypervisor mode.

TABLE 6-2 Trap Behavior *(1 of 3)*

			From privilege level:		
TT #	Hardware Trap Name	Priority	Nonprivileged	Privileged	Hyperprivileged
0_{16}	*Reserved*	—	—	—	—
1_{16}	*power_on_reset*	0	H	H	H
2_{16}	*watchdog_reset*	variable[2]	H	H	H
	guest watchdog	variable[3]	H	H	—
3_{16}	*externally_initiated_reset*	1.1	H	H	H
4_{16}	*software_initiated_reset*	1.3	-x-	-x-	H
5_{16}	*RED_state_exception*	1.4	H	H	H
6_{16}	*Reserved*	—	—	—	—
7_{16}	*store_error*	2.1	H	H	H
8_{16}	*IAE_privilege_violation*	3.1	H	-x-	-x-
9_{16}	*instruction_access_MMU_miss*	2.8	H	H	-x-[9]
A_{16}	*instruction_access_error*	4	H	H	H
B_{16}	*IAE_unauth_access*	2.9[4]	H	H	H^U
C_{16}	*IAE_nfo_page*	3.3	H	H	H^U
D_{16}	*instruction_address_range*	2.6	H	H	H^U
E_{16}	*instruction_real_range*	2.6	H	H	H^U
F_{16}	*Reserved*	—	—	—	—
10_{16}	*illegal_instruction*	6.1[5]	H	H	H
11_{16}	*privileged_opcode*	7	P	-x-	-x-
12_{16}	*unimplemented_LDTW*	—	—	—	—
13_{16}	*unimplemented_STTW*	—	—	—	—

TABLE 6-2 Trap Behavior *(2 of 3)*

TT #	Hardware Trap Name	Priority	From privilege level: Nonprivileged	Privileged	Hyperprivileged
14_{16}	*DAE_invalid_asi*	12.1	H	H	H^U
15_{16}	*DAE_privilege_violation*	12.4	H	H	H^U
16_{16}	*DAE_nc_page*	12.5	H	H	H^U
17_{16}	*DAE_nfo_page*	12.6	H	H	H^U
18_{16}–$1F_{16}$	*Reserved*	—	—	—	—
20_{16}	*fp_disabled*	8.1	P	P	H^U
21_{16}	*fp_exception_ieee_754*	11.1	P	P	H^U
22_{16}	*fp_exception_other*	11.1	P	P	H^U
23_{16}	*tag_overflow*	14	P	P	H^U
24_{16}–27_{16}	*clean_window*	10.1	P	P	H^U
28_{16}	*division_by_zero*	15	P	P	H^U
29_{16}	*internal_processor_error*	8.2 or 12.10[6]	H	H	H
$2A_{16}$	*instruction_invalid_TSB_entry*	2.10[7]	H	H	-x-
$2B_{16}$	*data_invalid_TSB_entry*	12.3	H	H	H
$2C_{16}$	*Reserved*	—	—	—	—
$2D_{16}$	*mem_real_range*	11.3	H	H	H^U
$2E_{16}$	*mem_address_range*	11.3	H	H	H^U
$2F_{16}$	*Reserved*	—	—	—	—
30_{16}	*DAE_so_page*	12.6	H	H	H^U
31_{16}	*data_access_MMU_miss*	12.3	H	H	H
32_{16}	*data_access_error*	12.9	H	H	H
33_{16}	*data_access_protection*	—	—	—	—
34_{16}	*mem_address_not_aligned*	10.2	H	H	H^U
35_{16}	*LDDF_mem_address_not_aligned*	10.1	H	H	H^U
36_{16}	*STDF_mem_address_not_aligned*	10.1	H	H	H^U
37_{16}	*privileged_action*	11.1	H	H	-x-
38_{16}	*LDQF_mem_address_not_aligned*	—	—	—	—
39_{16}	*STQF_mem_addess_not_aligned*	—	—	—	—
$3A_{16}$	*Reserved*	—	—	—	—
$3B_{16}$	*unsupported_page_size*	13	H	H	H^U
$3C_{16}$	*control_word_queue_interrupt*	16.5	H	H	H
$3D_{16}$	*modular_arithmetic_interrupt*	16.4	H	H	H
$3E_{16}$	*inst_real_translation_miss*	2.8	H	H	N[9]
$3F_{16}$	*data_real_translation_miss*	12.3	H	H	H
40	*sw_recoverable_error*	33.1	H	H	H
41_{16}–$4F_{16}$	*interrupt_level_n*	$32 - n$	P	P	-x-

TABLE 6-2 Trap Behavior *(3 of 3)*

TT #	Hardware Trap Name	Priority	From privilege level: Nonprivileged	Privileged	Hyperprivileged
$4F_{16}$	*pic_overflow* (*interrupt_level_15*)	16.0, variable[8]	P	P	-x-
50_{16}–$5D_{16}$	*Reserved*	—	—	—	—
$5E_{16}$	*hstick_match*	16.1	H	H	H
$5F_{16}$	*trap_level_zero*	2.2	H	H	-x-
60_{16}	*interrupt_vector_trap*	16.3	H	H	H
61_{16}	*PA_watchpoint*	12.8	H	H	H
62_{16}	*VA_watchpoint*	11.2	H	H	-x-
63_{16}	*hw_corrected_error*	33.2	H	H	H
64_{16}[1]	*fast_instruction_access_MMU_miss*	2.8	H	H	-x-
68_{16}[1]	*fast_data_access_MMU_miss*	12.3	H	H	H
$6C_{16}$[1]	*fast_data_access_protection*	12.7	H	H	H
70_{16}	*Reserved*	—	—	—	—
71_{16}	*instruction_access_MMU_error*	2.7	H	H	-x-
72_{16}	*data_access_MMU_error*	12.2	H	H	H
73_{16}	*Reserved*	—	—	—	—
74_{16}	*control_transfer_instruction*	11.1	P	P	H
75_{16}	*instruction_VA_watchpoint*	2.5	H	H	-x-
76_{16}	*instruction_breakpoint*	6.2[5]	H	H	H
77_{16}–$7B_{16}$	*Reserved*	—	—	—	—
$7C_{16}$	*cpu_mondo_trap*	16.6	P	P	-x-
$7D_{16}$	*dev_mondo_trap*	16.7	P	P	-x-
$7E_{16}$	*resumable_error*	33.3	P	P	-x-
$7F_{16}$	*nonresumable_error* (generated by software only)	—	—	—	—
80_{16}–$9C_{16}$[1]	*spill_n_normal* (n = 0–7)	9	P	P	H^U
A0–BC_{16}[1]	*spill_n_other* (n = 0–7)	9	P	P	H^U
$C0_{16}$–DC_{16}[1]	*fill_n_normal* (n = 0–7)	9	P	P	H^U
$E0_{16}$–FC_{16}[1]	*fill_n_other* (n = 0–7)	9	P	P	H^U
100_{16}–$17F_{16}$	*trap_instruction*	16.2	P	P	H
180_{16}–$1FF_{16}$	*htrap_instruction*	16.2	-x-	H	H^U

1. Trap extends across four TT #s to allow trap handler to contain 32 inline instructions instead of the standard 8 inline instructions.
2. The *watchdog_reset* priority is inherited from the underlying exception.
3. The *guest_watchdog* priority is inherited from the underlying exception.
4. OpenSPARC T2 deviates from the 3.2 priority in UltraSPARC Architecture 2007 for *IAE_unauth_access*.

5. OpenSPARC T2 deviates from UltraSPARC Architecture 2007 and swaps the priority of *illegal_instruction* (6.2 in UltraSPARC Architecture 2007) and *instruction_breakpoint* (6.1 in UltraSPARC Architecture 2007)
6. IRF/FRF errors that are encountered on the second or subsequent passes of a multicycle operation (partial store, compare and swap, block store) generate an *internal_processor_error* trap at priority 12.10. All other *internal_processor_error* traps are priority 8.2. See *Error Trap Vectors* on page 217.
7. OpenSPARC T2 deviates from the UltraSPARC Architecture 2007 priority of 2.8 for *instruction_invalid_TSB_entry*.
8. To make the *pic_overflow* trap the highest-priority disrupting trap, *pic_overflow* has an elevated priority over a normal *interrupt_level_15* trap (such as would be generated by writing 1 to SOFTINT{15}). Per Section 10.2, *SPARC Performance Instrumentation Counter*, on page 84, if the *pic_overflow* trap is taken on the instruction that caused the overflow, then the effective priority of the *pic_overflow* inherits from the condition that caused the overflow.
9. Hyperprivileged instruction access always bypasses translation. See *Translation* on page 122.

6.3 Trap Masking

TABLE 6-3 specifies the codes used inTABLE 6-3.

TABLE 6-3 Codes

Code	Meaning
(nm)	Never Masked — when the condition occurs in this running mode, it is never masked out and the trap is always taken.
(ie)	When the outstanding disrupting trap condition occurs in this privilege mode, it may be conditioned (masked out) by PSTATE.ie = 0 (but remains pending).
PIL	Masked by PSTATE.ie and PIL
tct	Masked by PSTATE.tct
ibe	Masked by HPSTATE.ibe
tlz	Masked by HPSTATE.tlz
M	Always masked
—	This trap can only legitimately be generated by hyperprivileged software, not by the CPU hardware. So, for the purposes of sun4v, the trap vector has to be correct, but for a hardware CPU implementation these trap types are not generated by the hardware, therefore the resultant running mode is irrelevant.

For example, trap $7C_{16}$ (“cpu mondo”) in TABLE 6-4 is masked by PSTATE.ie in nonprivileged and privileged mode and is always masked in hyperprivileged mode.

TABLE 6-4 lists the trap mask behavior.

TABLE 6-4 Trap Mask Behavior *(1 of 3)*

			From privilege level:		
TT #	**Hardware Trap Name**	**Type**	**Nonprivileged**	**Privileged**	**Hyperprivileged**
0_{16}	*Reserved*	Reset	(nm)	(nm)	(nm)
1_{16}	*power_on_reset*	Reset	(nm)	(nm)	(nm)
2_{16}	*watchdog_reset*	Reset	(nm)	(nm)	(nm)
	guest watchdog	Reset	(nm)	(nm)	—
3_{16}	*externally_initiated_reset*	Reset	(nm)	(nm)	(nm)
4_{16}	*software_initiated_reset*	Reset	(nm)	(nm)	(nm)
5_{16}	*RED_state_exception*	Reset	(nm)	(nm)	(nm)
6_{16}	*Reserved*	—	—	—	—
7_{16}	*store_error*	Deferred	(nm)	(nm)	(nm)
8_{16}	*IAE_privilege_violation*	Precise	(nm)	(nm)	(nm)
9_{16}	*instruction_access_MMU_miss*	Precise	(nm)	(nm)	—
A_{16}	*instruction_access_error*	Precise	(nm)	(nm)	(nm)
B_{16}	*IAE_unauth_access*	Precise	(nm)	(nm)	(nm)
C_{16}	*IAE_nfo_page*	Precise	(nm)	(nm)	(nm)
D_{16}	*instruction_address_range*	Precise	(nm)	(nm)	(nm)
E_{16}	*instruction_real_range*	Precise	(nm)	(nm)	(nm)
F_{16}	*Reserved*	—	—	—	—
10_{16}	*illegal_instruction*	Precise	(nm)	(nm)	(nm)
11_{16}	*privileged_opcode*	Precise	(nm)	—	—
12_{16}	*unimplemented_LDTW*	—	—	—	—
13_{16}	*unimplemented_STTW*	—	—	—	—
14_{16}	*DAE_invalid_asi*	Precise	(nm)	(nm)	(nm)
15_{16}	*DAE_privilege_violation*	Precise	(nm)	(nm)	(nm)
16_{16}	*DAE_nc_page*	Precise	(nm)	(nm)	(nm)
17_{16}	*DAE_nfo_page*	Precise	(nm)	(nm)	(nm)
18_{16}–$1F_{16}$	*Reserved*	—	—	—	—
20_{16}	*fp_disabled*	Precise	(nm)	(nm)	(nm)
21_{16}	*fp_exception_ieee_754*	Precise	(nm)	(nm)	(nm)
22_{16}	*fp_exception_other*	Precise	(nm)	(nm)	(nm)
23_{16}	*tag_overflow*	Precise	(nm)	(nm)	(nm)
24_{16}–27_{16}	*clean_window*	Precise	(nm)	(nm)	(nm)
28_{16}	*division_by_zero*	Precise	(nm)	(nm)	(nm)
29_{16}	*internal_processor_error*	Precise	(nm)	(nm)	(nm)

TABLE 6-4 Trap Mask Behavior *(2 of 3)*

TT #	Hardware Trap Name	Type	From privilege level:		
			Nonprivileged	Privileged	Hyperprivileged
$2A_{16}$	*instruction_invalid_TSB_entry*	Precise	(nm)	(nm)	—
$2B_{16}$	*data_invalid_TSB_entry*	Precise	(nm)	(nm)	(nm)
$2C_{16}$	*Reserved*	—	—	—	—
$2D_{16}$	*mem_real_range*	Precise	(nm)	(nm)	(nm)
$2E_{16}$	*mem_address_range*	Precise	(nm)	(nm)	(nm)
$2F_{16}$	*Reserved*	—	—	—	—
30_{16}	*DAE_so_page*	Precise	(nm)	(nm)	(nm)
31_{16}	*data_access_MMU_miss*	Precise	(nm)	(nm)	(nm)
32_{16}	*data_access_error*	Precise	(nm)	(nm)	(nm)
33_{16}	*data_access_protection*	—	—	—	—
34_{16}	*mem_address_not_aligned*	Precise	(nm)	(nm)	(nm)
35_{16}	*LDDF_mem_address_not_aligned*	Precise	(nm)	(nm)	(nm)
36_{16}	*STDF_mem_address_not_aligned*	Precise	(nm)	(nm)	(nm)
37_{16}	*privileged_action*	Precise	(nm)	—	—
38_{16}	*LDQF_mem_address_not_aligned*	—	—	—	—
39_{16}	*STQF_mem_addess_not_aligned*	—	—	—	—
$3A_{16}$	*Reserved*	—	—	—	—
$3B_{16}$	*unsupported_page_size*	Precise	(nm)	(nm)	(nm)
$3C_{16}$	*control_word_queue_interrupt*	Disrupting	(nm)	(nm)	(ie)
$3D_{16}$	*modular_arithmetic_interrupt*	Disrupting	(nm)	(nm)	(ie)
$3E_{16}$	*inst_real_translation_miss*	Precise	(nm)	(nm)	—
$3F_{16}$	*data_real_translation_miss*	Precise	(nm)	(nm)	(nm)
40	*sw_recoverable_error*	Disrupting	(nm)	(nm)	(ie)
41_{16}–$4F_{16}$	*interrupt_level_n*	Disrupting	PIL	PIL	M
$4F_{16}$	*pic_overflow* (*interrupt_level_15*)	Disrupting	PIL	PIL	M
50_{16}–$5D_{16}$	*Reserved*	—	—	—	—
$5E_{16}$	*hstick_match*	Disrupting	(nm)	(nm)	(ie)
$5F_{16}$	*trap_level_zero*	Disrupting	tlz	tlz	—
60_{16}	*interrupt_vector_trap*	Disrupting	(nm)	(nm)	(ie)
61_{16}	*PA_watchpoint*	Precise	(nm)	(nm)	(nm)
62_{16}	*VA_watchpoint*	Precise	(nm)	(nm)	—
63_{16}	*hw_corrected_error*	Disrupting	(nm)	(nm)	(ie)
64_{16}[1]	*fast_instruction_access_MMU_miss*	Precise	(nm)	(nm)	—
68_{16}[1]	*fast_data_access_MMU_miss*	Precise	(nm)	(nm)	(nm)
$6C_{16}$[1]	*fast_data_access_protection*	Precise	(nm)	(nm)	(nm)
70_{16}	*Reserved*	—	—	—	—

TABLE 6-4 Trap Mask Behavior *(3 of 3)*

TT #	Hardware Trap Name	Type	From privilege level: Nonprivileged	Privileged	Hyperprivileged
71_{16}	*instruction_access_MMU_error*	Precise	(nm)	(nm)	—
72_{16}	*data_access_MMU_error*	Precise	(nm)	(nm)	(nm)
73_{16}	*Reserved*	—	—	—	—
74_{16}	*control_transfer_instruction*	Precise	tct	tct	tct
75_{16}	*instruction_VA_watchpoint*	Precise	(nm)	(nm)	—
76_{16}	*instruction_breakpoint*	Precise	ibe	ibe	ibe
77_{16}–$7B_{16}$	*Reserved*	—	—	—	—
$7C_{16}$	*cpu_mondo_trap*	Disrupting	(ie)	(ie)	M
$7D_{16}$	*dev_mondo_trap*	Disrupting	(ie)	(ie)	M
$7E_{16}$	*resumable_error*	Disrupting	(ie)	(ie)	M
$7F_{16}$	*nonresumable_error* (generated by software only)	—	—	—	—
80_{16}–$9C_{16}$[1]	*spill_n_normal* (*n* = 0–7)	Precise	(nm)	(nm)	(nm)
A0–BC_{16}[1]	*spill_n_other* (*n* = 0–7)	Precise	(nm)	(nm)	(nm)
$C0_{16}$–DC_{16}[1]	*fill_n_normal* (*n* =0–7)	Precise	(nm)	(nm)	(nm)
$E0_{16}$–FC_{16}[1]	*fill_n_other* (*n* = 0–7)	Precise	(nm)	(nm)	(nm)
100_{16}–$17F_{16}$	*trap_instruction*	Precise	(nm)	(nm)	(nm)
180_{16}–$1FF_{16}$	*htrap_instruction*	Precise	—	(nm)	(nm)

1. Trap extends across four TT #s to allow trap handler to contain 32 inline instructions instead of the standard 8 inline instructions.

The OpenSPARC T2 implmentation deviates from UltraSPARC Architecture 2007 in following way: when an SIR or XIR reset occurs, the virtual processor enters `RED_state` **regardless** of the value of TL (that is, when TL = *MAXTL*, as well as when TL < *MAXTL*.). In particular, when TL = *MAXTL* and an XIR occurs, UltraSPARC Architecture 2007 specifies that the virtual processor enters `error_state` to generate a WDR reset, but an OpenSPARC T2 virtual processor instead directly takes an XIR reset.

CHAPTER 7

Interrupt Handling

The chapter describes the hardware interrupt delivery mechanism for the OpenSPARC T2 chip. I/O and CPU cross-call interrupts are delivered to hyperprivileged code on each virtual processor using an interrupt vector trap as described in *Interrupt Flow* on page 46. Error interrupts are delivered to hyperprivileged code on each virtual processor using the *sw_recoverable_error* and *hw_corrected_error* traps. These error interrupts are described in Chapter 25, *Error Handling*.

Hyperprivileged code notifies privileged code about *interrupt_vector, sw_recoverable_error,* and *hw_corrected_error* traps (and precise error traps) through the *cpu_mondo*, *dev_mondo*, and *resumable_error* traps as described in *Interrupt Queue Registers* on page 52. Software interrupts are delivered to each virtual processor using the *interrupt_level_n* traps. Software interrupts are described in the UltraSPARC Architecture 2006 Specification. The *pic_overflow* trap, generated by the performance counters, is described in Chapter 10, *Performance Instrumentation*. The *hstick_match* and *trap_level_zero* interrupts are described in the UltraSPARC Architecture 2006 specification.

Interrupt vector traps have a corresponding 64-bit `ASI_INTR_RECEIVE` register. I/O devices and inter-CPU cross-call interrupts contain a 6-bit identifier, which determines which interrupt vector (level) in the `ASI_INTR_RECEIVE` register the interrupt will target. Each virtual processor's `ASI_INTR_RECEIVE` register can queue up to 64 outstanding interrupts, one for each interrupt vector. Interrupt vectors are implicitly prioritized, with vector 63 being the highest priority and vector 0 being the lowest priority.

Two types of I/O interrupts are supported. "Internal" I/O interrupts, such as those generated by the Network Interface Unit, are generated by I/O devices on the OpenSPARC T2 processor. Unlike "mondo" interrupts, these interrupts do not contain any additional data payload. Each internal I/O interrupt source has a hardwired interrupt number, which is used to index a table of interrupt vector information (INT_MAN) in the NCU. Generally, each I/O interrupt source will be assigned a unique virtual processor target and vector level. This association is defined by software programming of the interrupt vector and strand fields in the INT_MAN table in the NCU. Software must maintain the association between interrupt vector and hardware interrupt number to index the appropriate entry in the INT_MAN table.

The second type of interrupts are external “mondo” interrupts, such as those generated by PCI-Express. These interrupts follow the standard mondo interrupt ACK/NACK flow control. Only the first two 64-bit words of mondo data are supported by OpenSPARC T2.

7.1 Interrupt Flow

7.1.1 Sources

CPU cross-call interrupts can be generated by writing the Interrupt Vector Dispatch register described in *Interrupt Vector Dispatch Register* on page 55. Dispatching inter-CPU interrupts is described in *Dispatching* on page 46.

TAP interrupts can be generated by writing the NCU Interrupt Vector/Trap Dispatch Register described in *Mondo Data Tables* on page 51, via the JTAG port.

SSI error interrupts (device ID = 1) are caused by SSI detected errors, as described in *Boot ROM Interface (SSI)* on page 321.

SSI interrupts (device ID = 2) are caused by an assertion (edge trigger) on the EXT_INT_L pin.

The network interface unit generates interrupts with device IDs 64–127.

7.1.2 Dispatching

CPU cross-call interrupts can be generated by writing the Interrupt Vector Dispatch register described in *Interrupt Vector Dispatch Register* on page 55. Unlike mondo interrupts, interrupts are always received by the destination, and stores to this register will follow the TSO memory model (no MEMBAR #Sync is required). The store data supplies the destination virtual processor and vector. The bit corresponding to the specified vector is set in the Interrupt Receive register of the destination virtual processor.

Note An interrupt that is sent to a virtual processor that is not enabled (has its bit clear in ASI_CORE_ENABLE_STATUS described in *ASI_CORE_ENABLE_STATUS* on page 179) will be lost. An interrupt that is sent to a virtual processor that is parked (has its bit clear in ASI_CORE_RUNNING_STATUS described in *ASI_CORE_RUNNING_STATUS* on page 183) will result in the bit in the interrupt receive register being set, and the interrupt will be taken once the virtual processor is unparked and the interrupt is enabled.

7.1.3 States

Each bit in the Interrupt Receive register can be in one of two states: set or cleared. If an incoming interrupt attempts to set an already set bit, the additional incoming interrupt will be lost (there is no overflow indication on the interrupt bits). Writes to the Interrupt Receive register will clear any bit in which the corresponding write data is 0, and reads to the Incoming Vector register will clear the bit of the highest-priority pending interrupt as a side effect. If another interrupt attempts to set a bit on the same cycle as the bit is being cleared by an Interrupt Receive register write or Incoming Vector register read, the additional interrupt will take precedence over the clear and the bit will remain set.

Mondo interrupts have two states in the MONDO_INT_BUSY table, namely, BUSY and IDLE (not BUSY). When a mondo interrupt transaction is received, if the current state is IDLE, the transaction is accepted (and ACKed to the requestor), state is changed to BUSY, the mondo data is stored in the MONDO_INT_DATA0/1 table, and the bit specified by MONDO_INT_VEC is set in the Interrupt Receive register of the virtual processor specified in the INT transaction. While in the BUSY state, any INT transactions to the same virtual processor will be rejected and NACKed back to the requestor, and the requestor is responsible for preserving that interrupt in a pending state. When software has adequately serviced the interrupt, it explicitly clears the busy bit in the MONDO_INT_BUSY register to return to the IDLE state.

For "internal" I/O interrupts, such as an SSI, network interface unit, or MCU error interrupt, the bit specified by INT_MAN is set in the Interrupt Receive register of the virtual processor specified in INT_MAN. Software can use the Incoming Vector register described in *Incoming Vector Register* on page 56 to atomically clear and return the vector of the highest status bit. Since there is no ACK/NACK flow control on the internal interrupts and no indication of interrupt bit overflow, for sources capable of generating multiple interrupts between interrupt servicing, software will need to properly handle the multiple interrupt case.

7.1.4 Prioritizing

Interrupt vector traps are implicitly prioritized by the Incoming Vector register described in Section 7.3.4 from bit 63 (highest) to bit 0 (lowest).

The priority of I/O interrupts is done by specifying the vector value in the INT_MAN table and in MONDO_INT_VEC (see Section 7.2.1).

7.1.5 Initialization

The interrupt vector traps are initialized by writing 0_{16} to the Interrupt Receive register described in *Interrupt Receive Register* on page 54.

I/O interrupt handling is initialized by

a. Specifying the strand/vector pair to receive "internal" I/O interrupts, such as those generated by the network interface unit, by programming the INT_MAN table, described in *SSI Interrupt Management Registers* on page 50.

b. Specifying the interrupt vector to receive external "mondo" interrupts, such as those generated by PCI-Express, by programming the MONDO_INT_VEC register, described in *SSI Interrupt Management Registers* on page 50.

c. Clearing the busy bits in the MONDO_INT_BUSY table, described in *Mondo Interrupt Busy Table* on page 52.

7.1.6 Servicing

Interrupt vector traps are typically serviced by reading the Incoming Vector register described in *Incoming Vector Register* on page 56. When this register is read by software, the 6-bit vector corresponding to the highest priority pending interrupt in the Interrupt Receive register is returned. The pending interrupt bit for that vector is cleared.

If the incoming interrupt matches the virtual processor and vector for SSI error interrupts (device ID = 1), the handler should read the SSI error logs, described in *SSI Error Registers* on page 322, to determine the cause of the interrupt. It should service the interrupt appropriately, checking for multiple interrupts that could have occurred on the same bit in the Interrupt Receive register.

If the incoming interrupt matches the virtual processor and vector for SSI interrupts (device ID = 2), the handler should service the interrupt appropriately, checking for multiple interrupts that could have occurred on the same bit in the Interrupt Receive register.

If the incoming interrupt matches the vector for “mondo” interrupts, the handler should then read the mondo source and data, from the MONDO_INT_ADATA0/1 registers (described in *Mondo Data Tables*) and MONDO_INT_ABUSY registers (described in *Mondo Interrupt Busy Table*). After reading these registers, it should enable receiving the next mondo interrupt to this virtual processor by clearing the busy bit in the MONDO_INT_ABUSY register.

7.2 NCU Interrupt Registers

The following registers are defined for interrupt and reset management. The base address is defined below.

The NCU handles interrupts generated externally through the SSI EXT_INT_L pin.

TABLE 7-1 lists the device ID assignment for interrupts.

TABLE 7-1 Device ID Assignments

Device	ID Range	Comment
Reserved	0	
SSI Errors	1	SSI parity or timeout error.
SSI Interrupt	2	SSI interrupt from EXT_INT_L pin.
Reserved	3–63	

On-chip interrupt hardware contains an SSI Interrupt Management table. Each internal “Device ID” in the I/O subsystem has an entry in the SSI Interrupt Management Table described in *SSI Interrupt Management Registers*.

Device ID 0 is used internally by hardware but is architecturally reserved.

Device ID 1 is used to report SSI Errors. Software will have to poll the SSI error registers to determine the error type.

Device ID 2 is the interrupt from EXT_INT_L pin, of the SSI interface, which is intended for use as a console interrupt.

Device IDs 3-63 are reserved.

7.2.1 SSI Interrupt Management Registers

The SSI Interrupt Management registers specify the CPU ID to send the interrupt and the interrupt vector associated with the interrupt issued by the NCU on behalf of the device.

Each device will send its device ID to the NCU. The device ID is used to index into the SSI Interrupt Management table. Before changing the SSI Interrupt Management register, software must disable all incoming interrupts. Note that the offset address of the corresponding device can be calculated by multiplying the device ID by 8 for INT_MAN.

Software or boot code must program the INT_MAN table before any of the non-mondo type interrupt is generated. Reading of the INT_MAN table without first initializing it by software or boot code will result in false parity errors in NCU.

TABLE 7-2 shows the format of the SSI Interrupt Management register.

TABLE 7-2 SSI/NIU Interrupt Management – INT_MAN Register ($80\ 0000\ 0000_{16}$)(Count 128 step 8)

Bit	Field	Initial Value	R/W	Description
63:14	—	0	RO	*Reserved*
13:8	cpu	X	RW	ID of virtual processor to manage the device.
7:6	—	0	RO	*Reserved*
5:0	vector	0	RW	Interrupt vector (encodes bit set in `ASI_INTR_RECEIVE`).

7.2.2 Mondo Interrupt Vector Register

The Mondo Interrupt Vector register specifies the interrupt vector for PCI-Express mondo interrupts. Since the virtual strand ID is specified in the mondo interrupt, this register is shared among the 64 virtual processors.

TABLE 7-3 shows the format of the Mondo Interrupt Vector register.

TABLE 7-3 Mondo Interrupt Vector Register – MONDO_INT_VEC ($80\ 0000\ 0A00_{16}$)

Bit	Field	Initial Value	R/W	Description
63:6	—	0	RO	*Reserved*
5:0	vector	X	RW	Interrupt vector for mondo interrupts (encodes bit set in `ASI_INTR_RECEIVE`).

MONDO_INT_VEC performs the identical function for PCI-Express Mondo interrupts that INT_MAN performs for the SSI interrupts, except that the virtual strand ID is specified in the mondo interrupt transaction.

7.2.3 Mondo Data Tables

The following registers manage the receipt from mondo interrupts.

The base address of the mondo interrupt registers is defined below.

There are two Mondo Interrupt Data tables. The tables are read-only by software, and the entries are updated by an incoming mondo interrupts provided that the interrupt is not busy. The NCU will ACK the interrupt if it is not busy; otherwise, the NCU will NACK it.

TABLE 7-4 shows the format of the Mondo Interrupt Data 0 table.

TABLE 7-4 Mondo Interrupt Data 1 – MONDO_INT_DATA0 ($80\ 0004\ 0000_{16}$) (Count 64 Step 8)

Bit	Field	Initial Value	R/W	Description
63:0	data0	X	RO	First 64 bits of mondo interrupt data.

TABLE 7-5 shows the format of the Mondo Interrupt Data 1 table.

TABLE 7-5 Mondo Interrupt Data 1 – MONDO_INT_DATA1 ($80\ 0004\ 0200_{16}$) (Count 64 Step 8)

Bit	Field	Initial Value	R/W	Description
63:0	data1	X	RO	Second 64 bits of mondo interrupt data.

TABLE 7-6 shows the format of the Mondo Interrupt Alias Data 0 table.

This register address is actually an alias for MONDO_INT_DATA0[*My CPUID*], so each virtual processor can read its own interrupt payload without having to do an address calculation based on strand_id. This address should never be accessed by the TAP (since it does not have a strand_id).

TABLE 7-6 Mondo Interrupt Alias Data 0 – MONDO_INT_ADATA0 ($80\ 0004\ 0400_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	data0	X	RO	First 64 bits of mondo interrupt data.

TABLE 7-7 shows the format of the Mondo Interrupt Alias Data 1 Table.

This register address is actually an alias for MONDO_INT_DATA1[*My CPUID*], so each virtual processor can read its own interrupt payload, without having to do an address calculation based on strand_id. This address should never be accessed by the TAP (since it does not have a strand_id).

TABLE 7-7 Mondo Interrupt Alias Data 1 – MONDO_INT_ADATA1 ($80\ 0004\ 0600_{16}$)

Bit	Field	Initial Value	R/W	Initial Value
63:0	data1	X	RO	Second 64 bits of mondo interrupt data.

7.2.4 Mondo Interrupt Busy Table

When the NCU receives a mondo interrupt, it sets the busy bit to 1 and ACKs the interrupt. When the busy bit is set, it implies an interrupt is waiting to be serviced or is being serviced. Software will reset the busy bit when it completes servicing the interrupt. If the busy bit is already set when an interrupt is received, a NACK will be sent to the interrupt source. The busy bit is set after a reset and software has to clear it to begin receiving interrupts.

TABLE 7-8 shows the format of the Mondo Interrupt Busy table.

TABLE 7-8 Mondo Interrupt Busy – MONDO_INT_BUSY $(80\ 0004\ 0800)_{16}$ (Count 64 Step 8)

Bit	Field	Initial Value	R/W	Description
63:7	—	0	RO	*Reserved*
6	busy	1	RW	Hardware sets busy to 1 when an interrupt is received. Hardware NACKs an incoming interrupt if busy is set.
5:0	—	0	RO	*Reserved*

TABLE 7-9 shows the format of the Mondo Interrupt Alias Busy table.

This register address is actually an alias for MONDO_INT_BUSY[*My CPUID*], so each virtual processor can update its own mondo interrupt busy bit without having to do an address calculation based on strand_id. This address should never be accessed by the TAP (since it does not have a strand_id).

TABLE 7-9 Mondo Interrupt Alias Busy – MONDO_INT_ABUSY $(80\ 0004\ 0A00_{16})$

Bit	Field	Initial Value	R/W	Description
63:7	—	0	RO	*Reserved*
6	busy	1	RW	Hardware sets busy to 1 when an interrupt is received. Hardware NACKs an incoming interrupt if busy is set.
5:0	—	0	RO	*Reserved*

7.3 CPU Interrupt Registers

7.3.1 Interrupt Queue Registers

Each virtual processor has eight ASI_QUEUE registers at ASI = 25_{16}, VA{63:0} = $3C0_{16}$-$3F8_{16}$ that are used for communicating interrupts to the operating system. These registers contain the head and tail pointers for four supervisor interrupt queues: *cpu_mondo, dev_mondo, resumable_error, nonresumable_error.*

The tail registers are read-only by supervisor, and read/write by hypervisor. Writes to the tail registers by the supervisor generate a *DAE_invalid_ASI* trap. The head registers are read/write by both supervisor and hypervisor.

Whenever the CPU_MONDO_HEAD register does not equal the CPU_MONDO_TAIL register, a *cpu_mondo* trap is generated. Whenever the DEV_MONDO_HEAD register does not equal the DEV_MONDO_TAIL register, a *dev_mondo* trap is generated. Whenever the RESUMABLE_ERROR_HEAD register does not equal the RESUMABLE_ERROR_TAIL register, a *resumable_error* trap is generated. Unlike the other queue register pairs, the *nonresumable_error* trap is *not* automatically generated whenever the NONRESUMABLE_ERROR_HEAD register does not equal the NONRESUMABLE_ERROR_TAIL register; instead, the hypervisor will need to generate the *nonresumable_error* trap.

TABLE 7-10 through TABLE 7-17 define the format of the eight ASI_QUEUE registers.

TABLE 7-10 CPU Mondo Head Pointer – ASI_QUEUE_CPU_MONDO_HEAD (ASI 25_{16}, VA $3C0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	head	X	RW	Head pointer for CPU mondo interrupt queue.
5:0	—	0	RO	*Reserved*

TABLE 7-11 CPU Mondo Tail Pointer – ASI_QUEUE_CPU_MONDO_TAIL (ASI 25_{16}, VA $3C8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	tail	X	RW (hyperpriv) RO (priv)	Tail pointer for CPU mondo interrupt queue.
5:0	—	0	RO	*Reserved*

TABLE 7-12 Device Mondo Head Pointer – ASI_QUEUE_DEV_MONDO_HEAD (ASI 25_{16}, VA $3D0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	head	X	RW	Head pointer for device mondo interrupt queue.
5:0	—	0	RO	*Reserved*

TABLE 7-13 Device Mondo Tail Pointer – ASI_QUEUE_DEV_MONDO_TAIL (ASI 25_{16}, VA $3D8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	tail	X	RW (hyperpriv) RO (priv)	Tail pointer for device mondo interrupt queue.
5:0	—	0	RO	*Reserved*

TABLE 7-14 Resumable Error Head Pointer – `ASI_QUEUE_RESUMABLE_HEAD` (ASI 25_{16}, VA $3E0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	Reserved.
17:6	head	X	RW	Head pointer for resumable error queue.
5:0	—	0	RO	*Reserved*

TABLE 7-15 Resumable Error Tail Pointer – `ASI_QUEUE_RESUMABLE_TAIL` (ASI 25_{16}, VA $3E8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	tail	X	RW (hyperpriv) RO (priv)	Tail pointer for resumable error queue.
5:0	—	0	RO	*Reserved*

TABLE 7-16 Nonresumable Error Head Pointer – `ASI_QUEUE_NONRESUMABLE_HEAD` (ASI 25_{16}, VA $3F0_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	head	X	RW	Head pointer for nonresumable error queue.
5:0	—	0	RO	*Reserved*

TABLE 7-17 Nonresumable Error Tail Pointer – `ASI_QUEUE_NONRESUMABLE_TAIL` (ASI 25_{16}, VA $3F8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:18	—	0	RO	*Reserved*
17:6	tail	X	RW (hyperpriv) RO (priv)	Tail pointer for nonresumable error queue.
5:0	—	0	RO	*Reserved*

7.3.2 Interrupt Receive Register

Each virtual processor has a hyperprivileged `ASI_INTR_RECEIVE` register at ASI = 72_{16}, VA{63:0} = 0. Each time an interrupt transaction arrives for that virtual processor, the bit corresponding to the interrupt vector will be set. Bit zero of the register corresponds to interrupt vector number zero and so on. Interrupt vectors are implicitly prioritized with vector number 63 being the highest priority and vector number 0 being the lowest priority. Software writes to this register are **and**ed with the register contents to allow the software to selectively clear register bits, although normally the incoming vector register described in Section 7.3.4 will be used to clear the bit corresponding to the pending interrupt. When an interrupt arrives at the

same time as a register write, the interrupt will take precedence over the write and the bit will be set. Software can read this register to determine all pending interrupts, although normally the incoming vector register will be used to get the highest priority pending interrupt. Nonprivileged or supervisor access to this register causes a *privileged_action* trap.

TABLE 7-18 defines the format of the ASI_INTR_RECEIVE register.

TABLE 7-18 Interrupt Receive Register – ASI_INTR_RECEIVE (ASI 72_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:0	pending	X	RW	Pending interrupts.

7.3.3 Interrupt Vector Dispatch Register

Each virtual processor has a hyperprivileged write-only ASI_INTR_W register at ASI = 73_{16}, VA{63:0} = 0 that is used to send CPU cross-call interrupts to other virtual processors. Unlike mondo interrupts, interrupts cannot be NACKed by the destination, and multiple interrupts that set the same Interrupt Receive register bit before it has been cleared will only generate a single interrupt. Interrupts generated by stores to this register will follow the TSO memory model (no MEMBAR #Sync is required). The store data supplies the destination virtual processor and vector. The bit corresponding to the specified vector is set in the Interrupt Receive register of the destination virtual processor.

Programming Note After an interrupt vector trap is taken by the destination virtual processor, it is the responsibility of the interrupt handler to clear the highest-priority pending bit in the interrupt register, usually by a read to the Incoming Vector register as described in *Incoming Vector Register*.

The format of the register is shown in TABLE 7-19.

TABLE 7-19 Interrupt Vector Dispatch Register – ASI_INTR_W (ASI 73_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:14	—	0	RO	*Reserved*
13:8	strand	X	W	Destination virtual processor
7:6	—	0	RO	*Reserved*.
5:0	vector	X	W	Interrupt Vector (encodes bit set in ASI_INTR_RECEIVE)

Nonprivileged or supervisor access to this register causes a *privileged_action* trap. A read from this ASI causes a *DAE_invalid_asi* trap.

Implementation Note This register is actually implemented in the NCU and is also available at address 90_{16}-$01CC_{16}$-0000_{16} as described in *Interrupt Vector Dispatch Register (ASI 7316 VA 016)* on page 210.

7.3.4 Incoming Vector Register

Each virtual processor has a hyperprivileged read-only `ASI_INTR_R` register at ASI = 74_{16}, VA{63:} = 0_{16}. When this register is read by software, the 6-bit vector corresponding to the highest priority pending interrupt in the Interrupt Receive register is returned. The pending interrupt bit for that vector is cleared. If no interrupt bits are set, a read of this register will return all zeros. When an interrupt arrives at the same time as the register is read, the interrupt will take precedence over the clearing and the bit will remain set. Nonprivileged or supervisor access to this register causes a *privileged_action* trap. A store to this register will result in a *DAE_invalid_ASI* trap.

Programming Note The interrupt handler will normally use the Incoming Vector register to determine the highest-priority interrupt that is pending while atomically clearing the bit corresponding to that highest priority interrupt.

TABLE 7-20 defines the format of the `ASI_INTR_R` register.

TABLE 7-20 Incoming Vector Register – `ASI_INTR_R` (ASI 74_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:6	—	0	RO	*Reserved*
5:0	vector	X	RO	Interrupt vector.

CHAPTER 8

Memory Models

SPARC V9 defines the semantics of memory operations for three memory models. From strongest to weakest, they are Total Store Order (TSO), Partial Store Order (PSO), and Relaxed Memory Order (RMO). The differences in these models lie in the freedom an implementation is allowed in order to obtain higher performance during program execution. The purpose of the memory models is to specify any constraints placed on the ordering of memory operations in uniprocessor and shared-memory multiprocessor environments. OpenSPARC T2 supports only TSO, with the exception that certain ASI accesses (such as block loads and stores) may operate under RMO.

Although a program written for a weaker memory model potentially benefits from higher execution rates, it may require explicit memory synchronization instructions to function correctly if data is shared. MEMBAR is a SPARC V9 memory synchronization primitive that enables a programmer to control explicitly the ordering in a sequence of memory operations. Processor consistency is guaranteed in all memory models.

The current memory model is indicated in the PSTATE.mm field. It is unaffected by normal traps, but is set to TSO (PSTATE.mm = 0) when the virtual processor enters `RED_state`. OpenSPARC T2 ignores the value set in this field and always operates under TSO.

A memory location is identified by an 8-bit address space identifier (ASI) and a 64-bit virtual address. The 8-bit ASI may be obtained from a ASI register or included in a memory access instruction. The ASI is used to distinguish between and provide an attribute for different 64-bit address spaces. For example, the ASI is used by the OpenSPARC T2 MMU and memory access hardware to control virtual-to-physical address translations, access to implementation-dependent control and data registers, and for access protection. Attempts by nonprivileged software (PSTATE.priv = 0) to access restricted ASIs (ASI{7} = 0) cause a *privileged_action* trap.

Memory is logically divided into real memory (cached) and I/O memory (noncached with and without side effects) spaces, based on bit 39 of the physical address (0 for real memory, 1 for I/O memory). Real memory spaces can be accessed without side effects. For example, a read from real memory space returns the information most recently written. In addition, an access to real memory space does

not result in program-visible side effects. In contrast, a read from I/O space may not return the most recently written information and may result in program-visible side effects.

8.1 Supported Memory Models

The following sections contain brief descriptions of the two memory models supported by OpenSPARC T2. These definitions are for general illustration. Detailed definitions of these models can be found in *The SPARC Architecture Manual-Version 9*. The definitions in the following sections apply to system behavior as seen by the programmer.

Notes Stores to OpenSPARC T2 internal ASIs, block loads, and block stores and block initializing stores are outside the memory model; that is, they need MEMBARs to control ordering.

Atomic load-stores are treated as both a load and a store and can only be applied to cacheable address spaces.

8.1.1 TSO

OpenSPARC T2 implements the following programmer-visible properties in Total Store Order (TSO) mode:

- Loads are processed in program order; that is, there is an implicit MEMBAR `#LoadLoad` between them.
- Loads may bypass earlier stores. Any such load that bypasses such earlier stores must check (snoop) the store buffer for the most recent store to that address. A MEMBAR `#Lookaside` is not needed between a store and a subsequent load at the same noncacheable address.
- A MEMBAR `#StoreLoad` must be used to prevent a load from bypassing a prior store if Strong Sequential Order is desired.
- Stores are processed in program order.
- Stores cannot bypass earlier loads.
- Accesses with PA{39} set (that is, to I/O space) are all strongly ordered with respect to each other.

Compatibility Note Prior UltraSPARC implementations strongly order accesses based on the e bit being set. The e bit is ignored by OpenSPARC T2 for the purposes of strong ordering; only PA{39} is used for determining strong ordering.

- An L2 cache update is delayed on a store hit until all outstanding stores reach global visibility. For example, a cacheable store following a noncacheable store is not globally visible until the noncacheable store has reached global visibility; there is an implicit MEMBAR #MemIssue between them.

8.1.2 RMO

OpenSPARC T2 implements the following programmer-visible properties for special ASI accesses that operate under Relaxed Memory Order (RMO) mode:

- There is no implicit order between any two memory references, either cacheable or noncacheable, except that noncacheable accesses with PA{39} set (that is, to I/O space) are all strongly ordered with respect to each other.

Compatibility Note Prior UltraSPARC implementations strongly order accesses based on the e bit being set. The e bit is ignored by OpenSPARC T2 for the purposes of strong ordering, only PA{39} is used for determining strong ordering.

- A MEMBAR must be used between cacheable memory references if stronger order is desired. A MEMBAR #MemIssue is needed for ordering of cacheable after noncacheable accesses. A MEMBAR #Lookaside should be used between a store and a subsequent load at the same noncacheable address.

CHAPTER 9

Address Spaces and ASIs

9.1 Physical Address Spaces

OpenSPARC T2 supports a 48-bit virtual address space and a 40-bit physical address space. The 40-bit physical address space is further broken into two sections, based on bit 39. If bit 39 is a 0, the address maps to a memory location. If bit 39 is a 1, the address maps to an I/O location.

9.1.1 Access to Nonexistent Physical Memory Addresses

Access to nonexistent physical memory addresses is described in *Access to Nonexistent Memory* on page 375.

9.1.2 Access to Nonexistent I/O Addresses

A load access from a nonexistent memory or I/O location will cause a *data_access_error* exception. An instruction fetch from a nonexistent memory or I/O location will cause an *instruction_access_error* exception. A store access to a nonexistent memory or I/O location will be silently discarded by the system.

9.1.3 Instruction Fetching from I/O

Instruction fetching from I/O addresses is only permitted from the SSI (FF 0000 0000_{16}–FF FFFF $FFFC_{16}$) and L2CSR spaces (A0 0000 0000_{16}–BF FFFF $FFFC_{16}$). Instruction fetches from I/O addresses outside the SSI and L2CSR spaces will take an *instruction_access_error* trap.

Warning | Instruction fetching from the L2CSR space can cause undefined behavior. Software needs to prevent instruction fetches from accessing the L2CSR space.

9.1.4 Supported vs. Unsupported Access Sizes to I/O

All I/O locations internal to OpenSPARC T2 are 64-bit locations, and only support 8-byte (64-bit) loads and stores. Accesses in other sizes may cause traps or have other unexpected results. In particular, non-8-byte aligned load accesses to internal OpenSPARC T2 I/O addresses (except internal PCI-Express or SSI locations) will result in *data_access_error* trap. Non-8-byte-aligned store accesses to internal OpenSPARC T2 I/O addresses (except internal PCI-Express or SSI locations) will be silently discarded by the system. Non-8-byte-aligned load accesses from internal PCI-Express or SSI locations are treated internally as 8-byte loads, with potentially undefined results. Non-8-byte-aligned store accesses to internal PCI-Express or SSI locations are treated internally as 8-byte stores, also with potentially undefined results.

OpenSPARC T2 supports 1-byte, 2-byte, 4-byte, and 8-byte loads and stores via the SSI bus (Boot ROM port). 8-byte stores under mask (generated by STDFA to `ASI_PST*`) are undefined. 16-byte loads (generated by an LDDA to `ASI_TWINX*`), block loads, and block stores generate a *DAE_nc_page* exception. (OpenSPARC T2 cannot generate a 16-byte store.)

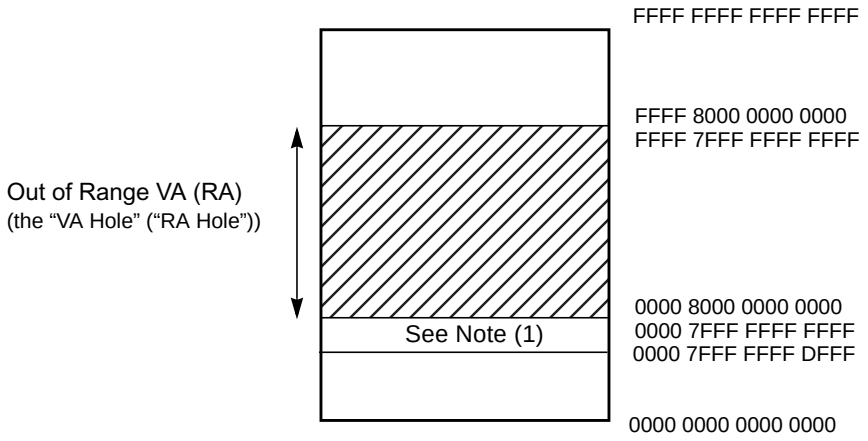
OpenSPARC T2 supports 1-byte, 2-byte, 4-byte, and 8-byte loads and stores, plus 8-byte stores under mask (generated by STDFA to `ASI_PST*`) and block stores, to PCI-Express (for external PCI-Express locations). 16-byte loads (generated by an LDDA to `ASI_TWINX*`) and block loads are not supported to PCI-Express and generate a *DAE_nc_page* exception.

9.1.5 48-bit Virtual and Real Address Spaces

OpenSPARC T2 supports a 48-bit subset of the full 64-bit virtual and real address spaces. Although the full 64 bits are generated and stored in integer registers, legal addresses are restricted to two equal halves at the extreme lower and upper portions of the full virtual (real) address space. Virtual (real) addresses between $0000\ 8000\ 0000\ 0000_{16}$ and $FFFF\ 7FFF\ FFFF\ FFFF_{16}$ inclusive lie within a "VA hole" ("RA hole"), are termed "out-of-range"[1], and are illegal. Prior UltraSPARC implementations introduced the additional restriction on software to not use pages within 4 Gbytes of the VA (RA) hole as instruction pages to avoid problems with prefetching into the VA (RA) hole. OpenSPARC T2 implements a hardware check for instruction fetching near the VA (RA) hole and generates an *instruction_address_range* or *instruction_real_range* trap when instructions are

[1] Another way to view an out-of-range address is as any address where bits {63:48} are not all equal to bit {47}.

executed from a location in the address range 0000 7FFF FFFF $FFE0_{16}$ to 0000 7FFF FFFF $FFFF_{16}$, inclusive. However, even though OpenSPARC T2 provides this hardware checking, it is still recommended that software should *not* use the 8-Kbyte page before the VA (RA) hole for instructions. Address translation and MMU related descriptions can be found in *Translation* on page 122.



Note (1): Use of this region restricted to data only.

FIGURE 9-1 OpenSPARC T2's 48-bit Virtual and Real Address Spaces, With Hole

Throughout this document, when virtual (real) address fields are specified as 64-bit quantities, they are assumed to be sign-extended based on VA{47} (RA{47}).

A number of state registers are affected by the reduced virtual and real address spaces. The PC, I/D-TLB Tag Access, instruction and data watchpoint registers are 48 bits, sign-extended to 64-bits on read accesses. DMMU SFAR, TBA, TPC, and TNPC, registers are 48-bits and their values are *not* sign-extended when read. No checks are done when these registers are written by software. It is the responsibility of privileged (or hyperprivileged) software to properly update these registers.

An out-of-range virtual (real) address during an instruction access, caused by execution into the VA (RA) hole or into 0000 7FFF FFFF $FFE0_{16}$ to 0000 7FFF FFFF $FFFF_{16}$ inclusive, results in an *instruction_address_range* (*instruction_real_range*) trap if PSTATE.am = 0. In addition, OpenSPARC T2 hardware detects when a branch target or the target of a DONE or RETRY instruction is in the VA (RA) hole, and PSTATE.am changes from being set ('1') for the branch, DONE, or RETRY to being cleared ('0') for the target instruction (via the branch delay slot instruction or TSTATE) and generates an *instruction_address_range* (*instruction_real_range*) exception.

Note The *instruction_address_range* and *instruction_real_range* exceptions occur on a fetch that enters the VA hole or that is in the cache line immediately before the VA hole (or when the target of a branch or DONE/RETRY instruction is in the VA hole and VA hole detection is enabled only for the target). Hardware does not store state to create an *instruction_address_range* or *instruction_real_range* exception when the strand is executing from the VA hole in a state that cannot detect VA hole exceptions (for example, when PSTATE.am = 1), and then software transitions the strand to be able to detect the VA hole (for example, by setting PSTATE.am = 0) from within the VA hole itself.

If the target virtual (real) address of a JMPL, RETURN, branch, or CALL instruction is an out-of-range address and PSTATE.am = 0, a *mem_address_range* (*mem_real_range*) trap is generated with TPC equal to the address of the JMPL, RETURN, branch, or CALL instruction. The target address is loaded into the D-MMU SFAR. Because the D-MMU SFAR contains only 48 bits, the trap handler must decode the load or store instruction if the full 64-bit virtual address is needed. See also *I-/D-MMU Synchronous Fault Address Registers (SFAR)* on page 133.

An out-of-range virtual (real) address during a data access results in a *mem_address_range* (*mem_real_range*) trap if PSTATE.am = 0. Because the D-MMU SFAR contains only 48 bits, the trap handler must decode the load or store instruction if the full 64-bit virtual address is needed. See also *I-/D-MMU Synchronous Fault Address Registers (SFAR)* on page 133.

9.1.6 I/O Address Spaces

I/O addresses are distinguished from memory addresses via their high-order physical address bit (bit 39). If bit 39 is 0, the address is a memory address. If bit 39 is 1, the address is an I/O address.

TABLE 9-1 summarizes the OpenSPARC T2 address space, which is broken down into sections based on the eight most significant bits of the physical address.

TABLE 9-1 OpenSPARC T2 Address Space

Address Range (PA{39:32})	Block Name	Comment
00_{16} – $7F_{16}$	DRAM	Main memory
80_{16}	NCU	Noncacheable Unit
81_{16}	—	—
82_{16}	—	*Reserved*
83_{16}	CCU	Clock Unit.

TABLE 9-1 OpenSPARC T2 Address Space *(Continued)*

Address Range (PA{39:32})	Block Name	Comment
84_{16}	MCU	Memory Control Unit, address bits {13:12} select one of the four MCUs.
85_{16}	TCU	JTAG/TAP unit.
86_{16}	DBG	Debug.
87_{16}	—1	*Reserved*
88_{16}	DMU	DMU CSRs.
89_{16}	RST	Reset.
$8A_{16}$–$8F_{16}$	—2	*Reserved*
90_{16}	ASI	CPU shared registers (directly accessible only by JTAG/TAP unit).
91_{16}–$9F_{16}$	—2	*Reserved*
$A0_{16}$–BF_{16}	L2CSR	L2 control and status registers.
$D0_{16}$–FE_{16}	—3	*Reserved*
FF_{16}	SSI	Boot ROM.

Programming Note Address space 90_{16} provides an alias for the shared CPU registers. This alias is directly accessible only by the JTAG/TAP unit. Access to these shared CPU registers by the strands should be done directly through the ASIs listed in *Alternate Address Spaces* on page 65.

9.2 Alternate Address Spaces

TABLE 9-2 summarizes the ASI usage in OpenSPARC T2. The Section/Page column contains a reference to the detailed explanation of the ASI (the page number refers to this chapter). For internal ASIs, the legal VAs are listed (or the field contains "Any" if all VAs are legal). Only bits 47:0 are checked when determining the legal VA range. An access outside the legal VA range will generate a *DAE_invalid_asi* trap.

Notes | All internal, nontranslating ASIs in OpenSPARC T2 can only be accessed using LDXA and STXA.

ASIs 80_{16}–FF_{16} are unrestricted (access allowed in all modes -- nonprivileged, privileged, and hyperprivileged). ASIs 00_{16}–$2F_{16}$ are restricted to privileged and hyperprivileged modes, while ASIs 30_{16}–$7F_{16}$ are restricted to hyperprivileged mode only. Attempted access by nonprivileged or privileged code to a hyperprivileged ASI will result in a *privileged_action* trap.

TABLE 9-2 OpenSPARC T2 ASI Usage *(1 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
00_{16}–03_{16}			Any	—	*DAE_invalid_asi*	
04_{16}	ASI_NUCLEUS	RW	Any	—	Implicit address space, nucleus context, TL > 0	(See UA-2007)
05_{16}–$0B_{16}$			Any	—	*DAE_invalid_asi*	
$0C_{16}$	ASI NUCLEUS_LITTLE	RW	Any	—	Implicit address space, nucleus context, TL > 0 (LE)	(See UA-2007)
$0D_{16}$–$0F_{16}$			Any	—	*DAE_invalid_asi*	
10_{16}	ASI_AS_IF_USER_PRIMARY	RW	Any	—	Primary address space, user privilege	(See UA-2007)
11_{16}	ASI_AS_IF_USER_SECONDARY	RW	Any	—	Secondary address space, user privilege	(See UA-2007)
12_{16}–13_{16}			Any	—		
14_{16}	ASI_REAL	RW	Any	—	Real address (normally used as cacheable)	page 76
15_{16}	ASI_REAL_IO	RW	Any	—	Real address (normally used as noncacheable, with side effect)	page 76
16_{16}	ASI_BLOCK_AS_IF_USER_PRIMARY	RW	Any	—	64-byte block load/store, primary address space, user privilege	5.3
17_{16}	ASI_BLOCK_AS_IF_USER_SECONDARY	RW	Any	—	64-byte block load/store, secondary address space, user privilege	5.3
18_{16}	ASI_AS_IF_USER_PRIMARY_LITTLE	RW	Any	—	Primary address space, user privilege (LE)	(See UA-2007)
19_{16}	ASI_AS_IF_USER_SECONDARY_LITTLE	RW	Any	—	Secondary address space, user privilege (LE)	(See UA-2007)
$1A_{16}$–$1B_{16}$			Any	—		

TABLE 9-2 OpenSPARC T2 ASI Usage *(2 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
$1C_{16}$	ASI_REAL_LITTLE	RW	Any	—	Real address (normally used as cacheable) (LE)	page 76
$1D_{16}$	ASI_REAL_IO_LITTLE	RW	Any	—	Real address (normally used as noncacheable, with side effect) (LE)	page 76
$1E_{16}$	ASI_BLOCK_AS_IF_USER_PRIMARY_LITTLE	RW	Any	—	64-byte block load/store, primary address space, user privilege (LE)	5.3
$1F_{16}$	ASI_BLOCK_AS_IF_USER_SECONDARY_LITTLE	RW	Any	—	64-byte block load/store, secondary address space, user privilege (LE)	5.3
20_{16}	ASI_SCRATCHPAD	RW	0_{16}–18_{16}	Y	Scratchpad registers	page 76
20_{16}	ASI_SCRATCHPAD	RW	20_{16}–28_{16}	—		page 76
20_{16}	ASI_SCRATCHPAD	RW	30_{16}–38_{16}	Y	Scratchpad registers	page 76
21_{16}	ASI_MMU	RW	8_{16}	Y	I/DMMU Primary Context register 0	12.11.2
21_{16}	ASI_MMU	RW	10_{16}	Y	DMMU Secondary Context register 0	12.11.2
21_{16}	ASI_MMU	RW	108_{16}	Y	I/DMMU Primary Context register 1	12.11.2
21_{16}	ASI_MMU	RW	110_{16}	Y	DMMU Secondary Context register 1	12.11.2
22_{16}	ASI_TWINX_AIUP, ASI_STBI_AIUP	RW	Any	—	Load: 128-bit atomic load twin extended word, primary address space, user privilege Store: Block initializing store, primary address space, user privilege	5.7.4
23_{16}	ASI_TWINX_AIUS, ASI_STBI_AIUS	RW	Any	—	Load: 128-bit atomic load twin extended word, secondary address space, user privilege Store: Block initializing store	(See UA-2007)
24_{16}	ASI_TWINX	RO	Any	—	128-bit atomic load twin extended word	(See UA-2007)
25_{16}	ASI_QUEUE	RW	$3C0_{16}$	Y	CPU Mondo Queue head pointer	7.3.1

TABLE 9-2 OpenSPARC T2 ASI Usage *(3 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
25_{16}	ASI_QUEUE	RW (hyperpriv) RO (priv)	3C8	Y	CPU Mondo Queue tail pointer	7.3.1
25_{16}	ASI_QUEUE	RW	$3D0_{16}$	Y	Device Mondo Queue head pointer	7.3.1
25_{16}	ASI_QUEUE	RW (hyperpriv) RO (priv)	$3D8_{16}$	Y	Device Mondo Queue tail pointer	7.3.1
25_{16}	ASI_QUEUE	RW	$3E0_{16}$	Y	Resumable Error Queue head pointer	7.3.1
25_{16}	ASI_QUEUE	RW (hyperpriv) RO (priv)	$3E8_{16}$	Y	Resumable Error Queue tail pointer	7.3.1
25_{16}	ASI_QUEUE	RW	$3F0_{16}$	Y	Nonresumable Error Queue head pointer	7.3.1
25_{16}	ASI_QUEUE	RW (hyper-priv) RO (priv)	$3F8_{16}$	Y	Nonresumable Error Queue tail pointer	7.3.1
26_{16}	ASI_TWINX_REAL	R	Any	—	128-bit atomic LDDA, real address	(See UA-2007)
27_{16}	ASI_TWINX_NUCLEUS, ASI_STBI_N	RW	Any	—	Load: 128-bit atomic load twin extended word from nucleus context Store: Block initializing store from nucleus context	(See UA-2007)
28_{16}–29_{16}			Any	—	*DAE_invalid_asi*	
$2A_{16}$	ASI_TWINX_AIUPL, ASI_STBI_AIUPL	RW	Any	—	Load: 128-bit atomic load twin extended word, primary address space, user privilege, little endian Store: Block initializing store, primary address space, user privilege, little endian	(See UA-2007)

TABLE 9-2 OpenSPARC T2 ASI Usage *(4 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
$2B_{16}$	ASI_TWINX_AIUSL, ASI_STBI_AIUSL	RW	Any	—	Load: 128-bit atomic load twin extended word, secondary address space, user privilege, little endian Store: Block initializing store, secondary address space, user privilege, little endian	(See UA-2007)
$2C_{16}$	ASI_TWINX_LITTLE	RO	Any	—	128-bit atomic load twin extended word, little endian	(See UA-2007)
$2D_{16}$			Any	—	*DAE_invalid_asi*	
$2E_{16}$	ASI_TWINX_REAL_LITTLE	RO	Any	—	128-bit atomic LDDA, real address (LE)	(See UA-2007)
$2F_{16}$	ASI_TWINX_NL, ASI_STBI_NL	RW	Any	—	Load: 128-bit atomic load twin extended word from nucleus context, little endian Store: Block initializing store from nucleus context, little endian	(See UA-2007)
30_{16}	ASI_AS_IF_PRIV_PRIMARY	RW	Any	—	Primary address space, privilege access	(See UA-2007)
31_{16}	ASI_AS_IF_PRIV_SECONDARY	RW	Any	—	Secondary address space, privilege access	(See UA-2007)
32_{16}–35_{16}			Any	—	*DAE_invalid_asi*	
36_{16}	ASI_AS_IF_PRIV_NUCLEUS	RW	Any	—	Nucleus address space, privilege access	(See UA-2007)
37_{16}			Any	—	*DAE_invalid_asi*	
38_{16}	ASI_AS_IF_PRIV_PRIMARY_LITTLE	RW	Any	—	Primary address space, privileged access, little endian	(See UA-2007)
39_{16}	ASI_AS_IF_PRIV_SECONDARY_LITTLE	RW	Any	—	Secondary address space, privilege access, little endian	(See UA-2007)
$3A_{16}$–$3D_{16}$			Any	—	*DAE_invalid_asi*	
$3E_{16}$	ASI_AS_IF_PRIV_NUCLEUS_LITTLE	RW	Any	—	Nucleus address space, privilege access, little endian	(See UA-2007)
$3F_{16}$			Any	—	*DAE_invalid_asi*	

TABLE 9-2 OpenSPARC T2 ASI Usage *(5 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
41_{16}	ASI_CMP	RO	0	S	Strand available	14.1.1
41_{16}	ASI_CMP	RO	10_{16}	S	Strand enable status	14.1.2
41_{16}	ASI_CMP	RW	20_{16}	S	Strand enable	14.1.3
41_{16}	ASI_CMP	RW	30_{16}	S	XIR steering	14.1.4
41_{16}	ASI_CMP	RW	38_{16}	S	Tick_Enable	14.1.5
41_{16}	ASI_CMP	RW	50_{16}	S	Running_RW	14.1.7
41_{16}	ASI_CMP	RO	58_{16}	S	Running Status	14.1.8
41_{16}	ASI_CMP	WO	60_{16}	S	Running_W1S	14.1.9
41_{16}	ASI_CMP	WO	68_{16}	S	Running_W1C	14.1.10
42_{16}	ASI_INST_MASK_REG	RW	8_{16}	N	SPARC Instruction Mask register	28.3.1
42_{16}	ASI_LSU_DIAG_REG	RW	10_{16}	N	Load/Store Unit Diagnostic register	28.4.1
43_{16}	ASI_ERROR_INJECT_REG	RW	0	N	ASI_ERROR_INJECT_REG	25.8.9
44_{16}			Any		*DAE_invalid_asi*	
45_{16}	ASI_LSU_CONTROL_REG	RW	0	Y	Load/Store Unit Control register	28.1
45_{16}	ASI_DECR	RW	8_{16}	N	Trap unit	25.7.10
45_{16}	ASI_RST_VEC_MASK	RW	18_{16}	S	SOC	29.1.4
46_{16}	ASI_DCACHE_DATA	RW	Any	Y	Dcache data array diagnostic access	28.6.1
47_{16}	ASI_DCACHE_TAG	RW	Any	Y	Dcache tag and valid bit diagnostic access	28.6.2
48_{16}	ASI_IRF_ECC_REG	RO	Any	Y	IRF ECC diagnostic access	28.7.1
49_{16}	ASI_FRF_ECC_REG	RO	Any	Y	FRF ECC diagnostic access	28.8.1
$4A_{16}$	ASI_STB_ACCESS	RO	Any	Y	Store buffer diagnostic access	28.9
$4B_{16}$			Any	—	*DAE_invalid_asi*	
$4C_{16}$	ASI_DESR	RO	0	Y	Disrupting Error Status register (DESR)	25.8.5
$4C_{16}$	ASI_DFESR	RO	8_{16}	Y	Deferred Error Status register	25.8.6
$4C_{16}$	ASI_CERER	RW	10_{16}	N	ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER	25.8.1

TABLE 9-2 OpenSPARC T2 ASI Usage *(6 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
$4C_{16}$	ASI_SETER	RW	18_{16}	Y	ASI_STRAND_ERROR_TRAP_ENABLE_REGISTER	25.8.2
$4C_{16}$	ASI_CLESR	RO	20_{16}	Y	ASI_CLESR	25.8.7
$4C_{16}$	ASI_CLFESR	RO	28_{16}	Y	ASI_CLFESR	25.8.8
$4D_{16}$					*DAE_invalid_asi*	
$4E_{16}$	ASI_SPARC_PWR_MGMT	RW	0	N	SPARC power management register	27.1
$4F_{16}$	ASI_HYP_SCRATCHPAD	RW	0_{16}–38_{16}	Y	Hypervisor scratchpad	page 77
50_{16}	ASI_ITSB_TAG_TARGET	RO	0	Y	IMMU Tag Target register	12.10.3
50_{16}	ASI_ISFSR	RW	18	Y	IMMU Synchronous Fault Status register	25.8.3
50_{16}	ASI_ITLB_TAG_ACCESS	RW	30	Y	IMMU TLB Tag Access register	12.10.5
50_{16}	ASI_IMMU_VA_WATCHPOINT	RW	38	Y	IMMU Watchpoint register	28.2.2
51_{16}	ASI_MRA_ACCESS	RO		Y	HWTW MRA access	28.13
52_{16}	ASI_MMU_REAL_RANGE_0	RW	108_{16}	Y	MMU TSB real range 0	12.10.9
52_{16}	ASI_MMU_REAL_RANGE_1	RW	110_{16}	Y	MMU TSB real range 1	12.10.9
52_{16}	ASI_MMU_REAL_RANGE_2	RW	118_{16}	Y	MMU TSB real range 2	12.10.9
52_{16}	ASI_MMU_REAL_RANGE_3	RW	120_{16}	Y	MMU TSB real range 3	12.10.9
52_{16}	ASI_MMU_PHYSICAL_OFFSET_0	RW	208_{16}	Y	MMU TSB physical offset 0	12.10.10
52_{16}	ASI_MMU_PHYSICAL_OFFSET_1	RW	210_{16}	Y	MMU TSB physical offset 1	12.10.10
52_{16}	ASI_MMU_PHYSICAL_OFFSET_2	RW	218_{16}	Y	MMU TSB physical offset 2	12.10.10
52_{16}	ASI_MMU_PHYSICAL_OFFSET_3	RW	220_{16}	Y	MMU TSB physical offset 3	12.10.10
53_{16}	ASI_ITLB_PROBE	RO		Y	ITLB Probe	12.10.8
54_{16}	ASI_ITLB_DATA_IN_REG	WO	0, 400_{16}	Y	IMMU data in register	12.10.15
54_{16}	ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_0	RW	10_{16}	Y	Context zero TSB Config 0	12.10.11
54_{16}	ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_1	RW	18_{16}	Y	Context zero TSB Config 1	12.10.11
54_{16}	ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_2	RW	20_{16}	Y	Context zero TSB Config 2	12.10.11
54_{16}	ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_3	RW	28_{16}	Y	Context zero TSB Config 3	12.10.11

TABLE 9-2 OpenSPARC T2 ASI Usage *(7 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
54_{16}	ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_0	RW	30_{16}	Y	Context nonzero TSB Config 0	12.10.11
54_{16}	ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_1	RW	38_{16}	Y	Context nonzero TSB Config 1	12.10.11
54_{16}	ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_2	RW	40_{16}	Y	Context nonzero TSB Config 2	12.10.11
54_{16}	ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_3	RW	48_{16}	Y	Context nonzero TSB Config 3	12.10.11
54_{16}	ASI_MMU_ITSB_PTR_0	RO	50_{16}	Y	I-ITSB Pointer 0	12.10.12
54_{16}	ASI_MMU_ITSB_PTR_1	RO	58_{16}	Y	I-TSB Pointer 1	12.10.12
54_{16}	ASI_MMU_ITSB_PTR_2	RO	60_{16}	Y	I-TSB Pointer 2	12.10.12
54_{16}	ASI_MMU_ITSB_PTR_3	RO	68_{16}	Y	I-TSB Pointer 3	12.10.12
54_{16}	ASI_MMU_DTSB_PTR_0	RO	70_{16}	Y	D-TSB Pointer 0	12.10.12
54_{16}	ASI_MMU_DTSB_PTR_1	RO	78_{16}	Y	D-TSB Pointer 1	12.10.12
54_{16}	ASI_MMU_DTSB_PTR_2	RO	80_{16}	Y	D-TSB Pointer 2	12.10.12
54_{16}	ASI_MMU_DTSB_PTR_3	RO	88_{16}	Y	D-TSB Pointer 3	12.10.12
54_{16}	ASI_PENDING_TABLEWALK_CONTROL	RW	90_{16}	Y	Pending tablewalk control	12.10.13
54_{16}	ASI_PENDING_TABLEWALK_STATUS	RO	98_{16}	N	Pending Tablewalk status	12.10.14
55_{16}	ASI_ITLB_DATA_ACCESS_REG	RW	$0–1F8_{16}$, $400_{16}–5F8$	Y	IMMU TLB Data Access register	12.10.15
56_{16}	ASI_ITLB_TAG_READ_REG	RO	$0–1F8_{16}$, $400_{16}–5F8_{16}$	Y	IMMU TLB Tag Read register	12.10.15
57_{16}	ASI_IMMU_DEMAP	WO	Any	Y	IMMU TLB demap	12.11.1
58_{16}	ASI_DTSB_TAG_TARGET	RO	0	Y	DMMU Tag Target register	12.10.3
58_{16}	ASI_DSFSR	RW	18_{16}	Y	DMMU Synchronous Fault Status register	25.8.4.1
58_{16}	ASI_DSFAR	RO	20_{16}	Y	DMMU Synchronous Fault Address register	25.8.4.2
58_{16}	ASI_DTLB_TAG_ACCESS	RW	30_{16}	Y	DMMU TLB Tag Access register	12.10.5
58_{16}	ASI_DMMU_WATCHPOINT	RW	38_{16}	Y	DMMU Watchpoint register	28.2.1
58_{16}	ASI_HWTW_CONFIG	RW	40_{16}	Y	I/DMMU Hardware Tablewalk Config register	12.10.7
58_{16}	ASI_PARTITION_ID	RW	80_{16}	Y	I/DMMU Partition ID	12.10.6

TABLE 9-2 OpenSPARC T2 ASI Usage *(8 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
59_{16}	ASI_SCRATCHPAD_ ACCESS	RO	Any	Y	Scratchpad register diagnostic access	28.10
$5A_{16}$	ASI_TICK_ACCESS	RO	$0–8_{16}$, 10_{16}, $20_{16}–30_{16}$	Y	TICK register diagnostic access	28.11
$5B_{16}$	ASI_TSA_ACCESS	RO	Any	Y	TSA diagnostic access	28.12
$5C_{16}$	ASI_DTLB_DATA_IN_REG	WO	$0,400_{16}$	Y	DMMU data-in register	12.10.15
$5D_{16}$	ASI_DTLB_DATA_ACCESS_R EG	RW	$0–7F8_{16}$	Y	DMMU TLB Data Access register	12.10.15
$5E_{16}$	ASI_DTLB_TAG_READ_REG	RO	$0–7F8_{16}$	Y	DMMU TLB Tag Read register	12.10.15
$5F_{16}$	ASI_DMMU_DEMAP	WO	Any	Y	DMMU TLB demap	12.11
$60_{16}–62_{16}$			Any		*DAE_invalid_asi*	
63_{16}	ASI_CMT_CORE_INTR_ID	RO	0	Y	Strand interrupt ID	14.2.1
63_{16}	ASI_CMT_STRAND_ID	RO	10_{16}	Y	Strand ID	14.2.2
$64_{16}–65_{16}$			Any		*DAE_invalid_asi*	
66_{16}	ASI_ICACHE_INSTR	RW	Any	Y	Icache data array diagnostics access	28.5.1
67_{16}	ASI_ICACHE_TAG	RW	Any	Y	Icache tag and valid bit diagnostics access	28.5.2
$68_{16}–71_{16}$			Any		*DAE_invalid_asi*	
72_{16}	ASI_INTR_RECEIVE	RW	0	Y	Interrupt Receive register	7.3.2
73_{16}	ASI_INTR_W	WO	0	Y	Interrupt Vector Dispatch register	7.3.3
74_{16}	ASI_INTR_R	RO	0	Y	Incoming Vector register	7.3.4
$75_{16}–7F_{16}$			Any	—	*DAE_invalid_asi*	
80_{16}	ASI_PRIMARY	RW	Any	—	Implicit primary address space	(See UA-2007)
81_{16}	ASI_SECONDARY	RW	Any	—	Implicit secondary address space	(See UA-2007)
82_{16}	ASI_PRIMARY_NO_FAULT	RO	Any	—	Primary address space, no fault	(See UA-2007)
83_{16}	ASI_SECONDARY_NO_ FAULT	RO	Any	—	Secondary address space, no fault	(See UA-2007)
$84_{16}–87_{16}$			Any	—	*DAE_invalid_asi*	
88_{16}	ASI_PRIMARY_LITTLE	RW	Any	—	Implicit primary address space (LE)	(See UA-2007)
89_{16}	ASI_SECONDARY_LITTLE	RW	Any	—	Implicit secondary address space (LE)	(See UA-2007)

TABLE 9-2 OpenSPARC T2 ASI Usage *(9 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
$8A_{16}$	ASI_PRIMARY_NO_FAULT_LITTLE	RO	Any	—	Primary address space, no fault (LE)	(See UA-2007)
$8B_{16}$	ASI_SECONDARY_NO_FAULT_LITTLE	RO	Any	—	Secondary address space, no fault (LE)	(See UA-2007)
$8C_{16}$–BF_{16}			Any	—	*DAE_invalid_asi*	
$C0_{16}$	ASI_PST8_P		Any	—	Eight 8-bit conditional stores, primary address	(See UA-2007)
$C1_{16}$	ASI_PST8_S		Any	—	Eight 8-bit conditional stores, secondary address	(See UA-2007)
$C2_{16}$	ASI_PST16_P		Any	—	Four 16-bit conditional stores, primary address	(See UA-2007)
$C3_{16}$	ASI_PST16_S		Any	—	Four 16-bit conditional stores, secondary address	(See UA-2007)
$C4_{16}$	ASI_PST32_P		Any	—	Two 32-bit conditional stores, primary address	(See UA-2007)
$C5_{16}$	ASI_PST32_S		Any	—	Two 32-bit conditional stores, secondary address	(See UA-2007)
$C6_{16}$–$C7_{16}$			Any	—	*DAE_invalid_asi*	
$C8_{16}$	ASI_PST8_PL		Any	—	Eight 8-bit conditional stores, primary address, little endian	(See UA-2007)
$C9_{16}$	ASI_PST8_SL		Any	—	Eight 8-bit conditional stores, secondary address, little endian	(See UA-2007)
CA_{16}	ASI_PST16_PL		Any	—	Four 16-bit conditional stores, primary address, little endian	(See UA-2007)
CB_{16}	ASI_PST16_SL		Any	—	Four 16-bit conditional stores, secondary address, little endian	(See UA-2007)
CC_{16}	ASI_PST32_PL		Any	—	Two 32-bit conditional stores, primary address, little endian	(See UA-2007)
CD_{16}	ASI_PST32_SL		Any	—	Two 32-bit conditional stores, secondary address, little endian	(See UA-2007)
CE_{16}–CF_{16}			Any	—	*DAE_invalid_asi*	
$D0_{16}$	ASI_FL8_P		Any	—	8-bit load/store, primary address	(See UA-2007)
$D1_{16}$	ASI_FL8_S		Any	—	8-bit load/store, secondary address	(See UA-2007)

TABLE 9-2 OpenSPARC T2 ASI Usage *(10 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
$D2_{16}$	ASI_FL16_P		Any	—	16-bit load/store, primary address	(See UA-2007)
$D3_{16}$	ASI_FL16_S		Any	—	16-bit load/store, secondary address	(See UA-2007)
$D4_{16}$–$D7_{16}$			Any	—	*DAE_invalid_asi*	
$D8_{16}$	ASI_FL8_PL		Any	—	8-bit load/store, primary address, little endian	(See UA-2007)
$D9_{16}$	ASI_FL8_SL		Any	—	8-bit load/store, secondary address, little endian	(See UA-2007)
DA_{16}	ASI_FL16_PL		Any	—	16-bit load/store, primary address, little endian	(See UA-2007)
DB_{16}	ASI_FL16_SL		Any	—	16-bit load/store, secondary address, little endian	(See UA-2007)
DC_{16}–DF_{16}			Any	—	*DAE_invalid_asi*	
$E0_{16}$	ASI_BLK_COMMIT_PRIMARY	RW	Any	—	64-byte block commit store, primary address	5.3
$E1_{16}$	ASI_BLK_COMMIT_SECONDARY	RW	Any	—	64-byte block commit store, secondary address	5.3
$E2_{16}$	ASI_TWINX_P, ASI_STBI_P	RW	Any	—	Load: 128-bit atomic load twin extended word, primary address space Store: Block initializing store, primary address space	(See UA-2007)
$E3_{16}$	ASI_TWINX_S, ASI_STBI_S	RW	Any	—	Load: 128-bit atomic load twin extended word, sedondary address space Store: Block initializing store, sedondary address space	(See UA-2007)
$E4_{16}$–$E9_{16}$			Any	—	*DAE_invalid_ASI*	
EA_{16}	ASI_TWINX_PL, ASI_STBI_PL	RW	Any	—	Load: 128-bit atomic load twin extended word, primary address space, little endian Store: Block initializing store, primary address space, little endian	(See UA-2007)

TABLE 9-2 OpenSPARC T2 ASI Usage *(11 of 11)*

ASI	ASI Name	R/W	VA	Copy per Strand	Description	Section/Page
EB_{16}	ASI_TWINX_PL, ASI_STBI_PL	RW	Any	—	Load: 128-bit atomic load twin extended word, sedondary address space, little endian Store: Block initializing store, sedondary address space, little endian	(See UA-2007)
EC_{16}–EF_{16}			Any	—	*DAE_invalid_asi*	
$F0_{16}$	ASI_BLK_P	RW	Any	—	64-byte block load/store, primary address	5.3
$F1_{16}$	ASI_BLK_S	RW	Any	—	64-byte block load/store, secondary address	5.3
$F2_{16}$–$F7_{16}$			Any	—	*DAE_invalid_asi*	
$F8_{16}$	ASI_BLK_PL	RW	Any	—	64-byte block load/store, primary address (LE)	5.3
$F9_{16}$	ASI_BLK_SL	RW	Any	—	64-byte block load/store, secondary address (LE)	5.3
FA_{16}–FF_{16}			Any	—	*DAE_invalid_asi*	

9.2.1 ASI_REAL, ASI_REAL_LITTLE, ASI_REAL_IO, and ASI_REAL_IO_LITTLE

These ASIs are used to bypass the VA-to-RA translation. For these ASIs, the real address is set equal to the truncated virtual address (that is, RA{39:0} ← VA{39:0}), and the attributes used are those present in the matching TTE. The hypervisor will normally set the TTE attributes for ASI_REAL and ASI_REAL_LITTLE to cacheable (cp = 1) and for ASI_REAL_IO and ASI_REAL_IO_LITTLE to noncacheable, with side effect (cp = 0, e = 1).

9.2.2 ASI_SCRATCHPAD

Each virtual processor has a set of six privileged ASI_SCRATCHPAD registers at ASI 20_{16} with VA{63:} = 0_{16}–18_{16}, 30_{16}–38_{16}. These registers are for scratchpad use by privileged software.

OpenSPARC T2 Implementation Note Standard support of the ASI_SCRATCHPAD is eight registers, so accesses to VA 20_{16} and 28_{16} cause a *DAE_invalid_asi* trap to allow hyperprivileged software to emulate the additional registers.

OpenSPARC T2 Implementation Note

There is only a single set of eight scratchpad registers, which are accessible via both ASI_SCRATCHPAD and ASI_HYP_SCRATCHPAD. ASI_SCRATCHPAD is intended to used primarily by privileged code, and only has access to the first four and last two registers of the eight entry scratchpad array. ASI_HYP_SCRATCHPAD can only be accessed when hyperprivileged and has full access to all eight scratchpad registers. Note that the registers at VA 20_{16} and 28_{16} are exclusively (directly) accessible via ASI_HYP_SCRATCHPAD.

9.2.3 ASI_HYP_SCRATCHPAD

Each virtual processor has a set of eight hyperprivileged ASI_HYP_SCRATCHPAD registers at ASI $4F_{16}$, VA{63:0} = 0_{16}–$3_{16}8$. These registers are for scratchpad use by the hypervisor and for aliased access to the supervisor scratchpad registers.

OpenSPARC T2 Implementation Note

There is only a single set of eight scratchpad registers, which are accessible via both ASI_SCRATCHPAD and ASI_HYP_SCRATCHPAD. ASI_SCRATCHPAD is intended to used primarily by privileged code and only has access to the first four and last two registers of the eight entry scratchpad array. ASI_HYP_SCRATCHPAD can only be accessed when hyperprivileged, and has full access to all eight scratchpad registers. Note that the registers at VA 20_{16} and 28_{16} are exclusively (directly) accessible via ASI_HYP_SCRATCHPAD.

CHAPTER **10**

Performance Instrumentation

10.1 SPARC Performance Control Register

Each virtual processor has a privileged Performance Control register. Nonprivileged accesses to this register cause a *privileged_opcode* trap. The Performance Control register contains thirteen fields: hold_ov1, hold_ov0, ov1, sl1, mask1, ov0, sl0, mask0, toe, ht, ut, st, and priv. hold_ov1 and hold_ov0 read as 0 and control whether ov1 and ov0, respectively, are updated on a write. All bits except ov1 and ov0 are always updated on a Performance Control register write. ov1 and ov0 are state bits associated with the PIC.h and PIC.l overflow traps and are provided to allow software to determine which PIC counter has overflowed. sl1 and sl0 controls which events are counted in PIC.h and PIC.l, respectively. mask1 (mask0) is used in conjunction with sl1 (sl0) in determining which set of subevents are counted in PIC.h (PIC.l). toe controls whether a trap is generated when the PIC counter overflows. ut controls whether user-level events are counted. st controls whether supervisor-level events are counted. ht controls whether hypervisor level events are counted. priv controls whether the PIC register can be read or written by nonprivileged software. The format of this register is shown in TABLE 10-1. Note that changing the fields in PCR does not affect the PIC values. To change the events monitored, software needs to disable counting via PCR, reset the PIC, and then enable the new event via the PCR.

Note | As the ht bit controls the counting of hyperprivileged events, writes to this bit while privileged are ignored.

TABLE 10-1 Performance Control Register – PCR (ASR 10_{16})

Bit	Field	Initial Value	R/W	Description
63	hold_ov1	0	Write to 0 or 1, reads as 0	If set to 0 on a write, update ov1 from bit 31 of the write data; else, don't update ov1. In this case ov1 holds its previous value.
62	hold_ov0	0	Write to 0 or 1, reads as 0	If set to 0 on a write, update ov0 from bit 18 of the write data; else, don't update ov0. In this case ov0 holds its previous value.
61:32	—	0	RO	*Reserved*
31	ov1	0	RW	Set to 1 when PIC.h wraps from 2^{32} –1 to 0, or when PIC.h is within --16..-1 inclusively, and an event occurs which causes PIC.h to increment. Once set, ov1 remains set until reset by software.
30:27	sl1	0	RW	Selects 1 of 16 events to be counted for PIC.h as per the following table.
26:19	mask1	0	RW	Mask event for PIC.h as listed in TABLE 10-2.
18	ov0	0	RW	Set to 1 when PIC.l wraps from 2^{32} –1 to 0, or when PIC.l is within --16..-1 inclusively, and an event occurs which causes PIC.l to increment. Once set, ov0 remains set until reset by software.
17:14	sl0	0	RW	Selects one of sixteen events to be counted for PIC.l as per the following table.
13:6	mask0	0	RW	Mask event for PIC.l as listed in TABLE 10-2.
5:4	toe	0	RW	Trap-on-Event: This field controls whether a disrupting *pic_overflow* trap will occur if the corresponding counter overflows. toe{1} corresponds to ov1, and toe{0} to ov0. Hardware will **and** the value of toe{*i*} with ov{*i*} to produce a trap. Events in event groups 2) and 3) are "precisely" trapped, assuming *pic_overflow* traps are enabled - TPC will contain the address of an instruction that generated a count event. If *pic_overflow* traps are not enabled when the counter overflows, TPC will contain the address of the instruction to be executed next when the trap is eventually taken. Events in other event groups are not directly related to the instruction stream; therefore, the TPC may be some number of instructions later than when the overflow event occurred.
3	ht	0	RO (priv) RW (hyperpriv)	If ht = 1, count events in hyperprivileged mode; otherwise, ignore hyperprivileged mode events.
2	ut	0	RW	If ut = 1, count events in user mode; otherwise, ignore user mode events.
1	st	0	RW	If st = 1, count events in privileged mode; otherwise, ignore privileged mode events.
0	priv	0	RW	If priv = 1, prevent access to PIC by user-level code. If priv = 0, allow access to PIC by user-level code.

Note that hold_ov1 and hold_ov0 control whether ov1 and ov0, respectively, are updated when a write occurs. All of the 4 combinations of the hold_ov1 and hold_ov0 fields are supported: both, either, or none of the ov1 and ov0 bits can be

updated independently. This allows software to avoid a race condition that may occur, for example, if ov1 is set when the trap handler is entered, then ov0 is set by a counter overflow between the time software reads PCR and resets ov1 prior to leaving the trap handler.

TABLE 10-2 describes the settings of the sl0 and sl1 fields. Note that with the exception of sl = 0, all events correspond to a given strand. Most sl fields have a mask associated with them. Setting multiple mask bits at the same time can lead to multiple events being counted as one event. More details are described in TABLE 10-2.

TABLE 10-2 sl Field Settings *(1 of 3)*

sl	mask	Event	Description
0	—	All strands idle	Count cycles when no strand can be picked for the physical core on which the monitoring strand resides.[2]
1	—	—	*Reserved*
2	01_1	Completed branches	
	02_{16}	Taken branches	Taken branches are always mispredicted[3]
	04_{16}	FGU arithmetic instructions	All FADD, FSUB, FCMP, convert, FMUL, FDIV, FNEG, FABS, FSQRT, FMOV, FPADD, FPSUB, FPACK, FEXPAND, FPMERGE, FMUL8, FMULD8, FALIGNDATA, BSHUFFLE, FZERO, FONE, FSRC, FNOT1, FNOT2, FOR, FNOR, FAND, FNAND, FXOR, FXNOR, FORNOT1, FORNOT2, FANDNOT1, FANDNOT2, PDIST, SIAM.
	08_{16}	Load instructions	
	10_{16}	Store instructions	
	20_{16}	`sethi %hi(fc000`$_{16}$`), %g0`	Software count instructions.
	40_{16}	Other instructions	
	80_{16}	Atomics	Atomics are LDSTUB/A, CASA/XA, SWAP/A
	Any other value 03_{16}– FF_{16}	Any subset of instructions	Count instruction types identified by a 1 in the corresponding mask register bit; e.g., FD_{16} counts all instructions. Certain instructions (e.g., LDSTUB, CAS, SWAP) are decoded as both Load and Store instructions.

TABLE 10-2 sl Field Settings *(2 of 3)*

sl	mask	Event	Description
3	01_{16}	Icache misses	**Note:** This counts only primary instruction cache misses, and does not count duplicate instruction cache misses.[4] Also, only "true" misses are counted. If a thread encounters an I$ miss, but the thread is redirected (due to a branch misprediction or trap, for example) before the line returns from L2 and is loaded into the I$, then the miss is not counted.
	02_{16}	Dcache misses	Note: This counts both primary and duplicate data cache misses.[4]
	04_{16}	—	Undefined operation.
	08_{16}	—	Undefined operation.
	10_{16}	L2 cache instruction misses	
	20_{16}	L2 cache load misses	**Note:** Block loads are treated as one L2 miss event. In reality, each individual load can hit or miss in the L2 since the block load is not atomic.
	03_{16}, 11_{16}, 12_{16}, 13_{16}, 21_{16}, 22_{16}, 23_{16}, 30_{16}, 31_{16}, 32_{16}, 33_{16}	Subset of misses	Count subset of misses identified by a '1' in corresponding mask bit; e.g., 23_{16} counts I-cache, D-cache, and L2 load misses; this counter can advance at most 1 per cycle. **Note:** Instructions that get both an I-Cache miss (or an L2 cache instruction miss) and a D-Cache miss (or L2 cache load miss) count as one event.
	Any other value	—	*Reserved*, Undefined operation.
4	01_{16}	—	*Reserved*
	02_{16}	—	*Reserved*
	04_{16}	ITLB references to L2	For each ITLB miss with hardware tablewalk enabled, count each access the ITLB hardware tablewalk makes to L2.
	08_{16}	DTLB references to L2	For each DTLB miss with hardware tablewalk enabled, count each access the DTLB hardware tablewalk makes to L2.
	10_{16}	ITLB references to L2 which miss in L2	For each ITLB miss with hardware tablewalk enabled, count each access the ITLB hardware tablewalk makes to L2 which misses in L2. **Note:** Depending upon the hardware table walk configuration, each ITLB miss may issue from 1 to 4 requests to L2 to search TSBs.
	20_{16}	DTLB references to L2 which miss in L2	For each DTLB miss with hardware tablewalk enabled, count each access the DTLB hardware tablewalk makes to L2 which misses in L2. **Note:** Depending upon the hardware tablewalk configuration, each DTLB miss may issue from 1 to 4 requests to L2 to search TSBs.
	C_{16}, 14_{16}, 18_{16}, $1C_{16}$, 24_{16}, 28_{16}, $2C_{16}$, 34_{16}, 38_{16} $3C_{16}$	Subset of above events	Count subset of misses identified by a 1 in corresponding mask bit; e.g., 14_{16} counts ITLB and DTLB hardware tablewalk references to L2; this counter can advance at most 1 per cycle. Certain combinations (14_{16}, 28_{16}, 34_{16}, 38_{16}, $3C_{16}$) are likely not useful.
	Any other value	—	*Reserved*. Undefined operation.

TABLE 10-2 sl Field Settings *(3 of 3)*

sl	mask	Event	Description
5	01_{16}–02_{16}	—	*Reserved*
	04_{16}	CPU Load to PCX	Count CPU loads to L2.
	08_{16}	CPU I-fetch to PCX	Count I-fetches to L2.
	10_{16}	CPU Store to PCX	Count CPU stores to L2.
	20_{16}	MMU Load to PCX	Count MMU loads to L2.
	Any other value 03_{16}–$3F_{16}$	Subset of PCX requests	Count subset of PCX requests identified by a '1' in corresponding mask bit; e.g., $3F_{16}$ counts all PCX requests; this counter increments at most one per cycle.
	40_{16}–FF_{16}	—	*Reserved*
6[1]	00_{16}–$3F_{16}$	—	*Reserved*
	40_{16}–FF_{16}	—	*Reserved*
7[1]	00_{16}–$3F_{16}$	—	*Reserved*
	40_{16}–FF_{16}	—	*Reserved*
8-10	—	—	*Reserved*
11	04_{16}	ITLB misses	Includes all misses (successful and unsuccessful tablewalks).
	08_{16}	DTLB misses	Includes all misses (successful and unsuccessful tablewalks).
	$0C_{16}$	TLB misses	Count both ITLB and DTLB misses, including successful and unsuccessful tablewalks.
	Any other value	—	*Reserved.* Undefined operation.
12-15	—	—	*Reserved*

1. PCR.UT, PCR.HT, and PCR.ST must all be set in order to properly count events in groups 6 and 7.
2. Unrestricted access to performance events for sl field setting 0 may have security implications since they contain information about other strands. OS software can protect against unrestricted access by setting the PCR.priv bit. Hypervisor software can protect against unrestricted access by not having partitions span an eight-strand boundary.
3. In conjunction with the completed branch count, the taken branch count can be used to compute not-taken prediction accuracy. Also it can be used to sum idle cycles in single-strand mode by assuming a fixed number of pipeline bubble cycles per mispredicted branch.
4. A duplicate miss is a miss for which another thread has already missed in the cache for the line, and the cache fill is pending. UltraSPARC {N2} does not count duplicate I-cache misses but does count duplicate D-cache misses.

10.2 SPARC Performance Instrumentation Counter

Each virtual processor has a Performance Instrumentation Counter register. Access privilege is controlled by the setting of PCR.priv. When PCR.priv = 1 an attempt to access this register in nonprivileged mode causes a *privileged_action* trap.

The PIC counter contains two fields: h and l. The h field counts the event select by PCR.sl1. The l field counts the event selected by PCR.sl0. The ut, st, and ht fields for PCR control which combination of user, supervisor, and/or hypervisor events are counted.

For the setting sl0 (sl1) = 2 and sl0 (sl1) = 3, when a counter overflow occurs for the event, hardware generates a disrupting *pic_overflow* trap that is guaranteed to occur immediately before an instruction that generated a count event. This instruction causing the counter to be within epsilon[1] of overflow will not have been executed, and the PC and NPC of the instruction will be stored on the trap stack, assuming the *pic_overflow* trap is enabled and is the highest priority trap when the counter overflows[2]. The corresponding PIC counter will be incremented. In addition, the ov0 or ov1 bit (depending on which counter overflowed) will be set to help software determine which counter overflowed. ov0 and ov1 can be cleared independently by a write that sets the bit to 0 (see TABLE 10-1 on page 80 above).

For other settings of sl0 (sl1), the trap will not be “precise” to the instruction causing the counter overflow. The amount of skid possible is TBD.

Counter overflow is recorded in the ov0 or ov1 bit of the counter as well as in bit 15 of the SOFTINT register. The overflow causes a disrupting *pic_overflow* exception. The strand takes a *pic_overflow* trap if PSTATE.ie is set and the value in the Processor Interrupt Level (PIL) is less than 15. The *pic_overflow* priority of 16.0 is higher than the *interrupt_level_15* trap priority of 17.

The format of the PIC register is shown in TABLE 10-3.

1. The definition of epsilon is –16 to -1, inclusive, instructions generating the event being counted before/after the overflow. The PC is guaranteed to point to an instruction that generated the event being counted, if the trap is taken when the counter "overflows".

2. In certain corner cases, the counter will not be incremented nor will the corresponding ov0/1 bit be set. These cases are: [a) the counter is in range and the instruction will cause the counter to increment, or, b) the OV bit is already set and the instruction will cause the counter to increment,] and c) one of the following four pending disrupting conditions are present: i) "disrupting single step completion" exception (this only occurs for special ’replay’ conditions that occur in Single Step mode, which is a debug mode controlled via JTAG), or ii) XIR request, or iii) store_error trap request, or iv) SIR (i.e., this instruction is an SIR instruction).

TABLE 10-3 Performance Instrumentation Counter Register – PIC (ASR 11_{16})

Bit	Field	Initial Value	R/W	Description
63:32	h	0	RW	Programmable event counter, event controlled by PCR.sl1.
31:0	l	0	RW	Programmable event counter, event controlled by PCR.sl0.

10.3 DRAM Performance Counter

Each DRAM channel has a pair of performance counters, packed into a single register, plus a register to control what is counted. The counters count all events for that particular DRAM channel, which corresponds to traffic from a pair of L2 banks.

TABLE 10-4 DRAM Performance Control Register – DRAM_PERF_CTL_REG (84_{16}-0000_{16}-0400_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	X	RO	*Reserved*
7:4	sel0	0	RW	Select code for performance counter 0.
3:0	sel1	0	RW	Select code for performance counter 1.

TABLE 10-5 DRAM Performance Counter Register – DRAM_PERF_COUNT_REG (84_{16}-0000_{16}-0408_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63	sticky0	0	RW	Sticky overflow for counter 0.
62:32	counter0	0	RW	Performance counter 0
31	sticky1	0	RW	Sticky overflow for counter 1.
30:0	counter1	0	RW	Performance counter 1.

TABLE 10-6 DRAM Performance Counter Select Codes

Select	Description
0000	Read transactions.
0001	Write transactions.
0010	Read + write transactions.
0011	Bank busy stalls; incremented by 1 each cycle there are requests in the queue, but none can issue because of bank conflicts

TABLE 10-6 DRAM Performance Counter Select Codes

Select	Description
0100	Read queue latency; incremented by *n* each cycle, where *n* is the number of read transactions in the queue.
0101	Write queue latency; incremented by *n* each cycle, where *n* is the number of write transactions in the queue
0110	(Read + Write) queue latency; incremented by *n* each cycle, where *n* is the number of transactions in the queue
0111	Writeback buffer hits; incremented by 1 each time a read transaction is deferred because it conflicts with a queued write transaction.
1xxx	*Reserved*

CHAPTER 11

Implementation Dependencies

11.1 SPARC V9 General Information

11.1.1 Level-2 Compliance (Impdep #1)

OpenSPARC T2 is designed to meet Level-2 SPARC V9 compliance. It

- Correctly interprets all nonprivileged operations, and
- Correctly interprets all privileged elements of the architecture.

Note | System emulation routines (for example, quad-precision floating-point operations) shipped with OpenSPARC T2 also must be Level-2 compliant.

11.1.2 Unimplemented Opcodes, ASIs, and ILLTRAP

SPARC V9 unimplemented, *reserved*, ILLTRAP opcodes, and instructions with invalid values in *reserved* fields (other than *reserved* FPops) encountered during execution cause an *illegal_instruction* trap. Unimplemented and *reserved* ASI values cause a *DAE_invalid_ASI* trap.

11.1.3 Trap Levels (Impdep #37, 38, 39, 40, 114, 115)

OpenSPARC T2 supports two privileged trap levels and six hyperprivileged trap levels; that is, *MAXPTL* = 2 and at *MAXTL* = 6. Normal execution is at TL = 0. Traps at *MAXTL* − 1 cause the virtual processor to enter `RED_state`. If a trap is generated while the virtual processor is operating at TL = *MAXTL*, the virtual processor will pass through `error_state` and generate a watchdog reset (WDR). Window traps that cause a watchdog reset trap still update CWP if they would have done so with no watchdog trap being generated.

A virtual processor normally executes at trap level 0 (`execute_state`, TL = 0). Per SPARC V9, a trap causes the virtual processor to enter the next higher trap level, which is a very fast and efficient process because there is one set of trap state registers for each trap level. After saving the most important machine states (PC, NPC, PSTATE) on the trap stack at this level, the trap (or error) condition is processed.

For a complete description of traps and `RED_state` handling, see *Machine State After Reset and in RED_State* on page 164.

Note | The `RED_state` trap vector address (*RSTVADDR*) is 256 Mbytes below the top of the virtual address space; this is, at virtual address FFFF FFFF F000 0000_{16}, which is passed through to physical address FF F000 0000_{16} in `RED_state`.

11.1.4 Trap Handling (Impdep #16, 32, 33, 35, 36, 44)

OpenSPARC T2 supports precise trap handling for all operations except for deferred and disrupting traps from hardware failures and interrupts. OpenSPARC T2 implements precise traps, interrupts, and exceptions for all instructions, including long-latency floating-point operations. Multiple traps levels are supported, allowing graceful recovery from faults. Three of the trap levels (zero through two) are provided for application and OS use. The remaining three levels are provided for hyperprivileged and `RED_state` use. OpenSPARC T2 can efficiently execute kernel code even in the event of multiple nested traps, promoting strand efficiency while dramatically reducing the system overhead needed for trap handling.

Four sets of global registers are provided, for use by TL0, TL1, TL2, and TL3-5. This further increases OS performance, providing fast trap execution by avoiding the need to save and restore registers while processing exceptions.

All traps supported in OpenSPARC T2 are listed in TABLE 6-2 on page 38.

11.1.5 SIR Support (Impdep #116)

OpenSPARC T2 initiates a software-initiated reset (SIR) by executing a SIR instruction while in hyperprivileged mode. When executed in privileged or user mode, SIR generates an *illegal_instruction* trap. See also *Watchdog Reset (WDR) and error_state* on page 163.

11.1.6 Secure Software

To establish an enhanced security environment, it may be necessary to initialize certain virtual processor states between contexts. Examples of such states are the contents of integer and floating-point register files, condition codes, and state registers. See also *Clean Window Handling (Impdep #102)*.

11.1.7 Operation in Nonprivileged Mode with TL > 0

Operation with HPSTATE.hpriv = 0, PSTATE.priv = 0, and TL > 0 is invalid and will result in an *IAE_privilege_violation* trap on OpenSPARC T2.

11.1.8 Address Masking (Impdep #125)

OpenSPARC T2 follows UltraSPARC Architecture 2007 for PSTATE.am masking. In addition to the masking required by UltraSPARC Architecture 2007, addresses to non-translating ASIs and *REAL* ASIs are masked if PSTATE.am = 1. Translating accesses that bypass translation are also masked if PSTATE.am = 1.

11.2 SPARC V9 Integer Operations

11.2.1 Integer Register File and Window Control Registers (Impdep #2)

OpenSPARC T2 implements an eight-window 64-bit integer register file; that is, *N_REG_WINDOWS* = 8. OpenSPARC T2 truncates values stored in the CWP, CANSAVE, CANRESTORE, CLEANWIN, and OTHERWIN registers to three bits. This includes implicit updates to these registers by SAVE, SAVED, RESTORE, and RESTORED instructions. The most significant two bits of these registers read as zero.

11.2.2 Clean Window Handling (Impdep #102)

SPARC V9 introduced the concept of “clean window” to enhance security and integrity during program execution. A clean window is defined to be a register window that contains either all zeroes or addresses and data that belong to the current context. The CLEANWIN register records the number of available clean windows.

When a SAVE instruction requests a window and there are no more clean windows, a *clean_window* trap is generated. System software needs to clean one or more windows before returning to the requesting context.

11.2.3 Integer Multiply and Divide

Integer multiplications (MULScc, SMUL{cc}, MULX) and divisions (SDIV{cc}, UDIV{cc}, UDIVX) are executed directly in hardware.

11.2.4 MULScc

SPARC V9 does not define the value of xcc and rd{63:32] for MULScc. OpenSPARC T2 sets xcc.n to 0, xcc.z to 1 if rd{63:0} is zero and to 0 if rd{63:0} is not zero, xcc.v to 0, and xcc.c to 0. OpenSPARC T2 sets rd{63:33} to zeros, and sets rd{32} to icc.c (that is, rd{32} is set if there is a carry-out of rd{31}; otherwise, it is cleared).

11.2.5 Version Register (Impdep #2, 13, 101, 104)

Consult the product data sheet for the contents of the Version register for a specific OpenSPARC T2 implementation. The format of the Version register is described in *Hyperprivileged Version Register (hver)* on page 19.

11.3 SPARC V9 Floating-Point Operations

11.3.1 Subnormal Operands and Results; Nonstandard Operation

OpenSPARC T2 handles some cases of subnormal operands or results directly in hardware and traps on the rest. In the trapping cases, an *fp_exception_other* [fft = unfinished_FPop] trap is signaled and these operations are handled in system software.

Because trapping on subnormal operands and results can be quite costly, OpenSPARC T2 supports the nonstandard result option of the SPARC-V9 architecture. When the FSR.ns bit is set, subnormal operands or results encountered in trapping cases are flushed to zero and the unfinished_FPop floating-point trap is not taken.

11.3.2 Overflow, Underflow, and Inexact Traps (Impdep #3, 55)

OpenSPARC T2 implements precise floating-point exception handling. Underflow is detected before rounding. Prediction of overflow, underflow, and inexact traps for operations as well as prediction of invalid operation is used to simplify the hardware.

> Significant performance degradation may be observed while running with the inexact exception enabled.

11.3.3 Quad-Precision Floating-Point Operations (Impdep #3)

All quad-precision floating-point instructions, listed in TABLE 11-1, cause an *illegal_instruction* trap. These operations are then emulated by system software.

TABLE 11-1 Unimplemented Quad-Precision Floating-Point Instructions

Instruction	Description
F<s|d>TOq	Convert single-/double- to quad-precision floating-point.
F<i|x>TOq	Convert 32-/64-bit integer to quad-precision floating-point.
FqTO<s|d>	Convert quad- to single-/double-precision floating-point.
FqTO<i|x>	Convert quad-precision floating-point to 32-/64-bit integer.
FCMP<E>q	Quad-precision floating-point compares.
FMOVq	Quad-precision floating-point move.
FMOVqcc	Quad-precision floating-point move if condition is satisfied.
FMOVqr	Quad-precision floating-point move if register match condition.
FABSq	Quad-precision floating-point absolute value.
FADDq	Quad-precision floating-point addition.
FDIVq	Quad-precision floating-point division.
FdMULq	Double- to quad-precision floating-point multiply.
FMULq	Quad-precision floating-point multiply.
FNEGq	Quad-precision floating-point negation.
FSQRTq	Quad-precision floating-point square root.
FSUBq	Quad-precision floating-point subtraction.

11.3.4 Floating-Point Upper and Lower Dirty Bits in FPRS Register

The FPRS_dirty_upper (du) and FPRS_dirty_lower (dl) bits in the Floating-Point Registers State (FPRS) register are set when an instruction that modifies the corresponding upper or lower half of the floating-point register file is issued. Floating-point register file modifying instructions include floating-point operate, graphics, floating-point loads and block load instructions.

While SPARC V9 allows FPRS.du and FPRS.dl to be set pessimistically, OpenSPARC T2 only sets FPRS.du or FPRS.dl when an instruction that updates the floating-point register file successfully completes. This implies that floating-point instructions that do not update a floating-point register (for example, an FMOVcc that does not meet the condition or a floating-point operate instruction that takes a trap) leave FPRS.du and FPRS.dl unchanged.

11.3.5 Floating-Point Status Register (FSR) (Impdep #13, 19, 22, 23, 24)

OpenSPARC T2 supports precise-traps and implements all three exception fields (tem, cexc, and aexc) conforming to IEEE Standard 754-1985.

OpenSPARC T2 implements the FSR register according to the definition in UltraSPARC Architecture 2007, with the following implementation-specific clarifications:

- OpenSPARC T2 does not contain an FQ, therefore FSR.qne always reads as 0 and an attempt to read the FQ with an RDPR instruction causes an *illegal_instruction* trap.
- OpenSPARC T2 does not detect the unimplemented_FPop, sequence_error, hardware_error or invalid_fp_register floating-point trap types directly in hardware, therefore does not generate a trap when those conditions occur.

11.4 SPARC V9 Memory-Related Operations

11.4.1 Load/Store Alternate Address Space (Impdep #5, 29, 30)

Supported ASI accesses are listed in *Alternate Address Spaces* on page 65.

11.4.2 Read/Write ASR (Impdep #6, 7, 8, 9, 47, 48)

Supported ASRs are listed in Chapter 3, *Registers*.

11.4.3 MMU Implementation (Impdep #41)

OpenSPARC T2 memory management is based on Hardware Tablewalk-managed (or software-managed if Hardware Tablewalk is disabled) instruction and data Translation Lookaside Buffers (TLBs) and in-memory Translation Storage Buffers (TSBs) backed by a Software Translation Table. See Chapter 12, *Memory Management Unit* for more details.

11.4.4 FLUSH and Self-Modifying Code (Impdep #122)

FLUSH is needed to synchronize code and data spaces after code space is modified during program execution. FLUSH is described in *Memory Synchronization: MEMBAR and FLUSH* on page 513. On OpenSPARC T2, the FLUSH effective address is ignored, and as a result, FLUSH cannot cause a *DAE_invalid_ASI* or a *data_access_MMU_miss* trap.

Note | SPARC V9 specifies that the FLUSH instruction has no latency on the issuing virtual processor. In other words, a store to instruction space prior to the FLUSH instruction is visible immediately after the completion of FLUSH. When a flush is performed, OpenSPARC T2 guarantees that earlier code modifications will be visible across the whole system.

11.4.5 PREFETCH{A} (Impdep #103, 117)

For OpenSPARC T2, PREFETCH{A} instructions follow TABLE 11-2 based on the fcn value. All prefetches in OpenSPARC T2 are of the "weak" variety (that is, on an MMU miss, the prefetch is dropped) so the only trap generated by prefetch is *illegal_instruction* (for fcn = 5_{16}–F_{16}).

TABLE 11-2 PREFETCH{A} Variants in OpenSPARC T2

fcn	Prefetch Function	Action
0_{16}	Weak prefetch for several reads	Weak prefetch into Level 2 cache.
1_{16}	Weak prefetch for one read	
2_{16}	Weak prefetch for several writes	
3_{16}	Weak prefetch for one write	

TABLE 11-2 PREFETCH{A} Variants in OpenSPARC T2

fcn	Prefetch Function	Action
4_{16}	Prefetch Page	No operation.
5_{16}–F_{16}	—	*Illegal_instruction* trap.
10_{16}	Invalidate read-once prefetch	Weak prefetch into Level 2 cache.
11_{16}	Prefetch for read to nearest unified cache	Weak prefetch into Level 2 cache.
12_{16}–13_{16}	Strong prefetches	Weak prefetch into Level 2 cache.
14_{16}	Strong prefetch for several reads	Weak prefetch into Level 2 cache.
15_{16}	Strong prefetch for one read	
16_{16}	Strong prefetch for several writes	
17_{16}	Strong prefetch for one write	
18_{16}	Invalidate cache entry	No operation for PREFETCHA. For Prefetch, if executed in user or privileged mode, no operation. If executed while hyperprivileged, invalidate cache line from Level 2 cache (writing back to memory if dirty) leaving Level 2 cache line invalid.
19_{16}–$1F_{16}$	—	No operation

11.4.6 LDD/STD Handling (Impdep #107, 108)

LDD and STD instructions are directly executed in hardware.

Note | LDD/STD are deprecated in SPARC V9. In OpenSPARC T2 it is more efficient to use LDX/STX for accessing 64-bit data. LDD/STD take longer to execute than two 32- or 64-bit loads/stores.

11.4.7 FP mem_address_not_aligned (Impdep #109, 110, 111, 112)

LDDF{A}/STDF{A} cause an *LDDF_/STDF_ mem_address_not_aligned* trap if the effective address is 32-bit aligned but not 64-bit (doubleword) aligned.

LDQF{A}/STQF{A} are not directly executed in hardware; they cause an *illegal_instruction* trap.

11.4.8 Supported Memory Models (Impdep #113, 121)

OpenSPARC T2 supports only the TSO memory model, although certain specific operations such as block loads and stores operate under the RMO memory model. See Chapter 8, Section 8.2. Supported Memory Models.".

11.4.9 I/O Operations (Impdep #118, 123)

I/O spaces and their accesses are specified in *I/O Address Spaces* on page 64.

11.4.10 Implicit ASI When TL > 0 (Impdep #124)

OpenSPARC T2 matches all UltraSPARC Architecture implementations and makes the implicit ASI for instruction fetching `ASI_NUCLEUS` when TL > 0, while the implicit ASI for loads and stores when TL > 0 is `ASI_NUCLEUS` if PSTATE.cle=0 or `ASI_NUCLEUS_LITTLE` if PSTATE.cle=1.

11.5 Non-SPARC V9 Extensions

11.5.1 Cache Subsystem

OpenSPARC T2 contains one or more levels of cache. The cache subsystem architecture is described in Appendix D, *Caches and Cache Coherency*.

11.5.2 Memory Management Unit

OpenSPARC T2 implements a multi-level memory management scheme. The MMU architecture is described in Chapter 12, *Memory Management Unit*.

11.5.3 Error Handling

OpenSPARC T2 implements a set of programmer-visible error and exception registers. These registers and their usage are described in Chapter 25, *Error Handling*.

11.5.4 Block Memory Operations

OpenSPARC T2 supports 64-byte block memory operations utilizing a block of eight double-precision floating point registers as a temporary buffer. See *Block Load and Store Instructions* on page 31.

11.5.5 Partial Stores

OpenSPARC T2 supports 8-/16-/32-bit partial stores to memory. See *Block Load and Store Instructions* on page 31.

11.5.6 Short Floating-Point Loads and Stores

OpenSPARC T2 supports 8-/16-bit loads and stores to the floating-point registers.

11.5.7 Load Twin Extended Word

OpenSPARC T2 supports 128-bit atomic load operations to a pair of integer registers. See *Load Twin Extended Word* on page 36.

11.5.8 Interrupt Vector Handling

CPUs and I/O devices can interrupt a selected virtual processor by assembling and sending an interrupt packet. This allows hardware interrupts and cross-calls to have the same hardware mechanism and to share a common software interface for processing. Interrupt vectors are described in Chapter 7, *Interrupt Handling*.

11.5.9 Power-Down Support

OpenSPARC T2 supports the ability to power down virtual processors and I/O devices to reduce power requirements during idle periods.

11.5.10 OpenSPARC T2 Instruction Set Extensions (Impdep #106)

The OpenSPARC T2 processor supports VIS 2.0. VIS instructions are designed to enhance graphics functionality and improve the efficiency of memory accesses.

Unimplemented IMPDEP1 and IMPDEP2 opcodes encountered during execution cause an *illegal_instruction* trap.

11.5.11 Performance Instrumentation

OpenSPARC T2 performance instrumentation is described in Chapter 10, *Performance Instrumentation*.

11.5.12 Debug and Diagnostics Support

OpenSPARC T2 support for debug and diagnostics is described in Chapter 28, *Configuration and Diagnostics Support*.

CHAPTER 12

Memory Management Unit

This chapter provides detailed information about the OpenSPARC T2 Memory Management Unit. It describes the internal architecture of the MMU and how to program it.

12.1 Translation Table Entry (TTE)

The Translation Table Entry holds information for a single page mapping. The TTE is broken 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.

. TABLE 12-1 shows the sun4v TTE tag format.

TABLE 12-1 TTE Tag Format

Bit	Field	Description
63:61	—	*Reserved*
60:48	context	The 13-bit context identifier associated with the TTE.
47:42	—	*Reserved*
41:0	va	Virtual Address Tag{63:22}. The virtual page number. Bits 21 through 13 are not maintained in the tag, since these bits are used to index the smallest TSB (512 entries). **NOTE:** Hardware only supports a 48-bit VA.

The sun4v TTE data format is shown in TABLE 12-2.

TABLE 12-2 TTE Data Format

Bit	Field	Description
63	v	Valid. If the Valid bit is set, the remaining fields of the TTE are meaningful.
62	nfo	No-fault-only. If this bit is set, loads with ASI_PRIMARY_NO_FAULT{_LITTLE}, ASI_SECONDARY_NO_FAULT{_LITTLE} are translated. Any other DMMU access will trap with a *DAE_nfo_page* trap. For the IMMU, if the nfo bit is set, an *iae_nfo_page* trap will be taken.
61:56	soft2	soft2 and soft are software-defined fields, provided for use by the operating system. Software fields are not implemented in the OpenSPARC T2 TLB. soft and soft2 fields may be written with any value; they read from the TLB as zero, with the exception of soft{61}, which contains the TLB data parity bit.
55:13	ra	The real page[1] number. For OpenSPARC T2, a 40-bit real address range is supported by the hardware tablewalker, and bits {55:40} should always be zero. **NOTE:** OpenSPARC T2 TLBs store physical addresses, not real addresses. Hyperprivileged code is responsible for translation between real and physical addresses. The OpenSPARC T2 TLBs store PA{39:13}.
12	ie	Invert endianess. If this bit is set, accesses to the associated page are processed with inverse endianness from what is specified by the instruction (big-for-little and little-for-big). See Section 12.6 on page 120 for details. For the IMMU, the ie bit in the TTE is written into the ITLB but ignored during ITLB operation. The value of the ie bit written into the ITLB will be read out on an ITLB Data Access read. **Note:** This bit is intended to be set primarily for noncacheable accesses.
11	e	Side effect. If this bit is set, noncacheable memory accesses other than block loads and stores are strongly ordered against other e bit accesses, and noncacheable stores are not merged. This bit should be set for pages that map I/O devices having side effects. Note, however, that the e bit does not prevent normal instruction prefetching. For the IMMU, the e bit in the TTE is written into the ITLB, but ignored during ITLB operation. The value of the e bit written into the ITLB will be read out on an ITLB Data Access read. **NOTE:** The e bit does not force an uncacheable access. It is expected, but not required, that the cp and cv bits will be set to zero when the e bit is set.
10:9	cp, cv	The cacheable-in-physically-indexed-cache and cacheable-in-virtually-indexed-cache (cp, cv) bits determine the placement of data in OpenSPARC T2 caches, according to TABLE 12-3. The MMU does not operate on the cacheable bits, but merely passes them through to the cache subsystem. The cv bit is ignored by OpenSPARC T2, and is not written into the TLBs and returns zero on a Data Access read.

TABLE 12-3 Cacheable Field Encoding (from TSB)

Cacheable (cp:cv)	Meaning of TTE When Placed in: iTLB (I-cache PA-Indexed)	Meaning of TTE When Placed in: dTLB (D-cache PA-Indexed)
0x	Cacheable L2 cache only	Cacheable L2 cache only
1x	Cacheable L2 cache, I-cache	Cacheable L2 cache, D-cache

TABLE 12-2 TTE Data Format *(Continued)*

Bit	Field	Description
8	p	Privileged. If the p bit is set, only privileged software can access the page mapped by the TTE. If the p bit is set and an access to the page is attempted when PSTATE.priv = 0, the MMU will signal an *IAE_privilege_violation* or *DAE_privilege_violation* trap.
7	ep	Executable. If the ep bit is set, the page mapped by this TTE has execute permission granted. Otherwise, execute permission is not granted and the hardware table-walker will not load the ITLB with a TTE with ep = 0. For the IMMU and DMMU, the ep bit in the TTE is not written into the TLB, and returns zero on a Data Access read.
6	w	Writable. If the w bit is set, the page mapped by this TTE has write permission granted. Otherwise, write permission is not granted and the MMU will cause a *fast_data_access_protection* trap if a write is attempted. For the IMMU, the w bit in the TTE is written into the ITLB, but ignored during ITLB operation. The value of the w bit written into the ITLB will be read out on an ITLB Data Access read.
5:4	soft	(see soft2, above)
3:0	size	The page size of this entry, encoded as shown in TABLE 12-4.

TABLE 12-4 Size Field Encoding (from TTE)

Size{2:0}	Page Size
0000	8 KB
0001	64 KB
0010	Reserved
0011	4 MB
0100	Reserved
0101	256 MB
0110-1111	Reserved

1. sun4v supports translation from virtual addresses (VA) to real addresses (RA) to physical addresses (PA). Privileged code manages the VA-to-RA translations, while hyperprivileged code manages the RA-to-PA translations. The TLBs contain VA-to-PA translations or RA-to-PA translations (the latter are distinguished from the former by a Real bit in the TLB).

12.2 Translation Storage Buffer (TSB)

A TSB is an array of TTEs managed entirely by software. It serves as a cache of the Software Translation table, used to quickly reload the TLB in the event of a TLB miss. The discussion in this section assumes the use of the hardware support for TSB access described in Section 12.3.1, although the operating system is not required to make use of this support hardware.

Inclusion of the TLB entries in a TSB is not required; that is, translation information may exist in the TLB that is not present in the TSB.

A TSB is arranged as a direct-mapped cache of TTEs. The OpenSPARC T2 MMU provides hardware tablewalk support and precomputed pointers into the TSB(s) for both zero and nonzero contexts for four different TSB, as specified in the following registers:

- ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_0
- ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_1
- ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_2
- ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_3
- ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_0
- ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_1
- ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_2
- ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_3

In each case, the *n* least significant bits of the respective virtual page number are used as the offset from the TSB base address, with *n* equal to log base 2 of the number of TTEs in the TSB.

Hardware TSB indexing support is provided for TTEs in the following registers:

- ASI_MMU_ITSB_PTR_0
- ASI_MMU_ITSB_PTR_1
- ASI_MMU_ITSB_PTR_2
- ASI_MMU_ITSB_PTR_3
- ASI_MMU_DTSB_PTR_0, ASI_MMU_DTSB_PTR_1
- ASI_MMU_DTSB_PTR_2
- ASI_MMU_DTSB_PTR_3

While the hardware tablewalk uses the TSB configuration generated by these pointers, the hardware tablewalk can be disabled and a full software implementation for TLB miss handling can be used. Under a full software implementation, simple modifications to the index pointers provided by the hardware allow formation of an M-way set-associative TSB, multiple TSBs per page size, multiple page sizes per TSB, and multiple TSBs per process.

The TSB exists as a normal data structure in memory and therefore may be cached. Indeed, the speed of the TLB miss handler relies on the TSB accesses hitting the level-2 cache at a substantial rate. This policy may result in some conflicts with normal instruction and data accesses, but the dynamic sharing of the level-2 cache resource should provide a better overall solution than that provided by a fixed partitioning.

FIGURE 12-1 shows the TSB organization. The constant *N* is determined by the size field in the TSB register; it may range from 512 entries to 16 M entries.

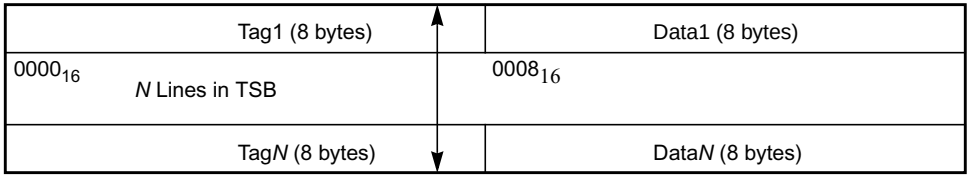


FIGURE 12-1 TSB Organization

12.3 Hardware Support for Hypervisor

To support hypervisor, a number of additions to the MMU are included.

First, a 3-bit pid (partition ID) field is included in each TLB entry to allow multiple guest OSs to share the MMU. This field is loaded with the value of the Partition Identifier register when a TLB entry is loaded. In addition, the PID entry of a TLB is compared against the Partition Identifier register to determine if a TLB hit occurs.

Second, the MMU is designed to support both virtual-to-physical and real-to-physical translations, using a single r (real translation) bit included in the TLB entry. This field is loaded with bit 10 from the VA used by the store to the I-/D-TLB Data In register or the I-/D-TLB Data Access register. The real bit distinguishes between VA → PA translations (r = 0) and RA → PA translations (r = 1). If the real bit is 1, the context ID is ignored when determining a TLB hit. TLB misses on real to physical translations generate a *data_real_translation_miss* or *inst_real_translation_miss* trap instead of the *fast_data_access_MMU_miss* and *fast_instruction_access_MMU_miss* traps respectively.

Finally, the translation operation performed depends on the state of HPSTATE.hpriv, PSTATE.priv, the MMU enables, and PSTATE.red (for IMMU), as described in *Translation* on page 122.

When the MMU is bypassed, TABLE 12-5 specifies the default physical page attribute bits. When bypassed, all LDXA and STXA operations to internal registers are correctly performed, and traps based on the page attribute bits are signaled just as if the MMU were not bypassed.

TABLE 12-5 Default Physical Page Attribute Bits

Physical Address{39}	Physical Page Attribute Bits							
	cp	ie	cv	e	p	ep	w	nfo
0	1	0	0	0	0	1	1	0
1	0	0	0	1	0	1	1	0

12.3.1 Hardware Support for TSB Access

The MMU hardware provides services to allow the TLB miss handler to efficiently reload a missing TLB entry. These services include:

- Hardware reload of missing TTE entry (hardware tablewalk).
- Formation of TSB Pointers based on the missing virtual address.
- 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.

12.3.1.1 Hardware Tablewalk

Hardware Tablewalk is a hardware state machine that services reload requests from the TLBs. It accesses the TSBs to find TTEs that match the VA and one of the contexts of the request. Hardware Tablewalk can access up to four separate TSBs for each request.

Note | If any of a strand's TSB Config Registers has the Enable bit sit, hardware tablewalk is considered to be enabled for the strand.

Hardware Tablewalk also provides a real page number (RPN) to physical page number (PPN) translation mechanism. The supervisor controls the TTE, but the supervisor cannot access or control physical memory, so its TTEs contain RPNs, not PPNs The hypervisor programs the RPN-to-PPN translation within Hardware Tablewalk to permit Hardware Tablewalk to load supervisor-controlled TTEs into the TLBs that can translate VAs into PAs.

Hardware Tablewalk does not translate real requests. In the event that a Real Address misses the TLB, the TLU signals a *inst_real_translation_miss* or *data_real_translation_miss* trap, and software loads the TLB as described in *Software TLB Reload* on page 108.

Hardware Tablewalk is stranded and pipelined; up to four TSB accesses for each of the eight strands can be in the pending at one time. The basic dataflow is pipelined, so that a single instance of the dataflow supports all eight strands.

A typical TLB miss and refill sequence when hardware tablewalk is enabled is as follows:

1. Hardware Tablewalk uses the TSB Configuration registers and the VA of the access to calculate the address of the TTE to examine. The TSB Configuration register provides the base address of the TSB as well as the number of TTEs in the TSB and the size of the pages translated by the TTEs.[1] Hardware Tablewalk uses a Nonzero Context TSB Configuration register if the context of the request is nonzero; otherwise, it uses a Zero Context TSB Configuration register. The context of the request is assumed to be the content of Context register 0 (in the event of a TLB miss on a Primary or Secondary Context access). Hardware Tablewalk uses the page size from the TSB Configuration register to calculate the presumed VPN for the given VA.[2] The VPN is generated with VA{63:48} sign-extended from VA{47} to allow TTE entries pointing into the VA hole to mismatch in the hardware tablewalk VPN comparison. Hardware Tablewalk then uses the number of TTE entries and the presumed VPN to generate an index into the TSB. This index is concatenated with the upper bits of the base address to generate the TTE address, using the formula specified in *MMU I-/D-TSB Pointer Registers* on page 140.

2. Hardware Tablewalk forwards a quadword load request for the TTE address to the L2 cache. At some later time, the L2 returns the TTE to Hardware Tablewalk.

3. Hardware Tablewalk compares the VPN (masked using the page size of the TTE) and context of the request and the page size from the configuration register to that from the TTE, and also examines the v bit, and for an ITLB miss, the ep bit of the TTE. If the v bit is set, reserved fields in the TTE Tag ({63:61} and {47:42}) are zero, the page size of the TTE is a supported page size that is not smaller than the page size of the configuration register, and the VPN and context match (and for an ITLB miss, the TTE ep bit is 1), Hardware Tablewalk forwards the TTE to the TLB with the RPN translated into a PPN (see *Real Page Number To Physical Page Number Translation* below). For an ITLB miss, if the v bit is set, reserved fields in the TTE Tag are zero, the page size of the TTE is supported and not smaller than the page size of the configuration register, and the VPN and context match, but the ep bit of the TTE is 0, an *IAE_unauth_access* trap is generated. If the v bit is clear, reserved fields in the TTE Tag are not all zero, the page size of the TTE is unsupported or smaller than the page size of the configuration, or the VPN or context do not match, Hardware Tablewalk waits for the rest of the enabled TSBs to return TTEs; Hardware Tablewalk supports four TSBs per strand for zero contexts and four for nonzero contexts. In some configurations, Hardware Tablewalk ignores the context match; see *Multiple Contexts* below.

1. Hardware tablewalk will only be able to refill the TLB when the desired TTE in the TSB is the same size or larger than that specified in the TSB Configuration register.

2. If pages of size larger than that specified in the TSB Configuration register are also cached in the TSB, this implies that the TTE entry for the larger pages must be replicated in the TSB (e.g., a 64-Kbyte page in a TSB configured for 8-Kbyte pages must occupy eight consecutive TSB entries).

4. If none of the TTE entries from the four TSBs meet the v bit, reserved TTE Tag fields, page size, and matching VPN and context requirements, hardware generates an *instruction_access_MMU_miss* or *data_access_MMU_miss* trap.

Multiple Contexts. Multiple primary and secondary contexts permit different processes to share TTEs within the TLBs. The use_context_0 and use_context_1 bits in the TSB Configuration register disable the context match for Hardware Tablewalk. Hardware Tablewalk ignores the contexts in the TSB TTEs if either of these bits is active for requests with nonzero contexts. If either bit is 1 and the TTE v bit is set, reserved fields in the TTE Tag are zero, the page size is supported by OpenSPARC T2 and not smaller than the page size in the configuration register, and the VPN matches, Hardware Tablewalk signals the TLB to write either context 0 or context 1 (depending on which bit is set) as the context of the TTE when it is loaded (instead of the context in the TTE itself). Hardware Tablewalk ignores these bits for requests with a zero (nucleus) context value.

Real Page Number To Physical Page Number Translation. When Hardware Tablewalk fetches a TTE from a TSB, it can treat the ra field as either an RA or a PA under control of the ra_not_pa field of the TSB config register. If the ra_not_pa bit is set, the hardware tablewalker will translate the Real Page Number in the TTE into a Physical Page Number. The TLBs store this physical page number. The TLBs use this PPN to translate VAs into PAs. The hypervisor controls the RPN to PPN translation mechanism.

The RPN-to-PPN translation mechanism provides both range checking as well as mapping of address ranges from one location to another. The first check is that the RPN does not contain a non-zero bits 55:40. If RPN{55:40} is nonzero, then an *instruction_invalid_TSB_entry* or *data_invalid_TSB_entry* trap is generated to the strand that initiated the Hardware Tablewalk. Otherwise, the translation mechanism uses the RPN and page size in the TTE and calculates the starting and ending addresses for the specified real page. It then checks that these addresses lie in one of four ranges specified by the Real Range registers. If the real page lies completely inside one of the ranges (and the range is enabled), then the translation mechanism adds the RPN in the TTE to the corresponding field in the Physical Offset register to create the Physical Page Number.[1] If the real page does not lie completely within either range, then an *instruction_invalid_TSB_entry* or *data_invalid_TSB_entry* trap is generated to strand that initiated the Hardware Tablewalk. Each strand has four dedicated ranges with corresponding physical offsets. The RPN-to-PPN translation does not depend on the context value being zero or nonzero.

Note | When the TSB Config register has ra_not_pa = 0, no range checking is provided for PPNs.

1. Strictly speaking, this is not a real page number but a real address, as the bits below the page size boundary in the RPN may not be zeroed. They are zeroed when written into the TLB so the value that is written into the TLB is a true RPN.

The following is pseudocode for the hardware tablewalk sequence (translation miss address is in VA, miss context in `ctxt`, instruction miss is in `inst_translate`, context type selection is in primary):

```
tsb_config = (ctxt) ? TSB_NONZERO_CONFIG[i] : TSB_ZERO_CONFIG[i];
for (i = 0; i < 4, i++) {
  vpn = generate_vpn(VA, tsb_config.page_size);
  ignore_context = (ctxt == 0) || tsb_config.use_context_0 ||
    tsb_config.use_context_1;
  tte_ptr = generate_tte_ptr(VA, tsb_config);
  tte = *tte_ptr;
  tte_vpn = generate_tte_vpn(tte.va, tsb_config.page_size)
  if ((tte.v == 1) && (tte.rsvd0 == 0) && (tte.rsvd1 == 0) &&
       ((tte.size < 2) || (tte.size == 3) || (tte.size == 5)) &&
       (tte.size >= tsb_config.page_size) &&
       (ignore_context || (tte.context == ctxt)) &&
       (tte_vpn == vpn)) {
    if (inst_translate && (tte.ep == 0) raise IAE_UNAUTH_ACCESS;
    break;
  }
}
if (i == 4) {
  // no matching TTE found
  raise (inst_translate) ?
    INST_ACCESS_MMU_MISS : DATA_ACCESS_MMU_MISS;
}
if (ctxt == 0) {
  tte.context = 0;
} else if (tsb_config.use_context_0) {
  tte.context = (primary_context) ? ASI_PRIMARY_CONTEXT_0 :
    ASI_SECONDARY_CONTEXT_0;
} else if (tsb_config.use_context_1) {
  tte.context = (primary_context) ? ASI_PRIMARY_CONTEXT_1 :
    ASI_SECONDARY_CONTEXT_1;
}
if (tsb_config.RA_not_PA == 0) {
  load_tlb(tte);
} else {
  if (tte.ra && RA_55_40_MASK) {
    raise (inst_translate) ?
      INST_INVALID_TSB_ENTRY : DATA_INVALID_TSB_ENTRY;
  }
  mask = (1 << (tte.size*3)) - 1;
  rpn_low = tte.ra & ~mask;
  rpn_high = tte.ra | mask;
  for (i = 0; i < 4; i++ ) {
    if ((rpn_low >= MMU_REAL_RANGE[i].rpn_low) &&
        (rpn_high <= MMU_REAL_RANGE[i].rpn_high)) {
      tte.ra += MMU_PHYSICAL_OFFSET[i].ppn;
```

```
      load_tlb(tte);
      break;
    }
  }
  if (i == 4) {
    // ra does not lie entirely inside any real range
    raise (inst_translate) ?
      INST_INVALID_TSB_ENTRY : DATA_INVALID_TSB_ENTRY;
  }
}
```

12.3.1.2 Software TLB Reload

TLB misses can be either for virtual-to-physical translations or for real-to-physical translations.

Virtual to Physical Address Translation. For a virtual address, a typical TLB miss and refill sequence when hardware tablewalk is disabled is as follows:

1. A TLB miss causes either a *fast_instruction_access_MMU_miss* or 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 alternate space.
3. Using this information, the TLB miss handler checks to see if the desired TTE exists in the TSB. If so, the TTE data is stored into the TLB Data In register (with the Real bit in the virtual address clear) to initiate an atomic write of the TLB entry chosen by the replacement algorithm.
4. If the TTE does not exist in the TSB, the TLB miss handler jumps to a more sophisticated (and slower) TSB miss handler.

The virtual address used in the formation of the pointer addresses comes from the Tag Access register (described in *I-/D-TLB Tag Access Registers* on page 134), which holds the virtual address oF the load or store responsible for the MMU exception. (Note that there are no separate physical registers in OpenSPARC T2 hardware for the Pointer registers, but rather they are implemented through a dynamic re-ordering of the data stored in the Tag Access and the TSB registers.)

Pointers are provided by hardware in the TSB Pointer registers for all four TSBs. These pointers give the physical addresses where the TTEs for that VA and context combination would be stored if it is present in the TSB.

The TSB Tag Target register (described in *I-/D-TSB Tag Target Registers* on page 132) is formed by aligning the missing access VA (from the Tag Access register) and the current context to positions found in the description of the TTE tag. This allows an XOR instruction for TSB hit detection.

Real-to-Physical Address Translation. For a real address, a typical TLB miss and refill sequence when hardware tablewalk is disabled is as follows:

1. A TLB miss causes either a *instruction_real_translation_miss* or a *data_real_translation_miss* exception.
2. The appropriate real miss handler determines whether and how to create a real to physical translation.
3. If a real-to-physical translation is created it is inserted into the TLB with the r bit set and the instruction is retried.

The real address used in the software formation of the pointer addresses comes from the Tag Access register (described in Section 12.10.5), which holds the real address and context of the load or store responsible for the MMU exception.

No pointers are provided by hardware for real to physical translations.

The TSB Tag Target register (described in Section 12.10.3) is formed by aligning the missing access RA (from the Tag Access register) and the current context to positions found in the description of the TTE tag. This allows an XOR instruction for TSB hit detection.

12.3.2 Real-to-Physical Address Mapping and Speculative Instruction Fetch

OpenSPARC T2 speculatively fetches instructions. Under certain conditions, this can cause the memory controller to receive an unsupported physical address. Consider the following instruction sequence which exits hyperprivileged mode and returns to user or privileged mode (executed with HPSTATE.hpriv initially set to 1):

```
jmpl %g3 + %g0, %g0
wrhpr   %g0, %g0, %hpstate
```

This will cause the IMMU to go from bypass (during which VA{39:0} is passed directly to PA{39:0}) into either RA → PA or VA → PA translation. However, since the fetch of the target of the jmpl is fetched speculatively, the memory controller may see VA{39:0} of the target of the jmpl as a physical address. This address may not be supported, in which case a disrupting *software_recoverable_error* trap could result, even though no real error has occurred.

To avoid this disrupting trap, hypervisor should avoid changing translation in the delay slot of delayed control transfer instructions. For example, the sequence above could be replaced with the following code:

```
mov    %tl, %g5
add    %g5, 1, %g5
mov    %g5, %tl
```

```
mov    %g3, %tnpc
mov    0, %htstate
done
```

Although the example refers to changes in HPSTATE, any instruction that can potentially change translation should avoid being placed in the delay slot of delayed control transfer instructions. These include writes to PSTATE, I-/D-TLB Data-In/ Data-Access registers, I-/D-MMU Demap registers, and the ASI_LSU_CONTROL_REG register.

12.4 MMU-Related Faults and Traps

TABLE 12-6 lists the traps recorded by the MMU.

TABLE 12-6 MMU Traps

		Register Update		
Trap Name	**Trap Cause**	**I-Tag Access**	**D-SFAR**	**D-Tag Access**
fast_instruction_access_MMU_miss	iTLB miss with hardware tablewalk disabled	x		
instruction_access_MMU_miss	iTLB miss with hardware tablewalk enabled	x		
instruction_real_translation_miss	iTLB miss	x		
instruction_invalid_TSB_entry	RA out of range on iTLB hardware tablewalk reload	x		
IAE_privilege_violation	Privilege violation	x		
IAE_unauth_access	Hardware tablewalk attempts to load IMMU with page with ep clear	x		
IAE_NFO_page	Instruction fetch from nfo page	x		
instruction_address_range	Fetch address out of range	x		
instruction_real_range	Fetch address out of range	x		
fast_data_access_MMU_miss	dTLB miss with hardware tablewalk disabled			x
data_access_MMU_miss	dTLB miss with hardware tablewalk enabled			x
data_invalid_TSB_entry	RA out of range on dTLB hardware tablewalk reload			x
data_real_translation_miss	dTLB miss			x
DAE_invalid_asi	Invalid ASI, size, etc.		x	

TABLE 12-6 MMU Traps *(Continued)*

Trap Name	Trap Cause	Register Update		
		I-Tag Access	D-SFAR	D-Tag Access
DAE_privilege_violation	Privilege violation		x	x
DAE_nc_page	Atomic to noncacheable		x	x
DAE_nfo_page	Access to nfo page by non no-faulting		x	x
DAE_side_effect_page	Access to page with e =1 by nonfaulting load		x	x
mem_address_range	va out of valid range		x	
mem_real_range	ra out of valid range		x	
fast_data_access_protection	Protection violation		x	x
privileged_action	Use of privileged ASI			
pa_watchpoint, va_watchpoint	Data watchpoint hit		x	
instruction_va_watchpoint	Instruction watchpoint hit			
**_mem_address_not_aligned*	Misaligned memory op		x	
unsupported_page_size	TLB or TSB register loaded with illegal page size			

Note | The *fast_data_access_protection* trap is generated instead of the *data_access_protection* trap.

12.4.1 *fast_instruction_access_MMU_miss* Trap

This trap occurs when Hardware Tablewalk is disabled and the I-MMU is unable to find a translation for an instruction access that is executing using a virtual address.

Real-to-physical translations (for example, when LSU_CONTROL.im = 0) that miss in the MMU generate an *instruction_real_translation_miss* trap instead.

12.4.2 *instruction_access_MMU_miss* Trap

This trap occurs when Hardware Tablewalk is enabled and the I-MMU is unable to find a translation for an instruction access that is executing using a virtual address.

Implementation Note | This trap is taken when the appropriate TTE is not present in the iTLB with the r bit cleared and Hardware Tablewalk is unable to find the appropriate TTE in any of up to four TSBs.

Note | Real-to-physical translations (for example, when LSU_CONTROL.im=0) that miss in the MMU generate an *instruction_real_translation_miss* trap instead.

12.4.3 *instruction_real_translation_miss* Trap

This trap occurs when the I-MMU is unable to find a translation for an instruction access that is executing using a real address.

Note | Hardware Tablewalk does not load real to physical translations, and therefore *instruction_real_translation_miss* is taken regardless of whether Hardware Tablewalk is enabled or disabled.

Programming Note | The MMU Real Range and Physical Offset registers are used in the real-to-physical translation portion of a Hardware Tablewalk's virtual-to-physical translation and have no effect on whether an *instruction_real_translation_miss* trap is taken.

12.4.4 *instruction_invalid_TSB_entry* Trap

This trap occurs when the Hardware Tablewalk is loading the I-MMU with RA-to-PA translation enabled and is unable to complete the RA-to-PA portion of the translation due to the real address not lying completely in the range specified by any of the valid MMU Range registers. It also occurs on real to physical translations where bits 55:40 of the real address are non-zero.

12.4.5 *IAE_privilege_violation* Trap

The I-MMU detects a privilege violation for an instruction fetch; that is, an attempted access to a privileged page when PSTATE.priv = 0.

12.4.6 *IAE_unauth_access* Trap

The I-MMU detects an access to a page marked with the ep (execute privilege) bit clear during hardware tablewalk. There is no ep bit in the I-MMU, so on software loads of the I-MMU there is no hardware check of the ep bit.

12.4.7 *IAE_nfo_page* Trap

The I-MMU detects an access to a page marked with the nfo (no-fault-only) bit.

Implementation Note | The nfo bit is only checked on I-MMU translations. It is not checked on hardware tablewalk.

12.4.8 *instruction_address_range* Trap

The *instruction_address_range* occurs when the virtual address out of range and PSTATE.am = 0 (see *48-bit Virtual and Real Address Spaces* on page 62). It also occurs whenever OpenSPARC T2 is fetching from the address range 0000 7FFF FFFF $FFE0_{16}$ to FFFF 7FFF FFFF $FFFF_{16}$ inclusive, the IMMU is in VA → PA translation mode, and PSTATE.am = 0. The *instruction_address_range* exception also occurs whenever a branch target or the target of a DONE or RETRY instruction is in the range 0000 800 0000 000_{16} to FFFF 7FFF FFFF $FFFF_{16}$ inclusive, the IMMU is in VA → PA translation mode, and PSTATE.am = 0.

12.4.9 *instruction_real_range* Trap

The *instruction_real_range* trap occurs when the Real address out of range (see *48-bit Virtual and Real Address Spaces* on page 62). It also occurs whenever OpenSPARC T2 is fetching from the address range 0000 7FFF FFFF $FFE0_{16}$ to FFFF 7FFF FFFF $FFFF_{16}$ inclusive, the IMMU is in RA → PA translation mode, and PSTATE.am = 0. The *instruction_real_range* exception also occurs whenever a branch target or the target of a DONE or RETRY instruction is in the range 0000 8000 0000 0000_{16} to FFFF 7FFF FFFF $FFFF_{16}$ inclusive, the IMMU is in RA → PA translation mode, and PSTATE.am = 0.

12.4.10 *fast_data_access_MMU_miss* Trap

This trap occurs when Hardware Tablewalk is disabled and the MMU is unable to find a translation for a data access that is using a virtual-to-physical translation.

Note | Real-to-physical translations (for example, through `ASI_*REAL*`) that miss in the MMU generate a *data_real_translation_miss* trap instead.

12.4.11 *data_access_MMU_miss* Trap

This trap occurs when Hardware Tablewalk is enabled and the MMU is unable to find a translation for a data access that is using a virtual-to-physical translation.

Note | Real-to-physical translations (for example, through `ASI_*REAL*`) that miss in the MMU generate a *data_real_translation_miss* trap instead.

12.4.12 *data_invalid_TSB_entry* Trap

This trap occurs when Hardware Tablewalk is loading the D-MMU with RA-to-PA translation enabled and is unable to complete the RA-to-PA portion of the translation due to the real address not lying completely in the range specified by any of the valid MMU Range registers. It also occurs on real-to-physical translations where bits 55:40 of the real address are non-zero.

12.4.13 *data_real_translation_miss* Trap

This trap occurs when the MMU is unable to find a translation for a data access that is using a real-to-physical translation.

Note | Hardware Tablewalk does not load real-to-physical translations, and therefore *data_real_translation_miss* is taken regardless of whether hardware tablewalk is enabled or disabled.

Programming Note | The MMU Real Range and Physical Offset registers are used in the real-to-physical translation portion of a Hardware Tablewalk's virtual-to-physical translation and have no effect on whether a *data_real_translation_miss* trap is taken.

12.4.14 *DAE_privilege_violation* Trap

The D-MMU detects a privilege violation for a data access; that is, an attempted access to a privileged page when PSTATE.priv = 0.

12.4.15 *DAE_side_effect_page* Trap

A speculative (nonfaulting) load instruction issued to a page marked with the side-effect (e) bit = 1.

12.4.16 *DAE_nc_page* Trap

An atomic instruction (including 128-bit atomic load) issued to a memory address marked uncacheable in a physical cache; that is, with cp = 0 or with PA{39} = 1.

Implementation Note For OpenSPARC T2, cp only controls cacheability in the primary cache, not the shared secondary, and thus the hardware supports the ability to complete an atomic operation for pages with the cp bit = 0 as long as the secondary cache is enabled. However, to keep OpenSPARC T2 compliant with the UltraSPARC Architecture 2006 specification, the *DAE_nc_page* trap is generated when an atomic is issued to a memory address marked with cp = 0.

12.4.17 *DAE_invalid_asi* Trap

An invalid LDA/STA ASI value, invalid virtual address, read to write-only register, or write to read-only register, but not for an attempted user access to a restricted ASI (see the *privileged_action* trap described below).

12.4.18 *DAE_nfo_page* Trap

An access with an ASI other than ASI_{PRIMARY,SECONDARY}_NO_FAULT{_LITTLE} to a page marked with the nfo (no-fault-only) bit.

12.4.19 *mem_address_range* Trap

Virtual address out of range and PSTATE.am = 0 for data access, JMPL/RETURN, or branch/CALL. See *48-bit Virtual and Real Address Spaces* on page 62.

12.4.20 *mem_real_range* Trap

Real address out of range for data access, JMPL/RETURN, or branch/CALL. See *48-bit Virtual and Real Address Spaces* on page 62.

12.4.21 *fast_data_access_protection* Trap

This trap occurs when the MMU detects a protection violation for a data access. A protection violation is defined to be an attempted store to a page without write permission.

Note Protection violations are checked for both virtual-to-physical translations and real-to-physical translations.

12.4.22 *privileged_action* Trap

This trap occurs when an access is attempted using a *restricted* ASI while in non-privileged mode (PSTATE.priv = 0).

12.4.23 *instruction_VA_watchpoint* Trap

This trap occurs when instruction virtual watchpoints are enabled and the I-MMU detects a instruction execution at the virtual address specified by the VA Instruction Watchpoint register. See *Watchpoint Support* on page 415.

Programming Note | *instruction_VA_watchpoint* is never generated when HPSTATE.red = 1 or HPSTATE.hpriv = 1. In addition, *instruction_VA_watchpoint* traps are only generated when a virtual-to-physical translation is performed. Real accesses do not generate *instruction_VA_watchpoint* traps.

12.4.24 *VA_watchpoint* Trap

This trap occurs when virtual watchpoints are enabled and the D-MMU detects a load or store to the virtual address specified by the VA Data Watchpoint register. See *Watchpoint Support* on page 415.

Programming Note | *VA_watchpoint* is never generated when HPSTATE.hpriv = 1. In addition, *VA_watchpoint* traps are only generated when a virtual-to-physical translation is performed. Real accesses (for example, through ASI_*REAL*) do not generate *VA_watchpoint* traps.

12.4.25 *PA_watchpoint* Trap

This trap occurs when physical watchpoints are enabled and the D-MMU detects a load or store to the physical address specified by the PA Data Watchpoint register. See *Watchpoint Support* on page 415.

12.4.26 **_mem_address_not_aligned* Traps

The *lddf_mem_address_not_aligned*, *stdf_mem_address_not_aligned*, and *mem_address_not_aligned* traps occur when a load, store, atomic, or JMPL/RETURN instruction with a misaligned address is executed. The LSU signals this trap, but the D-MMU records the fault information in the DSFAR.

12.4.27 *Unsupported_page_size* Trap

This trap occurs when the IMMU or DMMU is loaded with an illegal page size, or a TSB register is programmed with an illegal page size.

12.5 MMU Operation Summary

TABLE 12-9 summarizes the behavior of the D-MMU for noninternal ASIs using tabulated abbreviations. TABLE 12-10 summarizes the behavior of the I-MMU. In each case, and for all conditions, the behavior of the MMU is given by one of the abbreviations in TABLE 12-7. TABLE 12-8 lists abbreviations for ASI types.

TABLE 12-7 Abbreviations for MMU Behavior

Abbreviation	Meaning
ok	Normal translation
dmiss	fast_*data_access_MMU_miss* or data_access_MMU_miss trap
dasi	*DAE_invalid_asi* trap
dreal	*data_real_translation_miss* trap
dpriv	*DAE_privilege_violation* trap
dse	*DAE_side_effect_page* trap
dprot	*fast_data_access_protection* trap
imiss	*fast_instruction_access_MMU_miss* or *instruction_access_MMU_miss* trap
ireal	*instruction_real_translation_miss* trap
iexc	*IAE_privilege_violation* trap

TABLE 12-8 Abbreviations for ASI Types

Abbreviation	Meaning
NUC	`ASI_NUCLEUS*`
PRIM	Any ASI with PRIMARY translation, except `*NO_FAULT`
SEC	Any ASI with SECONDARY translation, except `*NO_FAULT`
PRIM_NF	`ASI_PRIMARY_NO_FAULT*`
SEC_NF	`ASI_SECONDARY_NO_FAULT*`
U_PRIM	`ASI_*_AS_IF_USER_PRIMARY*`
U_SEC	`ASI_*_AS_IF_USER_SECONDARY*`
U_PRIV	`ASI_*_AS_IF_PRIV_*`
REAL	`ASI_*REAL*`

Note | The *_LITTLE versions of the ASIs behave the same as the big-endian versions with regard to the MMU table of operations.

Other abbreviations include "w" for the writable bit, "e" for the side-effect bit, and "p" for the privileged bit.

TABLE 12-9 and TABLE 12-10 do not cover the following cases:

- Invalid ASIs, ASIs that have no meaning for the opcodes listed, or nonexistent ASIs; for example, ASI_PRIMARY_NO_FAULT for a store or atomic; also, access to OpenSPARC T2 internal registers other than LDXA, LDFA, STDFA or STXA; the MMU signals a *DAE_invalid_asi* trap for this case.
- Attempted access using a restricted ASI in nonprivileged mode; the MMU signals a *privileged_action* trap for this case. Attempted use of a hyperprivileged ASI in privileged mode; the MMU also signals *privileged_action* trap for this case.
- An atomic instruction (including 128-bit atomic load) issued to a memory address marked uncacheable in a physical cache (that is, with cp = 0 or pa{39} = 1); the MMU signals a *DAE_nc_page* trap for this case.
- A data access with an ASI other than ASI_{PRIMARY,SECONDARY}_NO_FAULT{_LITTLE} or an instruction access to a page marked with the nfo (no-fault-only) bit; the MMU signals a *DAE_nfo_page* or *IAE_nfo_page* trap for this case.
- An instruction fetch to a memory address marked non-executable (ep = 0). This is checked when Hardware Tablewalk attempts to load the I-MMU, and an *IAE_unauth_access* trap is taken instead.
- Real address out of range; the MMU signals a *mem_real_range* or *instruction_real_range* trap for this case.
- Virtual address out of range and PSTATE.am is not set; the MMU signals a *mem_address_range* or *instruction_address_range* trap for this case.

TABLE 12-9 D-MMU Operations for Normal ASIs

Condition				Behavior				
Opcode	priv Mode	ASI	w	TLB Miss	e = 0 p = 0	e = 0 p = 1	e = 1 p = 0	e = 1 p = 1
Load	non-privileged	PRIM, SEC	—	dmiss	ok	dpriv	ok	dpriv
		PRIM_NF, SEC_NF	—	dmiss	ok	dpriv	dse	dpriv
	privileged	PRIM, SEC, NUC	—	dmiss	ok			
		PRIM_NF, SEC_NF	—	dmiss	ok		dse	
		U_PRIM, U_SEC	—	dmiss	ok	dpriv	ok	dpriv
		REAL	—	dreal	ok			
	hyper-privileged	PRIM, SEC, NUC[1]	1	—	ok	—	ok	—
		PRIM_NF, SEC_NF[1]	1	—	ok	—	dse	—
		U_PRIM, U_SEC	—	dmiss	ok	dpriv	ok	dpriv
		U_PRIV	—	dmiss	ok			
		REAL	—	dreal	ok			
FLUSH	non-privileged		—	ok				
	privileged		—	ok				
	hyper-privileged		—	ok				
Store or Atomic	non-privileged	PRIM, SEC	0	dmiss	dprot	dpriv	dprot	dpriv
			1	dmiss	ok	dpriv	ok	dpriv
	privileged	PRIM, SEC, NUC	0	dmiss	dprot			
			1	dmiss	ok			
		U_PRIM, U_SEC	0	dmiss	dprot	dpriv	dprot	dpriv
			1	dmiss	ok	dpriv	ok	dpriv
		REAL	0	dreal	dprot			
			1	dreal	ok			
	hyper-privileged	PRIM, SEC, NUC[1]	1	—	ok	—	ok	—
		U_PRIM, U_SEC	0	dmiss	dprot	dpriv	dprot	dpriv
			1	dmiss	ok	dpriv	ok	dpriv
		U_PRIV	0	dmiss	dprot	ok	dprot	ok
			1	dmiss	ok			
		REAL	0	dreal	dprot			
			1	dreal	ok			

1. When hyperprivileged, these context values use the default physical page attributes from TABLE 12-5, which specify that w = 1, e = PA{39}, and p = 0.

When LSU_CONTROL_REG.dm = 0, the table above applies, but *dmiss* entries in the TLB Miss column change to *dreal*.

TABLE 12-10 I-MMU Operations

Condition	Behavior		
priv Mode	TLB Miss	P = 0	P =
nonprivileged	imiss	ok	iexc
privileged	imiss	ok	
hyperprivileged	—	ok	

When LSU_CONTROL_REG.im = 0, the table above applies, but *imiss* entries in the TLB Miss column change to *ireal*. When HPSTATE.red = 1, the ITLB is bypassed and has the same behavior as the hyperprivileged mode row in the table.

See *Alternate Address Spaces* on page 65 for a summary of the OpenSPARC T2 ASI map.

12.6 ASI Value, Context, and Endianness Selection for Translation

The MMU uses a two-step process to select the context for a translation:

1. The ASI is determined (conceptually by the Integer Unit) from the instruction, trap level, and the virtual processor endian mode
2. The context register is determined directly from the ASI.

The ASI value and endianness (little or big) are determined for the I-MMU and D-MMU respectively according to TABLE 12-11 and TABLE 12-12 on page 121.

Notes

The secondary context is never used to fetch instructions.

The endianness of a data access is specified by three conditions: the ASI specified in the opcode or ASI register, the PSTATE current little endian bit, and the D-MMU invert endianness bit.

The D-MMU invert endianness (ie) bit inverts the endianness for all accesses to translating ASIs, including LD/st/Atomic alternates that have specified an ASI. That is, LDXA [%g1]ASI_PRIMARY_LITTLE will be big-endian if the ie bit is on. Accesses to nontranslating ASIs are not affected by the D-MMU's ie bit. See *Alternate Address Spaces* on page 65 or information about nontranslating ASIs.

TABLE 12-11 ASI Mapping for Instruction Accesses

Condition for Instruction Access	Resulting Action	
PSTATE.tl	Endianness	ASI Value (in SFSR)
0	Big	ASI_PRIMARY
> 0	Big	ASI_NUCLEUS

TABLE 12-12 ASI Mapping for Data Accesses

Condition for Data Access				Access Processed with:	
Opcode	PSTATE.tl	PSTATE.cle	TTE.ie	Endianness	ASI Value (Recorded in SFSR)
LD/ST/Atomic/FLUSH (Using Default ASI)	0	0	0	Big	ASI_PRIMARY
			1	Little	
		1	0	Little	ASI_PRIMARY_LITTLE
			1	Big	
	> 0	0	0	Big	ASI_NUCLEUS
			1	Little	
		1	0	Little	ASI_NUCLEUS_LITTLE
			1	Big	
LD/st/Atomic Alternate with specified ASI *not* ending in "_LITTLE"	Don't Care	Don't Care	0	Big[1]	Specified ASI value from immediate field in opcode or ASI register
			1	Little[1]	
LD/st/Atomic Alternate with specified ASI ending in '_LITTLE"	Don't Care	Don't Care	0	Little	Specified ASI value from immediate field in opcode or ASI register
			1	Big	

[1] Accesses to nontranslating ASIs are always made in "big endian" mode, regardless of the setting of the various ie bits. See *Alternate Address Spaces* on page 65 for information about nontranslating ASIs.

The context registers used by the data and instruction MMUs is determined from TABLE 12-13. A comprehensive list of ASI values can be found in the ASI map in *Alternate Address Spaces* on page 65. The context register selection is not affected by the endianness of the access

.

TABLE 12-13 I-MMU and D-MMU Context Register Usage

ASI Value	Context Register
ASI_*NUCLEUS*[1]	Nucleus (0000_{16} hard-wired)
ASI_*PRIMARY*[2]	Primary 0 and Primary 1
ASI_*SECONDARY*[3]	Secondary 0 and Secondary 1
All other ASI values	(Not applicable, no translation)

1. Any ASI name containing the string "NUCLEUS".
2. Any ASI name containing the string "PRIMARY".

3. Any ASI name containing the string “SECONDARY”.

12.7 Translation

The translation operation of MMUs is determined by the LSU_CONTROL_REG and the HPSTATE registers.

12.7.1 Instruction Translation

TABLE 12-14 describes the operation of the I-MMU.

TABLE 12-14 IMMU Translation

Control State			
LSU.im	**HPSTATE.hpriv**	**HPSTATE.red**	**IMMU Translation**
Don’t Care	Don’t Care	1	Bypass[1]
Don’t Care	1	0	Bypass[1]
0	0	0	RA $\rightarrow$ PA[2]
1	0	0	VA $\rightarrow$ PA[3]

1. VA{39:0} passed directly to PA{39:0}
2. VA{47:0} passed directly to RA{47:0}, RA{47:0} translated via the IMMU.
3. VA{47:0} translated via the IMMU.

12.7.1.1 Instruction Prefetching

OpenSPARC T2 fetches instructions sequentially (including delay slots). OpenSPARC T2 fetches delay slots before the branch is resolved (before whether the delay slot will be annulled is known). OpenSPARC T2 also fetches the target of a DCTI before the delay slot executes. For both these cases, OpenSPARC T2 may fetch from a nonexistent PA (in the case of a fetch from memory) or from an I/O address with side effects. Hypervisor should protect against this for virtual- and real-to-physical translations by maintaining valid mappings of sequential and target addresses at all times. Hypervisor should protect against this for bypassing translations by ensuring that all sequential and target addresses are backed by memory.

Certain instructions change the translation mode (writes to HPSTATE, stores to ASI_LSU_CONTROL_REG). Both of these translation mode changes can only be made while hyperprivileged. Since ASI_LSU_CONTROL_REG.im has no effect on translation while hyperprivileged, there are no issues with the hypervisor storing to ASI_LSU_CONTROL_REG.

However, a write to HPSTATE may change both the translation and privilege level at the same time. This translation change may result in the sequential instruction or the branch target being fetched in both the old translation mode (bypass) and then refetched in the new translation mode (either real to physical or virtual to physical) when OpenSPARC T2 realizes that translation has changed. For this case, it is desirable that the hypervisor enforce that valid translations exist for fetches in both the old translation mode (bypass) and the new translation mode (real to physical or virtual to physical). If the hypervisor is only able to enforce translations for the real or virtual to physical case, it is possible that an error may be encountered on the bypassing instruction fetch. Any precise error trap associated with the error will be suppressed and not be presented to the strand; however, the error status registers in the L2 cache, memory controller, and/or IO subsystem will still record the error (and possibly generate a disrupting trap in addition). Therefore, if the hypervisor is unable to enforce translation for both the bypass and real-/virtual-to-physical translations, it must be able to handle the potential spurious error logging in the L2 cache, memory controller, and/or IO subsystem.

12.7.2 Data Translation

TABLE 12-15 describes the operation of the D-MMU.

TABLE 12-15 DMMU Translation

Control State		DMMU Translation
LSU.dm	HPSTATE.hpriv	
0	0	RA → PA[1]
0	1	Follows
1	See	Follows

1. VA{63:0} passed directly to RA{63:0}, RA{63:0} translated via the DMMU.

TABLE 12-16 DMMU Translation When LSU_CONTROL_REG.dm = 1 or HPSTATE.hpriv = 1 *(1 of 5)*

ASI Value (hex)	ASI NAME	Translation		
		Nonprivileged	Privileged	Hypervisor
00_{16}–03_{16}	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
04_{16}	ASI_NUCLEUS	*privileged_action*	VA → PA	bypass
05_{16}–$0B_{16}$	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
$0C_{16}$	ASI_NUCLEUS_LITTLE	*privileged_action*	VA → PA	bypass
$0D_{16}$–$0F_{16}$	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
10_{16}	ASI_AS_IF_USER_PRIMARY	*privileged_action*	VA → PA	VA → PA
11_{16}	ASI_AS_IF_USER_SECONDARY	*privileged_action*	VA → PA	VA → PA
12_{16}–13_{16}	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
14_{16}	ASI_REAL	*privileged_action*	RA → PA	RA → PA
15_{16}	ASI_REAL_IO	*privileged_action*	RA → PA	RA → PA
16_{16}	ASI_BLOCK_AS_IF_USER_PRIMARY	*privileged_action*	VA → PA	VA → PA
17_{16}	ASI_BLOCK_AS_IF_USER_SECONDARY	*privileged_action*	VA → PA	VA → PA
18_{16}	ASI_AS_IF_USER_PRIMARY_LITTLE	*privileged_action*	VA → PA	VA → PA
19_{16}	ASI_AS_IF_USER_SECONDARY_LITTLE	*privileged_action*	VA → PA	VA → PA
$1A_{16}$–$1B_{16}$	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
$1C_{161}$	ASI_REAL_LITTLE	*privileged_action*	RA → PA	RA → PA
$1D_{16}$	ASI_REAL_IO_LITTLE	*privileged_action*	RA → PA	RA → PA
$1E_{16}$	ASI_BLOCK_AS_IF_USER_PRIMARY_LITTLE	*privileged_action*	VA → PA	VA → PA
$1F_{16}$	ASI_BLOCK_AS_IF_USER_SECONDARY_LITTLE	*privileged_action*	VA → PA	VA → PA
20_{16}	ASI_SCRATCHPAD	*privileged_action*	nontranslating	nontranslating
21_{16}	ASI_MMU	*privileged_action*	nontranslating	nontranslating
22_{16}	ASI_TWINX_AIUP, ASI_STBI_AIUP	*privileged_action*	VA → PA	VA → PA
23_{16}	ASI_TWINX_AIUS, ASI_STBI_AIUS	*privileged_action*	VA → PA	VA → PA
24_{16}	ASI_TWINX	*privileged_action*	VA → PA	bypass
25_{16}	ASI_QUEUE	*privileged_action*	nontranslating	nontranslating
26_{16}	ASI_TWINX_REAL	*privileged_action*	RA → PA	RA → PA

TABLE 12-16 DMMU Translation When LSU_CONTROL_REG.dm = 1 or HPSTATE.hpriv = 1 *(2 of 5)*

ASI Value (hex)	ASI NAME	Translation		
		Nonprivileged	Privileged	Hypervisor
27_{16}	ASI_TWINX_NUCLEUS, ASI_STBI_N	*privileged_action*	VA → PA	bypass
28_{16}–29_{16}	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
$2A_{16}$	ASI_TWINX_AIUPL, ASI_STBI_AIUPL	*privileged_action*	VA → PA	VA → PA
$2B_{16}$	ASI_TWINX_AIUSL, ASI_STBI_AIUSL	*privileged_action*	VA → PA	VA → PA
$2C_{16}$	ASI_TWINX_LITTLE	*privileged_action*	VA → PA	bypass
$2D_{16}$	*Reserved*	*privileged_action*	*DAE_invalid_asi*	*DAE_invalid_asi*
$2E_{16}$	ASI_TWINX_REAL_LITTLE	*privileged_action*	RA → PA	RA → PA
$2F_{16}$	ASI_TWINX_NL, ASI_STBI_NL	*privileged_action*	VA → PA	bypass
30_{16}	ASI_AS_IF_PRIV_PRIMARY	*privileged_action*	*privileged_action*	VA → PA
31_{16}	ASI_AS_IF_PRIV_SECONDARY	*privileged_action*	*privileged_action*	VA → PA
32_{16}–35_{16}	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
36_{16}	ASI_AS_IF_PRIV_NUCLEUS	*privileged_action*	*privileged_action*	VA → PA
37_{16}	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
38_{16}	ASI_AS_IF_PRIV_PRIMARY_LITTLE	*privileged_action*	*privileged_action*	VA → PA
39_{16}	ASI_AS_IF_PRIV_SECONDARY_LITTLE	*privileged_action*	*privileged_action*	VA → PA
$3A_{16}$–$3D_{16}$	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
$3E_{16}$	ASI_AS_IF_PRIV_NUCLEUS_LITTLE	*privileged_action*	*privileged_action*	VA → PA
$3F_{16}$	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
40_{16}	ASI_STREAM	*privileged_action*	*privileged_action*	nontranslating
41_{16}	ASI_CMP	*privileged_action*	*privileged_action*	nontranslating
42_{16}	ASI_INST_MASK_REG/ ASI_LSU_DIAG_REG	*privileged_action*	*privileged_action*	nontranslating
43_{16}	ASI_ERROR_INJECT_REG	*privileged_action*	*privileged_action*	nontranslating
44_{16}	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
45_{16}	ASI_LSU_CONTROL_REG, ASI_DECR, ASI_RST_VEC_MASK	*privileged_action*	*privileged_action*	nontranslating
46_{16}	ASI_DCACHE_DATA	*privileged_action*	*privileged_action*	nontranslating
47_{16}	ASI_DCACHE_TAG	*privileged_action*	*privileged_action*	nontranslating
48_{16}	ASI_IRF_ECC_REG	*privileged_action*	*privileged_action*	nontranslating
49_{16}	ASI_FRF_ECC_REG	*privileged_action*	*privileged_action*	nontranslating

TABLE 12-16 DMMU Translation When LSU_CONTROL_REG.dm = 1 or HPSTATE.hpriv = 1 *(3 of 5)*

ASI Value (hex)	ASI NAME	Translation		
		Nonprivileged	Privileged	Hypervisor
$4A_{16}$	ASI_STB_ACCESS	*privileged_action*	*privileged_action*	nontranslating
$4B_{16}$	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
$4C_{16}$	ASI_DESR/ASI_DFESR/ASI_CERER/ ASI_SETER/ASI_CLESR/ASI_ CLFESR	*privileged_action*	*privileged_action*	nontranslating
$4D_{16}$	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
$4E_{16}$	ASI_SPARC_PWR_MGMT	*privileged_action*	*privileged_action*	nontranslating
$4F_{16}$	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
50_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
51_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
52_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
53_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
54_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
55_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
56_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
57_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
58_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
59_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
$5A_{16}$	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
$5B_{16}$	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
$5C_{16}$	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
$5D_{16}$	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
$5E_{16}$	ASI_IMMU	*privileged_action*	*privileged_action*	nontranslating
$5F_{16}$	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
60_{16}–62_{16}	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
63_{16}	ASI_HYP_SCRATCHPAD	*privileged_action*	*privileged_action*	nontranslating
64_{16}–65_{16}	ASI_IMMU	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
66_{16}	ASI_ICACHE_INSTR	*privileged_action*	*privileged_action*	nontranslating
67_{16}	ASI_ICACHE_TAG	*privileged_action*	*privileged_action*	nontranslating
64_{16}–71_{16}	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
72_{16}	ASI_INTR_RECEIVE	*privileged_action*	*privileged_action*	nontranslating
73_{16}	ASI_INTR_W	*privileged_action*	*privileged_action*	nontranslating
74_{16}	ASI_INTR_R	*privileged_action*	*privileged_action*	nontranslating

TABLE 12-16 DMMU Translation When LSU_CONTROL_REG.dm = 1 or HPSTATE.hpriv = 1 *(4 of 5)*

ASI Value (hex)	ASI NAME	Translation		
		Nonprivileged	Privileged	Hypervisor
75_{16}–$7F_{16}$	*Reserved*	*privileged_action*	*privileged_action*	*DAE_invalid_asi*
80_{16}	ASI_PRIMARY	VA → PA	VA → PA	bypass
81_{16}	ASI_SECONDARY	VA → PA	VA → PA	bypass
82_{16}	ASI_PRIMARY_NO_FAULT	VA → PA	VA → PA	bypass
83_{16}	ASI_SECONDARY_NO_FAULT	VA → PA	VA → PA	bypass
84_{16}–87_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
88_{16}	ASI_PRIMARY_LITTLE	VA → PA	VA → PA	bypass
89_{16}	ASI_SECONDARY_LITTLE	VA → PA	VA → PA	bypass
$8A_{16}$	ASI_PRIMARY_NO_FAULT_LITTLE	VA → PA	VA → PA	bypass
$8B_{16}$	ASI_SECONDARY_NO_FAULT_ LITTLE	VA → PA	VA → PA	bypass
$8C_{16}$–BF_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$C0_{16}$	ASI_PST8_P	VA → PA	VA → PA	bypass
$C1_{16}$	ASI_PST8_S	VA → PA	VA → PA	bypass
$C2_{16}$	ASI_PST16_P	VA → PA	VA → PA	bypass
$C3_{16}$	ASI_PST16_S	VA → PA	VA → PA	bypass
$C4_{16}$	ASI_PST32_P	VA → PA	VA → PA	bypass
$C5_{16}$	ASI_PST32_S	VA → PA	VA → PA	bypass
$C6_{16}$–$C7_{16}$	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$C8_{16}$	ASI_PST8_PL	VA → PA	VA → PA	bypass
$C9_{16}$	ASI_PST8_SL	VA → PA	VA → PA	bypass
CA_{16}	ASI_PST16_PL	VA → PA	VA → PA	bypass
CB_{16}	ASI_PST16_SL	VA → PA	VA → PA	bypass
CC_{16}	ASI_PST32_PL	VA → PA	VA → PA	bypass
CD_{16}	ASI_PST32_SL	VA → PA	VA → PA	bypass
CE_{16}–CF_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$D0_{16}$	ASI_FL8_P	VA → PA	VA → PA	bypass
$D1_{16}$	ASI_FL8_S	VA → PA	VA → PA	bypass
$D2_{16}$	ASI_FL16_P	VA → PA	VA → PA	bypass
$D3_{16}$	ASI_FL16_S	VA → PA	VA → PA	bypass
$D4_{16}$–$D7_{16}$		*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$D8_{16}$	ASI_FL8_PL	VA → PA	VA → PA	bypass

TABLE 12-16 DMMU Translation When LSU_CONTROL_REG.dm = 1 or HPSTATE.hpriv = 1 *(5 of 5)*

ASI Value (hex)	ASI NAME	Translation		
		Nonprivileged	Privileged	Hypervisor
$D9_{16}$	ASI_FL8_SL	VA → PA	VA → PA	bypass
DA_{16}	ASI_FL16_PL	VA → PA	VA → PA	bypass
DB_{16}	ASI_FL16_SL	VA → PA	VA → PA	bypass
DC_{16}–DF_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$E0_{16}$	ASI_BLK_COMMIT_PRIMARY	VA → PA	VA → PA	bypass
$E1_{16}$	ASI_BLK_COMMIT_SECONDARY	VA → PA	VA → PA	bypass
$E2_{16}$	ASI_TWINX_P, ASI_STBI_P	VA → PA	VA → PA	bypass
$E3_{16}$	ASI_TWINX_S, ASI_STBI_S	VA → PA	VA → PA	bypass
$E4_{16}$–$E9_{16}$	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
EA_{16}	ASI_TWINX_PL, ASI_STBI_PL	VA → PA	VA → PA	bypass
EB_{16}	ASI_TWINX_PL, ASI_STBI_PL	VA → PA	VA → PA	bypass
EC_{16}-EF_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$F0_{16}$	ASI_BLK_PRIMARY	VA → PA	VA → PA	bypass
$F1_{16}$	ASI_BLK_SECONDARY	VA → PA	VA → PA	bypass
$F2_{16}$-$F7_{16}$	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*
$F8_{16}$	ASI_BLK_PRIMARY_LITTLE	VA → PA	VA → PA	bypass
$F9_{16}$	ASI_BLK_SECONDARY_LITTLE	VA → PA	VA → PA	bypass
FA_{16}-FF_{16}	*Reserved*	*DAE_invalid_asi*	*DAE_invalid_asi*	*DAE_invalid_asi*

12.8 MMU Behavior During Reset and Upon Entering RED_state

MMU Reset and RED_state behavior is described in *Machine State After Reset and in RED_State* on page 164.

12.9 Compliance With the SPARC V9 Annex F

The OpenSPARC T2 MMU complies completely with the SPARC V9 MMU Requirements described in Annex F of the *The SPARC Architecture Manual, Version 9*. TABLE 12-17 shows how various protection modes can be achieved, if necessary, through the presence or absence of a translation in the I- or D-MMU. Note that this behavior requires specialized TLB miss handler code to guarantee these conditions.

TABLE 12-17 MMU Compliance With SPARC V9 Annex F Protection Mode

Condition			
TTE in D-MMU	**TTE in I-MMU**	**Writable Attribute Bit**	**Resultant Protection Mode**
Yes	No	0	Read-only
No	Yes	Don't Care	Execute-only
Yes	No	1	Read/Write
Yes	Yes	0	Read-only/Execute
Yes	Yes	1	Read/Write/Execute

12.10 MMU Internal Registers and ASI Operations

12.10.1 Accessing MMU Registers

All internal MMU registers can be accessed directly by the virtual processor through ASIs defined by OpenSPARC T2.

See Section 12.7 for details on the behavior of the MMU during all other OpenSPARC T2 ASI accesses.

Note | STXA to an MMU register *does not* require any subsequent instructions such as a MEMBAR #Sync, FLUSH, DONE, or RETRY before the register effect will be visible to load / store / atomic accesses. OpenSPARC T2 resolves all MMU register hazards via an automatic synchronization on all MMU register writes.

If the low order three bits of the VA are non-zero in an LDXA/STXA to/from these registers, a *mem_address_not_aligned* trap occurs. Writes to read-only, reads to write-only, illegal ASI values, or illegal VA for a given ASI may cause a *DAE_invalid_asi* trap.

Caution | OpenSPARC T2 does not check for out-of-range virtual addresses during an STXA to any internal register; it simply sign-extends the virtual address based on VA{47}. Software must guarantee that the VA is within range.

Writes to the TSB register, Tag Access register, and Instruction and Data Watchpoint Address registers are not checked for out-of-range VA. No matter what is written to the register, VA{63:47} will always be identical on a read.

TABLE 12-18 OpenSPARC T2 MMU Internal Registers and ASI Operations

I-MMU ASI	D-MMU ASI	VA{63:0}	Access	Register or Operation Name
21_{16}		8_{16}	Read/Write	Primary Context 0 register
—	21_{16}	10_{16}	Read/Write	Secondary Context 0 register
21_{16}		108_{16}	Read/Write	Primary Context 1 register
—	21_{16}	110_{16}	Read/Write	Secondary Context 1 register
50_{16}	58_{16}	0_{16}	Read-only	I-/D-TSB Tag Target registers
—	58_{16}	20_{16}	Read-only	D-TLB Synchronous Fault Address register
50_{16}	58_{16}	30_{16}	Read/Write	I-/D-TLB Tag Access registers
50_{16}	58_{16}	38_{16}	Read/Write	Watchpoint address
58_{16}		40_{16}	Read/Write	Hardware Tablewalk Config register
58_{16}		80_{16}	Read/Write	Partition identifier
52_{16}		108_{16}	Read/Write	MMU Real Range 0 register
52_{16}		110_{16}	Read/Write	MMU Real Range 1 register
52_{16}		118_{16}	Read/Write	MMU Real Range 2 register
52_{16}		120_{16}	Read/Write	MMU Real Range 3 register
52_{16}		208_{16}	Read/Write	MMU Physical Offset 0 register
52_{16}		210_{16}	Read/Write	MMU Physical Offset 1 register
52_{16}		218_{16}	Read/Write	MMU Physical Offset 2 register
52_{16}		220_{16}	Read/Write	MMU Physical Offset 3 register
54_{16}		10_{16}	Read/Write	MMU Context Zero TSB Config 0 register
54_{16}		18_{16}	Read/Write	MMU Context Zero TSB Config 1 register
54_{16}		20_{16}	Read/Write	MMU Context Zero TSB Config 2 register
54_{16}		28_{16}	Read/Write	MMU Context Zero TSB Config 3 register
54_{16}		30_{16}	Read/Write	MMU Context Nonzero TSB Config 0 register
54_{16}		38_{16}	Read/Write	MMU Context Nonzero TSB Config 1 register
54_{16}		40_{16}	Read/Write	MMU Context Nonzero TSB Config 2 register

TABLE 12-18 OpenSPARC T2 MMU Internal Registers and ASI Operations *(Continued)*

I-MMU ASI	D-MMU ASI	VA{63:0}	Access	Register or Operation Name
54_{16}		48_{16}	Read/Write	MMU Context Nonzero TSB Config 3 register
54_{16}		50_{16}	Read-only	MMU I-TSB Pointer 0 register
54_{16}		58_{16}	Read-only	MMU I-TSB Pointer 1 register
54_{16}		60_{16}	Read-only	MMU I-TSB Pointer 2 register
54_{16}		68_{16}	Read-only	MMU I-TSB Pointer 3 register
54_{16}		70_{16}	Read-only	MMU D-TSB Pointer 0 register
54_{16}		78_{16}	Read-only	MMU D-TSB Pointer 1 register
54_{16}		80_{16}	Read-only	MMU D-TSB Pointer 2 register
54_{16}		88_{16}	Read-only	MMU D-TSB Pointer 3 register
54_{16}		90_{16}	Read/Write	MMU Tablewalk Pending Control register
54_{16}		98_{16}	Read-only	MMU Tablewalk Pending Status register
54_{16}	$5C_{16}$	See Section 12.10.15	Write-only	I-/D-TLB Data In registers
55_{16}	$5D_{16}$	See Section 12.10.15	Read/Write	I-/D-TLB Data Access registers
56_{16}	$5E_{16}$	See Section 12.10.15	Read-only	I-/D-TLB Tag Read register
57_{16}	$5F_{16}$	See Section 12.11.1	Write-only	I-/D-MMU demap operation

12.10.2 Context Registers

OpenSPARC T2 supports a pair of primary and a pair of secondary context registers per strand, which are shared by the I- and D-MMUs. Primary Context 0 and Primary Context 1 are the primary context registers, and a TLB entry for a translating primary ASI can match the context field with either Primary Context 0 or Primary Context 1 to produce a TLB hit. Secondary Context 0 and Secondary Context 1 are the secondary context registers, and a TLB entry for a translating secondary ASI can match the context field with either Secondary Context 0 or Secondary Context 1 to produce a TLB hit.

Compatibility Note | To maintain backward compatibility with software designed for a single primary and single secondary context register, writes to Primary (Secondary) Context 0 Register also update Primary (Secondary) Context 1 Register.

The Primary Context 0 and Primary Context 1 registers are defined as shown in FIGURE 12-2, where pcontext is the context value for the primary address space.

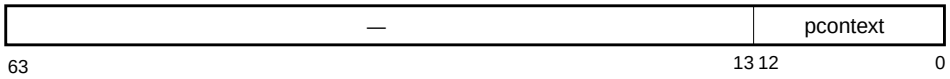


FIGURE 12-2 Primary Context 0/1 register

The Secondary Context 0 and Secondary Context 1 Registers are defined in FIGURE 12-3, where scontext is the context value for the secondary address space.

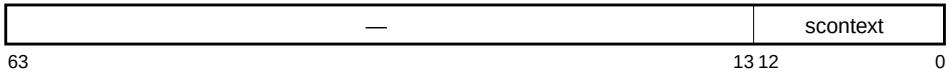


FIGURE 12-3 Secondary Context 0/1 Register

The contents of the Nucleus Context register are hardwired to the value zero:

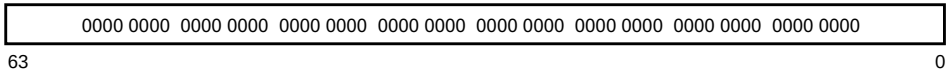


FIGURE 12-4 Nucleus Context Register

12.10.3 I-/D-TSB Tag Target Registers

The I- and D-TSB Tag Target registers are simply respective bit-shifted versions of the data stored in the I- and D-Tag Access registers. Since the I- or D-Tag Access registers are updated on I- or D-TLB misses, respectively, the I- and D-Tag Target registers appear to software to be updated on an I- or D-TLB miss. A write to this register results in a *DAE_invalid_asi* trap being taken. The registers are illustrated in FIGURE 12-5 and described in the table below the figure.

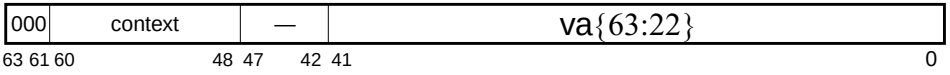


FIGURE 12-5 MMU Tag Target Registers (Two Registers)

Bit	Field	Description
60:48	context	I/D context{12:0}: The context associated with the missing virtual address. For real translations, the context value is set to zero.
41:0	va	I/D context{12:0}: The context associated with the missing virtual address. For real translations, the context value is set to zero.

Notes When PSTATE.am = 1, the upper 32 bits of the VA captured in this register will be zero.

For a 256-Mbyte page, VA{27:22} contain bits 27:22 of the virtual address and are not zeroed by hardware.

12.10.4 I-/D-MMU Synchronous Fault Address Registers (SFAR)

12.10.4.1 I-MMU Fault Address

There is no I-MMU Synchronous Fault Address register. Instead, software must read the TPC register appropriately as discussed here.

For *instruction_access_MMU_miss* traps, TPC contains the virtual address that was not found in the I-MMU TLB.

For *IAE_privilege_violation*, *IAE_unauth_access*, and *IAE_nfo_page* traps, TPC contains the virtual address of the instruction in the privileged page that caused the exception.

For *instruction_address_range* and *instruction_real_range* traps, note that the TPC in these cases contains only a 48-bit virtual (real) address, which is sign-extended based on bit VA{47} (RA{47}) for read. Thus, the TPC contains only the lower 48 bits of the virtual (real) address that is out of range.

12.10.4.2 D-MMU Fault Address

The Synchronous Fault Address register contains the virtual memory address of the access that caused the following exceptions:

- *DAE_invalid_asi*
- *DAE_privilege_violation*
- *DAE_nc_page*
- *DAE_nfo_page*
- *DAE_side_effect_page*
- *mem_address_range*
- *mem_real_range*
- *asi_data_access_protection*
- *privileged_action*
- *VA_watchpoint*
- *PA_watchpoint*
- *mem_address_not_aligned*
- *LDDF_mem_address_not_aligned*

- *STDF_mem_address_not_aligned*

This register is read-only, a write to this register results in a *DAE_invalid_asi* trap being taken.

FIGURE 12-6 illustrates the D-SFAR; the va field is described below the table.

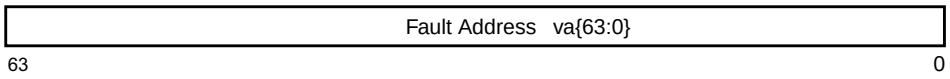


FIGURE 12-6 D-MMU Synchronous Fault Address Register (SFAR) Format

Bit	Field	Description
63:0	va	Fault Address: The virtual (real) address associated with the translation fault. This field is sign-extended based on VA{47} (RA{47}), so bits VA{63:48} (RA{63:48}) do not correspond to the virtual (real) address used in the translation for the case of a VA (RA) out-of-range *mem_address_range* (mem_real_range) trap (for this case, software must disassemble the trapping instruction).

Notes When PSTATE.am = 1, the upper 32 bits of the VA captured in this register will be zero.

The DSFAR is shared for precise error handling, and contains the error address as described in *DMMU Synchronous Fault Address Register* on page 252 following an *internal_processor_error*, *data_access_MMU_error*, or *data_access_error*.

12.10.5 I-/D-TLB Tag Access Registers

In each MMU the Tag Access register is used as a temporary buffer for writing the TLB Entry tag information. The Tag Access register may be updated during any of the following operations:

1. When the MMU signals a trap due to a miss, exception, or protection. The MMU hardware automatically writes the missing VA and the appropriate context (`ASI_PRIMARY_CONTEXT_0` for primary context accesses, `ASI_SECONDARY_CONTEXT_0` for secondary context accesses, `ASI_NUCLEUS_CONTEXT` for other accesses) into the Tag Access register to facilitate formation of the TSB Tag Target register. See TABLE 12-6 for the Tag Access register update policy.

2. An ASI write to the Tag Access register. Before an ASI store to the TLB Data Access registers, the operating system must set the Tag Access register to the values desired in the TLB Entry. Note that an ASI store to the TLB Data-In register for automatic replacement also uses the Tag Access register, but typically the value written into the Tag Access register by the MMU hardware is appropriate.
3. An I-/D-MMU demap operation. For an I-/D-MMU demap operation, the corresponding Tag Access register va field is loaded with the matching VA bits from the demap store address. If the Context ID field of the demap store address (see Section 12.11.1) is 00, then the context field of the Tag Access register is loaded with the context of `ASI_PRIMARY_CONTEXT_0`. Otherwise, the context field of the Tag Access register is loaded with all zeros.
4. An I-/D-TLB load by the Hardware Tablewalker.

Note | Any update to the Tag Access registers immediately affects the data that is returned from subsequent reads of the Tag Target and TSB Pointer registers.

Compatibility Note | The updating of Tag Access on a demap operation and hardware tablewalk is specific to OpenSPARC T2. No previous UltraSPARC processors updated Tag Access on demap, and no previous UltraSPARC processors supported Hardware Tablewalk.

The TLB Tag Access registers are defined in FIGURE 12-7; register bits are described in the table below the figure.

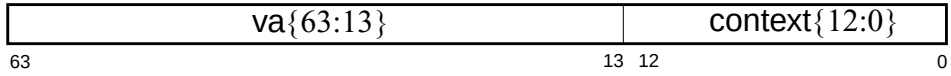


FIGURE 12-7 I/D MMU TLB Tag Access Registers

Bit	Field	Description
63:13	va	The 51-bit virtual page number. Note that writes to this field are not checked for out-of-range violation, but sign extended based on VA{47}. **NOTE:** When PSTATE.am = 1, the upper 32 bits of the VA captured in this register will be zero.
12:0	context	The 13-bit context identifier. This field reads zero when there is no associated context with the access, such as for an internal ASI or a real to physical translation.

Caution – Stores to the Tag Access registers are not checked for out-of-range violations. Reads from these registers are sign-extended based on VA{47}.

12.10.6 Partition Identifier

A partition identifier register is provided per strand to allow multiple OSs to share the same TLB. The partition identifier register contents are compared in all TLB operations such as demaps and translations, and are loaded into the pid field of the TLB tag during insertions.

The Partition Identifier register is defined in FIGURE 12-8, where pid is the 3-bit partition identifier.

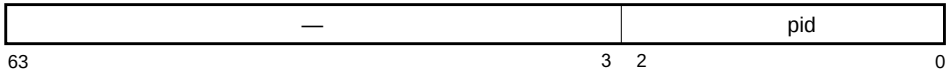


FIGURE 12-8 Partition Identifier Register

12.10.7 Hardware Tablewalk Configuration Register

Each strand has a Hardware Tablewalk Configuration register that controls operation of the Hardware Tablewalk unit.

The Hardware Tablewalk Configuration register is defined in FIGURE 12-9; the register bits are described in the table below the figure.

.

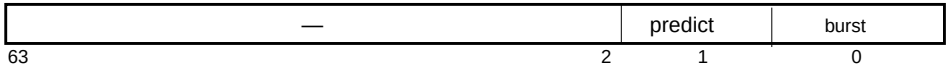


FIGURE 12-9 Hardware Tablewalk Config Register

Bit	Field	Description
1	predict	If burst is set to 0, predict controls whether hardware prediction is used to order the TSB reads. If set to 0, the order of TSBs is always 0, 1, 2, 3. If set to 1, a hardware predictor is used to order the TSB reads as either 0, 1, 2, 3 or 1, 0, 2, 3. Initial value is 0.
0	burst	If set to 1, TSB reads are issued to all four TSBs in parallel. If set to 0, TSB reads are issued sequentially, stopping when a TSB hit is detected. Initial value is 0.

12.10.8 ITLB Probe

A read-only ITLB probe ASI assists software with determining the physical address assigned to a virtual address of an instruction. The virtual address used with the ASI is presented to the ITLB, and if a translation exists for the specified address and the primary contexts in the ITLB, the result data will contain the physical page address and have the valid bit set. If a translation does not exist for the specified address and the primary contexts in the ITLB, the result data will have the valid bit clear. The ITLB probe does not trap on a probe that specifies a virtual address in the VA hole; bits 63:48 of the address are ignored.

Programming Note The ITLB probe always uses the primary context registers to determine an ITLB hit. If software desires to check for a nucleus translation, it must first zero the primary context before issuing a load to the ITLB probe ASI.

The format of the ITLB probe virtual address is shown in FIGURE 12-10; the fields are described in the table below the figure

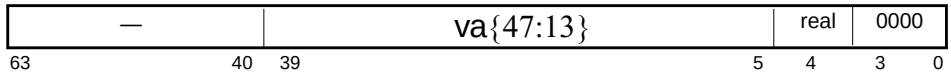


FIGURE 12-10 ITLB Probe Address Format.

Bit	Field	Description
39:13	va	The 35-bit virtual page number (real page number if the real bit is 1).
4	real	If set, the ITLB is checked for real-to-physical translations. If cleared, the ITLB is checked for virtual-to-physical translations.

The ITLB probe data format is defined in FIGURE 12-11; the fields are described in TABLE 12-19.

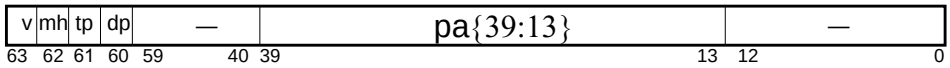


FIGURE 12-11 ITLB Probe Data Format.

TABLE 12-19 Format of ITLB Probe Data Fields

Bit	Field	Description
63	v	Valid bit for the physical page number. Set if there was a match in the TLB.
62	mh	Multiple hit. Valid only if v is 1. Set if there were multiple matches in the TLB.

TABLE 12-19 Format of ITLB Probe Data Fields

Bit	Field	Description
61	tp	Tag parity error. Valid only if v is 1 and mh is 0. Set if there was a tag parity error in the matching TLB entry.
60	dp	Data parity error. Valid only if v is 1, mh is 0, and tp is 0. Set if there was a data parity error in the matching TLB entry.
39:13	pa	The 27-bit physical page number.

12.10.9 MMU Real Range Registers

There are four Real Range registers per strand. The RPN-to-PPN translation associates each Real Range register with its corresponding Physical Offset register. The RA-to-PA translation applies to TTEs from TSBs with the ra_not_pa bit set in the TSB Config register, regardless of zero or non-zero context, as described in *Hardware Tablewalk* on page 104.

If the enable field is 0, then this range and offset pair are not used. If all range and offset pairs are disabled, any hit in a TSB with the ra_not_pa bit set in the TSB Config register results in an *instruction_invalid_TSB_entry* or *data_invalid_TSB_entry* trap.

TABLE 12-20 lists the fields of the MMU Real Range registers.

TABLE 12-20 MMU Real Range Register Format

Bit	Field	Description
63	enable	Enables range and offset pair.
62:54	—	*Reserved*
53:27	rpn_high	RA{39:13} of the upper limit of the RPN range (bounds).
26:0	rpn_low	RA{39:13} of the lower limit of the RPN range (base).

12.10.10 MMU Physical Offset Registers

There are four Physical Offset registers per strand. The RPN-to-PPN translation associates each Real Range register with its corresponding Physical Offset register. The RA-to-PA translation applies to TTEs from TSBs with the ra_not_pa bit set in the TSB Config register, regardless of zero or nonzero context, as described in Section 12.3.1.1.

Programming Note For proper operation at all page sizes, the value programmed into the ppn field must be aligned to the size of the largest page that will use the Physical Offset register for RA-to-PA translation.

TABLE 12-21 lists the fields of the MMU Physical Offset registers.

TABLE 12-21 MMU Physical Offset Register Format

Bit	Field	Description
63:40	—	*Reserved*
39:13	ppn	Added to RA{39:13} of the request to generate PA{39:13}.
12:0	—	*Reserved*

12.10.11 MMU TSB Config Registers

The TSB Config registers (MMU_{NON}ZERO_CONTEXT_TSB_CONFIG_<0,1,2,3>) to provide information for Hardware Tablewalk and for the hardware formation of TSB pointers and tag targets to assist software in handling TLB misses quickly. If the TSB concept is not employed in the software memory management strategy, and therefore the hardware tablewalk, pointer, and tag access registers are not used, then the TSB Config registers need not contain valid data other than having the enable bit set to 0. Each strand has four separate TSB pointers for both the zero and non-zero contexts.

TABLE 12-22 describes the fields of the TSB Config registers.

TABLE 12-22 TSB Config Register Format

Bit	Field	Description
63	enable	If set to 1, Hardware Tablewalk will search this TSB on TLB misses. **NOTE**: If any of a strand's TSB Config registers has the enable bit set, Hardware Tablewalk is considered to be enabled for the strand.
62	use_context_0	With use_context_1, controls whether Hardware Tablewalk checks the context value in the TTE from this TSB and what context value is written into the TTE in the TLB. If both bits are 0, then Hardware Tablewalk compares the context in the TTE from the TSB to the context of the request and stores that context into the TLB if the TTE matches. If either bit is 1, Hardware Tablewalk ignores the context of the TTE from the TSB. If use_context_0 is 1, Hardware Tablewalk writes the value of Context Register 0 to the TLB; otherwise, if use_context_1 is 1, Hardware Tablewalk writes the value of Context Register 1 to the TLB. **NOTE:** When the requesting context is zero (nucleus), Hardware Tablewalk ignores these bits.
61	use_context_1	See use_context_0 above.
60:40	—	*Reserved*
39:13	tsb_base	PA{39:13} of the base of the TSB table
12:9	—	*Reserved*
8	ra_not_pa	If set, enables RPN-to-VPN translation in Hardware Tablewalk. **CAUTION!** When using Hardware Tablewalk for a TSB, the TSB may contain either RAs or PAs, but not both. The ra_not_pa bit should be set when the TSB contains RAs.

TABLE 12-22 TSB Config Register Format *(Continued)*

Bit	Field	Description
7:4	page_size	Contains the size of the pages mapped by the TTEs in the TSB. This page size is used to generate the TSB pointer. If a reserved page size value is attempted to be stored to this field, an *unsupported_page_size* trap is taken instead.
3:0	tsb_size	The size field provides the size of the TSB according to the following: • Number of entries in the TSB = 512×2^{tsb_size}. • Number of entries in the TSB ranges from 512 entries at tsb_size = 0 (8-Kbyte TSB), to 16 M entries at tsb_size =1 5 (256-Mbyte TSB). **NOTE:** When the page size for a TSB Base is set to 5 (256-Mbyte pages), setting tsb_size to a value greater than 11 is a programming error that creates a TSB that maps a larger than 48-bit VA range. OpenSPARC T2 forces the TSB pointer bits generated by VA bits above VA{47} to be 0 for this case. **NOTE:** Any update to the TSB Config register immediately affects the data that is returned from later reads of the corresponding TSB Pointer registers.

12.10.12 MMU I-/D-TSB Pointer Registers

The per-strand TSB Pointer registers (MMU_ITSB_POINTER_<0,1,2,3>, MMU_DTSB_POINTER_<0,1,2,3>) are provided to allow software to location of a missing TTE in a software-maintained TSB.

The TSB Pointer registers are implemented as a reorder of the current data stored in the Tag Access register and the appropriate TSB Config register. If the Tag Access register or the TSB Config register is updated through a direct software write (via an STXA instruction), then the Pointer registers' values will be updated as well.

The I-/D-TSB Pointer registers are defined in FIGURE 12-12. pa{39:0} is the full physical address of the TTE in the TSB, as determined by the MMU hardware. The formula to generate this field is as follows:

```
PA{39:0} = TSB_Base{39:13+N} || VA{21+N+3*PS:13+3*PS} || 0000
```

where *N* is defined to be the tsb_size field of the TSB Config register; it ranges from 0 to 15. TSB_Base refers to the tsb_base field of the TSB Config register. PS refers to the page_size field of the TSB Config register.

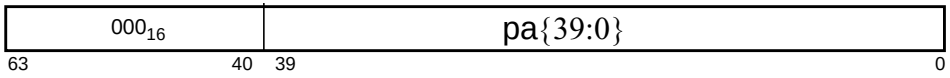


FIGURE 12-12 I-/D-TSB Pointer Registers

12.10.13 MMU Tablewalk Pending Control Register

Each strand has a MMU Tablewalk Pending Control register. This register can be used by software to indicate the status of a software tablewalk. Minimally, software should write a 1 to the stp bit before it fetches a TTE from a TSB and should write a 0 to the STP bit after it has written the TTE to the TLB or has determined that the TTE will not be written to the TLB.

Note | This register is completely maintained by software.

TABLE 12-24 lists the fields of the MMU Tablewalk Pending Control register.

TABLE 12-23 MMU Tablewalk Pending Control Register Format

Bit	Field	Description
63:1	—	*Reserved*
0	stp	Indicates whether a software tablewalk is in progress.

12.10.14 MMU Tablewalk Pending Status Register

Each physical core has a read-only MMU Tablewalk Pending Status register. This register allows software to identify when in-progress tablewalks have completed. Software can invalidate an entry in a TSB and then poll this register to identify tablewalks that may be temporarily caching the entry that has been invalidated. The bits that are 1 on the initial poll indicate pending tablewalks. A bit that initially sampled as 1 but later samples as 0 indicates an in-progress tablewalk has completed. Once each of the bits that initially were sampled as 1 have been sampled as 0, all tablewalks that were in progress when the initial poll was taken have been completed.

Programming Note | Because successive hardware tablewalks can set the htp bits again, it is possible for software to undersample. That is, polling software can miss a 1 to 0 transition if hardware clears and sets the bit between adjacent polls. Polling software should be structured to minimize the possibility of undersampling.

TABLE 12-24 lists the fields of the MMU Tablewalk Pending Control register.

TABLE 12-24 MMU Tablewalk Pending Control Register Format

Bit	Field	Description
63:40	—	*Reserved*
39:32	htp	Indicates whether a hardware tablewalk is in progress for strands 7:0.
31:8	—	*Reserved*
7:0	stp	Indicates whether a software tablewalk is in progress for strands 7:0.

12.10.15 I-/D-TLB Data-In/Data-Access/Tag-Read Registers

Access to the TLB is complicated due to the need to provide an atomic write of a TLB entry data item (tag and data) that is larger than 64 bits, the need to replace entries automatically through the TLB entry replacement algorithm as well as provide direct diagnostic access, the need to allow that multiple strands on the physical core that share the TLB to do a lock-free TLB update, and the need for hardware assist in the TLB miss handler. TABLE 12-25 shows the effect of loads and stores on the Tag Access register and the TLB.

TABLE 12-25 Effect of Loads and Stores on MMU Registers

Software Operation		Effect on MMU Physical Registers		
Load/Store	**Register**	**TLB tag**	**TLB data**	**Tag Access Register**
Load	Tag Read	No effect. Contents returned	No effect	No effect
	Tag Access	No effect	No effect	No effect. Contents returned
	Data In	Trap with *DAE_invalid_asi*		
	Data Access	No effect	No effect. Contents returned	No effect
Store	Tag Read	Trap with *DAE_invalid_asi*		
	Tag Access	No effect	No effect	Written with store data
	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
	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
TLB miss		No effect	No effect	Written with VA and context of access

The Data In and Data Access registers are the means of reading and writing the TLB for all operations. The TLB Data In register is used for TLB miss and TSB-miss handler automatic replacement writes; the TLB Data Access register is used for operating system and diagnostic directed writes (writes to a specific TLB entry). The real bit of the TLB is under the control of bit 10 of the VA. If this bit is set, the real bit of the TLB entry is set; otherwise, the real bit of the TLB entry is cleared.

Notes When a real-to-physical translation is loaded into the TLB, the context value loaded into the TLB is always 3_{16}.

Hardware Tablewalk updates the corresponding I- or D-Data In/Data Access registers (in addition to the Tag Access register) whenever it loads a translation into the TLB.

The hardware supports an autodemap function to handle the case where two strands sharing a TLB try to enter the same translation into the TLB (for example, due to near-simultaneous TLB misses on the same page). A TLB replacement that attempts to add an already existing translation will cause the existing translation to be removed from the TLB.

Notes Autodemapping of existing translations will always remove an existing page of the same size or larger than the one being added to the TLB. For example, an insertion of a 8-Kbyte page that sits inside the virtual address range of a 64-Kbyte page will cause the 64-Kbyte page to be autodemapped. Smaller pages that sit inside a page being added to the TLB may not be autodemapped. For example, an insertion of a 4-Mbyte page that overlaps the virtual address of one or more 64 KB pages may not autodemap the overlapping 64-Kbyte pages.[1] A subsequent multiple-hit error in the TLB could be generated as the result of a programming error that inserted a larger page in the TLB that overlapped smaller pages present in the TLB. A multiple-hit error occurs when a translation request matches more than one TTE in the TSB, and so a multiple-hit error only occurs for accesses of the region common to the overlapping pages.

The pids and real bits on the pages must match for autodemap to take place. If the real bit is 0, the context IDs must match as well.

IF a TLB replacement is attempted using a reserved page size value, an *unsupported_page_size* trap will be taken instead.

If a TLB replacement is attempted with the value of the valid bit (v) equal to 0, the MMU will treat that the same as if the valid bit was 1 for purposes of allocating and overwriting a TLB entry and autodemapping matching pages, and the entry will be written into the TLB with the v bit set to 0.

1. Whether the smaller pages are autodemapped depends on the actual demap address used and the position of the smaller page within the larger page.

The format of the TLB Data-In register virtual address is shown in FIGURE 12-13, where real is written to the real bit of the TLB entry.

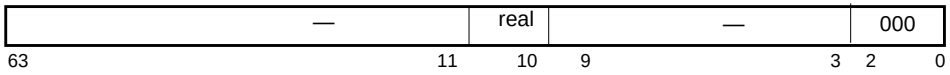


FIGURE 12-13 MMU TLB Data-In Virtual Address Format

The format of the TLB Data Access register virtual address is shown in FIGURE 12-14 and described in the table below the figure.

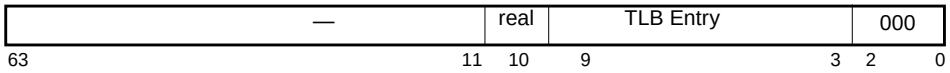


FIGURE 12-14 MMU TLB Data Access Virtual Address Format

Bit	Field	Description
10	real	Written to the Real bit of the TLB entry.
9:3	TLB Entry	The TLB Entry number to be accessed, in the range 0..63 for the ITLB, 0..127 for the DTLB.

The data format for TLB Data In and TLB Data Access registers is shown in TABLE 12-26. Reserved fields ignore writes and return all zeros on reads.

TABLE 12-26 I-/D-MMU TLB Data In and Data Access Registers

Bit	Field	Description
63	v	Valid.
62	nfo	No-fault-only.
61	parity	Parity for the TLB Data Entry.[1]
60:40	—	*Reserved*
39:13	pa	PA{39:13}.
12	ie	Invert endianess.
11	e	Side-effect.
10	cp	Cacheable in physically-indexed cache.
9	cv	*Reserved*
8	p	Privileged.
7	ep	*Reserved*
6	w	Writable.
5:4	soft	*Reserved*
3:0	size	Size.

1. Data parity is generated across pa{39:13}, nfo, ie, cp, e, p, w, and an encoded version of the page size. The page size is encoded in three bits as follows: 111– 256 Mbytes; 011 – 4 Mbytes; 001 – 64 Kbytes;
000 – 8 Kbytes. For the encoded page sizes, 256 Mbyte changes from the size{2:0} value of 101 to 111, while the encoding for all other pages match size{2:0}.

The format of the Tag Read register virtual address is shown in FIGURE 12-15 and described in the table below the figure.

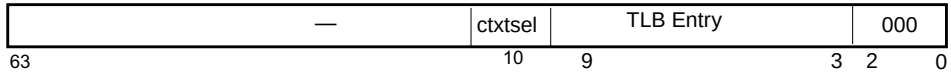


FIGURE 12-15 MMU Tag Read Virtual Address Format

Bit	Field	Description
10	ctxtsel	If 0, context A is read out in the context field. If 1, context B is read out in the context field. **NOTE:** The TLBs store duplicate copies of the context field for error detection, and ctxtsel allows software to examine both copies.
9:3	TLB Entry	The TLB Entry number to be accessed, in the range 0..63 for the ITLB, 0..127 for the DTLB

The data format for the Tag Read register is shown in FIGURE 12-16 and described in TABLE 12-27.

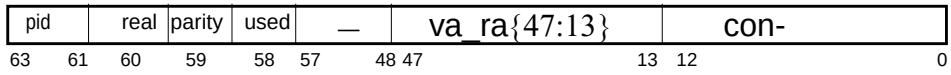


FIGURE 12-16 I-/D-MMU TLB Tag Read Registers

TABLE 12-27 Data Format for I-/D-MMU TLB Tag Read Registers

Bit	Field	Description
63:61	pid	3-bit partition identifier.
69	real	If set, identifies an RA-to-PA translation instead of a VA-to-PA.
59	parity	Parity for the tag entry. Parity is generated across pid, real, va{47:13}, and the context value that was written into the pair of context fields.

TABLE 12-27 Data Format for I-/D-MMU TLB Tag Read Registers

Bit	Field	Description
58	used	Used bit for replacement algorithm.
47:13	va_ra	If the r bit is 0, contains the lower bits of the 51-bit virtual page number (VA{47:13}). If the r bit is 1, contains the lower bits of a 51-bit real page number (RA{47:13}). Page offset bits for page sizes larger than 8 KB are stored as zeros in the TLB and returned for a Tag Read register read; that is, va_ra{15:13}, va_ra{21:13}, and va_ra{27:13} are zeroed for 64-Kbyte, 4-Mbyte, and 256-Mbyte pages, respectively. **PROGRAMMING NOTE:** Software needs to sign-extend the va_ra field based on va_ra{47}. **PROGRAMMING NOTE:** If the ra bit is 1, it is up to software to ensure that va_ra{47:40} are all zeros.
12:0	context	13-bit context identifier. The copy of context loaded from the TLB Tag entry is selected by the ctxtsel bit of the address. If ctxtsel is 0, this field contains context A If ctxtsel is 1, this field contains context B.

An ASI store to the TLB Data Access register initiates an internal atomic write to the specified TLB Entry. The TLB entry data is obtained from the store data, and the TLB entry tag is obtained from the current contents of the TLB Tag Access register.

An ASI store to the TLB Data-In register initiates an automatic atomic replacement of the TLB Entry pointed to by the replacement index generated internally by the TLB. The TLB data and tag are formed as in the case of an ASI store to the TLB Data Access register described above.

An ASI load from the TLB Data Access register initiates an internal read of the data portion of the specified TLB entry.

An ASI load from the TLB Tag Read register initiates an internal read of the tag portion of the specified TLB entry.

ASI loads from the TLB Data-In register are not supported and generate a *DAE_invalid_asi* trap.

12.11 I/D-MMU Demap

12.11.1 I-/D-MMU Demap

Demap is an MMU operation, as opposed to a register operation as described above. The purpose of demap is to remove zero, one, or more entries in the TLB. Four types of demap operation are provided: Demap Page, Demap Context, Demap All, and Demap All Pages. All demap operations only demap those pages whose PID matches the PID specified in the Partition Identifier register. Demap Page removes

zero or one[1] TLB entry that matches exactly the specified virtual page number and real bit. Demap Context removes zero, one, or many TLB entries that match the specified context identifier and have the real bit cleared. Demap Context will never demap a real translation (r = 1). Demap All Pages removes all pages that either have their real bit set (if the r bit in the demap address is set) or their real bit clear (if the r bit in the demap address is clear), regardless of their context. Demap All removes all pages, regardless of their context or real bit.

The Demap Page operation has two forms: demap page and demap real page. Address bit 10 controls which form of demap page is used. If address bit 10 is a 1, only a real translation entry in the TLB (r = 1) where the RA matches the VA portion of the demap address will be demapped (the context is ignored on demap real translation). If address bit 10 is a 0, only a virtual-to-physical translation entry in the TLB (r = 0) where the VA of the entry matches the VA of portion of the demap address and the context matches the specified context will be demapped.

Demap causes the associated tag access register to be updated; see Section 12.10.5, *I-/D-TLB Tag Access Registers*, on page 134.

Demap is initiated by an STXA with ASI = 57_{16} for I-MMU demap or $5F_{16}$ for D-MMU demap. FIGURE 12-17 shows the Demap format; TABLE 12-28 describes the fields.

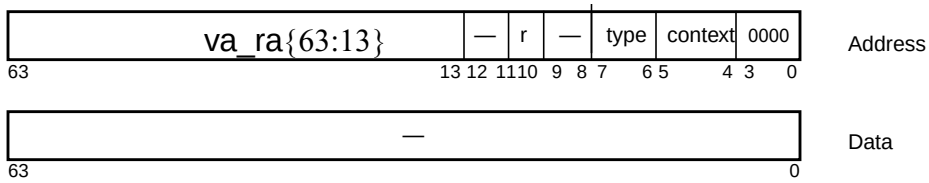


FIGURE 12-17 MMU Demap Operation Format

TABLE 12-28 Field Description for MMU Demap Operation Format

Bit	Field	Description
63:13	va_ra	The virtual page number of the TTE to be removed from the TLB; This field is not used by the MMU for the Demap Context, Demap All, or Demap All Pages operations. **NOTE**: The virtual address for demap is *not* checked for out-of-range violations; instead, va{63:48} is ignored.
10	r	Valid for Demap Page and Demap All Pages only, selects between demapping real translation(s) (r = 1) or virtual translation(s) (r = 0).

1. Demap Page may in fact remove more than one TLB entry for the error case where multiple TLB entries match the virtual page number and real bit. For this multiple match error case, Demap Page will remove all matching TLB entries.

TABLE 12-28 Field Description for MMU Demap Operation Format

Bit	Field	Description
7:6	type	The type of demap operation, as described in TABLE 12-29.
5:4	context	Context ID: Context register selection, as described in TABLE 12-30; Use of the reserved value causes the demap to be ignored for demap page and demap context, but is a valid value for a Demap All Pages or Demap All operation.

TABLE 12-29 MMU Demap Operation Type Field Description

Type Field	Demap Operation
00	Demap Page
01	Demap Context
10	Demap All
11	Demap All Pages

TABLE 12-30 MMU Demap Operation Context Field Description

Context ID Field	Context Used in Demap
00	Primary 0
01	Secondary 0
10	Nucleus
11	*Reserved*

Note | Address bits 12:11, 9:8 and 3 are ignored during a demap and may be any value.

A demap operation does not invalidate the TSB in memory. It is the responsibility of the software to modify the appropriate TTEs in the TSB before initiating any Demap operation.

The demap operation produces no output.

12.11.2 I-/D-Demap Page (type = 0)

Demap Page removes the TTE from the specified TLB matching the specified virtual page number, real bit, partition identifier register, and context register.

Virtual page offset bits {15:13}, {21:13}, and {27:13}, for 64-Kbyte, 4-Mbyte, and 256-Mbyte page TLB entries, respectively, are stored in the TLB, but are always set to zero and do not participate in the match for that entry. This is the same condition as for a translation match.

Note | For the IMMU, the Demap Page operation does not support the Secondary Context encoding, and using it will cause the demap to be ignored.

12.11.3 I-/D-Demap Context (type = 1)

Demap Context removes all TTEs from the specified TLB having the specified context, a real bit of 0, and matching the partition identifier register.

Note | For the IMMU, the Demap Context operation does not support the Secondary Context encoding, and using it will cause the demap to be ignored.

12.11.4 I-/D-Demap All (type = 2)

Demap All removes all TTEs from the specified TLB matching the partition identifier register.

12.11.5 I-/D-Demap All Pages (type = 3)

Demap Real removes all TTEs from the specified TLB matching the specified real bit and the partition identifier register.

12.12 TLB Hardware

12.12.1 TLB Operations

The TLB supports exactly one of the following operations per clock cycle:

- **Translation.** The TLB receives a virtual address or real address, a partition identifier and context identifier as input and produces a physical address and page attributes as output.
- **Demap operation.** The TLB receives a virtual address and a context identifier as input and sets the Valid bit to zero for any entry matching the demap page or demap context criteria. This operation produces no output.

- **Read operation**. The TLB reads either the CAM or RAM portion of the specified entry. (Since the TLB entry is greater than 64 bits, the CAM and RAM portions must be returned in separate reads. See *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142 for details.)
- **Write operation**. The TLB simultaneously writes the CAM and RAM portion of the specified entry, or the entry given by the replacement policy described in Section 12.12.2.
- **No operation.** The TLB performs no operation.

12.12.2 TLB Replacement Policy

OpenSPARC T2 uses a used bit scheme to generate a replacement index. Each TLB entry has an associated valid and used bit. An entry's used bit is set on each TLB translation hit and also on the write of an entry. When setting the used bit for a translation or TLB write would result in all used bits being set, the used bits for all TLB entries are cleared instead.

On an automatic write to the TLB initiated through an ASI store to the TLB Data-In register, the TLB replaces the first invalid or unused entry.

CHAPTER 13

Clocks, Reset, RED_state, and Initialization

13.1 Clock Unit

The clock unit block contains the control registers for chipwide clocking.

OpenSPARC T2 has three synchronous clock domains and two asynchronous clock domains. The synchronous clock domains consist of:

- CMP (physical processors, crossbar and L2 cache) clock domain (target 1.4 GHz)
- IO clock domain (target 350 MHz)
- Memory (DR) clock domain (target 333 MHz)

In the synchronous clock domains, all the clocks are derived from the same reference clock. There is one PLL to generate CMP, IO, and memory clocks.

The two asynchronous clock domains are

- PCI-Express clock domain 250 MHz
- Ethernet MAC clock domain 312.5 MHz

The PCI-Express clock domain derives its clock from a PLL in the Tx SerDes which is driven by an external clock. In mission mode, the external clock is asynchronous to the CMP, I/O, and memory clock domains. The Ethernet MAC also has its own clock domain asynchronous to the rest of the chip. It comes from its SerDes.

Controls for the PLL are found in the PLL Control register, whose format is shown in TABLE 13-1.

TABLE 13-1 PLL Control Register – PLL_CTL ($83\ 0000\ 0000_{16}$)

Bit	Field	Initial Value	WMR Protected	R/W	Description
63:37	—	0	—	RO	*Reserved.*
36	pll_clamp_ fltr	0	Yes	RW	PLL Clamp Filter Setting
35:34	st_delay_dr	0_{16}	Yes	RW	DR Stretch Delay Setting (40 ps intervals). [00, 01, 10, 11] → [40, 80, 120, 160] ps
33	pll_char_in	0	Yes	RW	PLL characterization test input.
32	change	1	Yes	RW	If 1, change frequency on next warm reset.
31:30	align_shift	0	Yes	RW	Shift align detect point by [-1:1] CMP cycle. Affects dr_sync pulse generation. All other sync pulses unchanged. 00 : No shift, 01 : +1 cycle, 10 : –1 cycle, 11 : No shift.
29	serdes_dtm2	0	Yes	RW	Mode 2 - IO/IO2x set to DR rate; used for observing debug data on MIO at (up to) CMP rate with DR sync en
28	serdes_dtm1	0	Yes	RW	Mode 1 - IO/IO2x set to DR rate; used for observing MCU TX CRC bits on MIO at DR rate
27:26	st_delay_ cmp	0	Yes	RW	CMP Stretch delay setting (40 ps intervals). [00, 01, 10, 11]→ [40, 80, 120, 160] ps
25	st_phase_hi	0	Yes	RW	If 1, stretch high phase of clock. If 0, stretch low phase of clock.
24:18	pll_div4	8	Yes	RW	PLL VCO divisor (D4) for DR.
17:12	pll_div3	1	Yes	RW	PLL VCO divisor (D3) for CMP.
11:6	pll_div2	7	Yes	RW	PLL feedback divisor (D2).
5:0	pll_div1	1	Yes	RW	PLL prescalar (D1).

CAUTION! The values of pll_div1, pll_div2, pll_div3, and pll_div4 in the PLL_CTL register are interdependent and must be changed together in a coherent fashion. Illegal values exist that will inhibit correct functional operation. In addition, valid values should not be reverse engineered by experimentation. Values exist that may work at some process, voltage, and temperature points, but do not allow sufficient margin for correct electrical operation across PVT combinations.

The recommended procedure for doing a frequency change with warm reset is listed below:

1. Write the PLL_CTL register to the values needed for the new frequency point. This write should have the change bit set to a 1 and must be a 64-bit write (doubleword store).
2. Generate a warm reset.

The PLL is programmed through a combination of registers (csr fields), direct chip-level pin control and combinational logic. External pin-level control is applicable typically in test mode.

Dividers D1, D2, and D3 perform integer division. D4 has fractional divide capability in discrete increments of 0.5 by using both phases of the VCO clock. The divider configurations allow cmp_pll_clk to run at different multiples of pll_sys_clk, but dr_pll_clk is always twice as fast as pll_sys_clk. The DR clock output may not have 50/50 duty cycle, but should be within 10%.This is not an issue within OpenSPARC T2 since there is no operation on the low phase.

The divider values are summarized in the table below with information on both effective and actual bits.

TABLE 13-2 PLL Divider Programming for Mission Mode

DIV	Bits	(Effective) Valid Range	Binary Encoded Values	Comments
D1	6	2	00_0001	Binary value = Effective value – 1
D2	6	8–21	00_0111 – 01_0100	Binary value = Effective value – 1
D3	6	2	00_0001	Binary value = Effective value – 1
D4	7	4.0–10.5	00_0100_0 – 00_1010_1	Binary value {6:1} = Effective value; bit 0 = 0 for integer effective, and 1 for effective x.5

Even though all four dividers can be programmed via CSR writes, there is a subset of values that are valid. D3, for example, needs to be set to divide by 2. Putting a divide by 3 or higher will result in a non 50/50 duty cycle CMP clock. dr_pll_clk may not be produced correctly since it uses both phases of the VCO clock. Acceptable values for normal operating or mission mode with corresponding clock frequencies are given in Tables13-3 and 13-4.

The clock frequency multiplication equations with respect to the frequency (fsys) of the sys_clk input pin are shown.

fvco ← (D2 × D3 ÷ D1) fsys

fcmp ← (1 /D3) fvco ← (D2 ÷ D1) fsys

fdr ← (1 /D4) fvco ← (D2 × D3) ÷ (D1 × D4) fsys

fio ← 1/4 fcmp ← (D2 ÷ 4D1) fsys

fio2x ← 1/2 fcmp ← (D2 ÷ 2D1) fsys

The first row in any of the three sets in the table below holds the default divider ratio during power-on-reset. The rows in blue (14 and 10) of the two sets refer to the targeted operating frequencies. Red sections are beyond the scope of expected operation, even though within OpenSPARC T2 there is no check for these configurations.

TABLE 13-3 Div Ratios for sys_clk = 133.33 MHz

No	Sys_clk (MHz)	Effec-tive D1	Effec-tive D2	Effec-tive D3	Effec-tive D4	D2 * D3	vco (MHz)	cmp_ clk (MHz)	io_ clk (MHz)	io2x_ clk (MHz)	dr_ clk (MHz)	cmp : dr ratio
1	133.33	2	8	2	4	16	1066.67	533.33	133.33	266.67	266.67	2.0
2	133.33	2	9	2	4.5	18	1200	600	150	300	266.67	2.25
3	133.33	2	10	2	5.0	20	1333.33	666.67	166.67	333.33	266.67	2.5
4	133.33	2	11	2	5.5	22	1466.67	733.33	183.33	367.67	266.67	2.75
5	133.33	2	12	2	6.0	24	1600	800	200	400	266.67	3.00
6	133.33	2	13	2	6.5	26	1733.33	866.67	216.67	433.33	266.67	3.25
7	133.33	2	14	2	7.0	28	1866.67	933.33	233.33	466.67	266.67	3.5
8	133.33	2	15	2	7.5	30	2000	1000	250	500	266.67	3.75
9	133.33	2	16	2	8.0	32	2133.33	1066.67	266.67	533.33	266.67	4.0
10	133.33	2	17	2	8.5	34	2266.67	1133.33	283.33	566.67	266.67	4.25
11	133.33	2	18	2	9.0	36	2400	1200	300	600	266.67	4.5
12	133.33	2	19	2	9.5	38	2533.33	1266.67	316.67	633.33	266.67	4.75
13	133.33	2	20	2	10.0	40	2666.67	1333.33	333.33	666.67	266.67	5.0
14	133.33	2	21	2	10.5	42	2800	1400	350	700	266.67	5.25

TABLE 13-4 Div Ratios for sys_clk = 166.67 MHz

No	Sys_clk (MHz)	Effec-tive D1	Effec-tive D2	Effec-tive D3	Effec-tive D4	D2 * D3	vco (MHz)	cmp_clk (MHz)	io_ clk (MHz)	io2x_clk (MHz)	dr_ clk (MHz)	cmp : dr ratio
1	166.67	2	8	2	4	16	1333.33	666.67	166.67	333.33	333.33	2.0
2	166.67	2	9	2	4.5	18	1500	750	187.5	375	333.33	2.25
3	166.67	2	10	2	5.0	20	1666.67	833.33	208.33	416.67	333.33	2.5
4	166.67	2	11	2	5.5	22	1833.33	916.67	229.17	458.33	333.33	2.75
5	166.67	2	12	2	6.0	24	2000	1000	250	500	333.33	3.0
6	166.67	2	13	2	6.5	26	2166.67	1083.33	270.83	541.67	333.33	3.25
7	166.67	2	14	2	7.0	28	2333.33	1166.67	291.67	583.33	333.33	3.5
8	166.67	2	15	2	7.5	30	2500	1250	312.5	625	333.33	3.75
9	166.67	2	16	2	8.0	32	2666.67	1333.33	333.33	666.67	333.33	4.0
10	166.67	2	17	2	8.5	34	2833.33	1416.67	354.17	708.33	333.33	4.25

TABLE 13-4 Div Ratios for sys_clk = 166.67 MHz

No	Sys_clk (MHz)	Effec-tive D1	Effec-tive D2	Effec-tive D3	Effec-tive D4	D2 * D3	vco (MHz)	cmp_clk (MHz)	io_ clk (MHz)	io2x_clk (MHz)	dr_ clk (MHz)	cmp : dr ratio
11	166.67	2	18	2	9.0	36	3000	1500	375	750	333.33	4.5
12	166.67	2	19	2	9.5	38	3166.67	1583.33	395.83	791.67	333.33	4.75
13	166.67	2	20	2	10.0	40	3333.33	1666.67	416.67	833.33	333.33	5.0
14	166.67	2	21	2	10.5	42	3500	1750	437.5	875	333.33	5.25

13.1.1 Other Clock Unit Registers

The clock unit also contains the random number generator registers.

Note | Data is shifted serially, so doing more than 1 read in 64 cycles will not produce truly random data in diagnostic mode (rng_ctl = 1).

TABLE 13-5 RNG_CTL Register (83 -0000 0020_{16})

Bit	Field	Initial Value	WMR Protected	R/W	Description
63:25	*Reserved*	0_{16}	—	R	*Reserved*
24:9	rng_wait_ cnt	$003E_{16}$	No	RW	Minimum wait time before successive rng data is sent.
8	rng_bypass	0_{16}	No	RW	Controls VCO voltage source. 0 = sets noise cell VCO control voltage = output of feedback amplifier. 1 sets noise cell VCO control voltage = output of bias generator
7:6	rng_vcoctrl_sel	0_{16}	No	RW	Pmos diode D/A setting bus. Controls VCO rate for each noise cell.
5:4	rng_anlg_ sel	0_{16}	No	RW	Analog mux select for characterization.
3	rng_ctl4	1_{16}	No	RW	Enables using LFSR or plain shift register. Set to LFSR mode by default.
2	rng_ctl3	1_{16}	No	RW	Control for using noise cell 3.
1	rng_ctl2	1_{16}	No	RW	Control for using noise cell 2.
0	rng_ctl1	1_{16}	No	RW	Control for using noise cell 1.

TABLE 13-6 RNG_DATA Register (83 0000 0030_{16})

Bit	Field	Initial Value	WMR Protected	R/W	Description
63:0	rng_data	X	—	R	Random-number-generator data.

The random number generator (rng) generates random numbers from three noise cells. There is one rng block and one LFSR (Linear Feedback Shift Register) to be shared among the eight processor cores. Only one of the cells may be active at a time, all three may be active, or none of them may be active. Any other combination defaults to selecting all three noise cells. The following encoding applies:

TABLE 13-7 Encoding for Noise Cell Selection

rng_ctl3	rng_ctl3	rng_ctl3	Effect
0	0	0	Deselect all noise cells (feeds 0 into LFSR)
0	0	1	Select noise cell 1
0	1	0	Select noise cell 2
1	0	0	Select noise cell 3
011,101,110,111			Select all 3 noise cells

The raw generators will serially output 1 data bit into a 64-bit register. Under functional mode, the register generates data by implementing the CRC polynomial.

P(x) = x64 + x61 + x57 + x56 + x52 + x51 + x50 + x48 + x47 + x46 + x43 + x42 + x41 + x39 + x38 + x37 + x35 + x32 + x28 + x25 + x22 + x21 + x17 + x15 + x13 + x12 + x11 + x7 + x5 + x + 1

After each read request, it is important to not maintain any correlation with the past generated values, so the LFSR will be flushed after every read acknowledge. The register will be flushed with a non-zero state $FFFF\_FFFF\_FFFF\_FFFF_{16}$. Also, multiple requests for rng_data are automatically separated by $n + 2$ cycles, where n can be programmed by writing to the 16-bit field rng_wait_cnt in the RNG_CTL register.

In diagnostic mode (ctl4 = 0), the LFSR acts as a simple shift register capturing the noise cell output directly, determined independently by ctl1, ctl2, and ctl3 as per encoding. The additional constraint in this mode is that successive read requests for the rng_data will be delayed by 64 iol2clk cycles. Also, flushing the LFSR after every read will be disabled in this mode.

The nominal frequency of the oscillator in each noise cell that generates a serial output can also be set independently by programming the rng_vcoctrl_sel{1:0} field. There are four settings that correspond to four different frequencies; however, each

cell must be programmed one at a time. As an example, consider the following configuration: noise cell1 → 00 setting, cell2 → 10 setting, cell 3 → 01 setting, and observe all 3 cells. One would proceed as follows:

1. Set CTL3,CTL2,CTL1 = 001 and set RNG_VCO_CTRL = 00

2. Set CTL3,CTL2,CTL1 = 010 and set RNG_VCO_CTRL = 10

3. Set CTL3,CTL2,CTL1 = 100 and set RNG_VCO_CTRL = 01

13.2 Reset Unit

13.2.1 Reset Generation

The Reset Generation register, shown in TABLE 13-8, allows software to generate an external (XIR) resets to all processors specified in the ASI_XIR_STEERING register or a chipwide warm reset (WMR) or debug reset (DBR).

TABLE 13-8 Reset Generation Register – RESET_GEN ($89\ 0000\ 0808_{16}$)

Bit	Field	Initial Value	R/W	Description
63:3	—0	0	RO	*Reserved*
3	dbr_gen	0	RW	Set to 1 to generate a DBR. Value is automatically cleared once the DBR is complete.
2	—	0	RO	*Reserved* (was por_gen on Fire).
1	xir_gen	0	RW	Set to 1 to generate an XIR. Value is automatically cleared once the XIR is complete.
0	wmr_gen	0	RW	Set to 1 to generate a WMR. Value is automatically cleared once the WMR is complete.

Programming Note Software may only write a 1 to one of the reset generation bits at a time. Behavior of OpenSPARC T2 is undefined if software writes 1's to multiple reset generation bits.

13.2.2 Reset Source

The Reset Source register, shown in TABLE 13-9 allows software to identify the source of a reset. The bits in this register are write-one to clear.

TABLE 13-9 Reset Source Register – RESET_SOURCE ($89\ 0000\ 0818_{16}$)

Bit	Field	Initial Value	R/W	Description
63:16	—0	0	RO	*Reserved*
15	l2t7_fatal	0	RW1C	Bank 7 of the L2 cache detected a fatal error.
14	l2t6_fatal	0	RW1C	Bank 6 of the L2 cache detected a fatal error.
13	l2t5_fatal	0	RW1C	Bank 5 of the L2 cache detected a fatal error.
12	l2t4_fatal	0	RW1C	Bank 4 of the L2 cache detected a fatal error.
11	l2t3_fatal	0	RW1C	Bank 3 of the L2 cache detected a fatal error.
10	l2t2_fatal	0	RW1C	Bank 2 of the L2 cache detected a fatal error.
9	l2t1_fatal	0	RW1C	Bank 1 of the L2 cache detected a fatal error.
8	l2t0_fatal	0	RW1C	Bank 0 of the L2 cache detected a fatal error.
7	ncu_fatal	0	RW1C	The NCU or a block interfacing to it detected a fatal error.
6	pb_xir	0	RW1C	The user asserted the BUTTON_XIR_ input pin.
5	pb_rst	0	RW1C	The user asserted the PB_RST_L input pin.
4	pwron_rst	1	RW1C	The system processor asserted the PWRON_RST_L input pin.
3	dbr_gen	0	RW1C	Software wrote a 1 to the dbr_gen field of the RESET_GEN register.
2	—	0	RO	*Reserved* (In Fire was, software wrote a 1 to the por_gen field of the RESET_GEN register).
1	xir_gen	0	RW1C	Software wrote a 1 to the xir_gen field of the RESET_GEN register.
0	wmr_gen	0	R/W1C	Software wrote a 1 to the wmr_gen field of the RESET_GEN register.

13.2.3 Reset Fatal Error Enable

The Reset Fatal Error Enable register, shown in TABLE 13-10 allows software to control whether L2 fatal errors generate a warm reset.

TABLE 13-10 Reset Fatal Error Enable Register – RESET_FEE ($89\ 0000\ 0820_{16}$)

Bit	Field	Initial Value	R/W	Description
63:16	—0	0	RO	*Reserved*
15	l2t7_fee	0	RW	Allow Bank 7 of the L2 cache to generate a fatal error.
14	l2t6_fee	0	RW	Allow Bank 6 of the L2 cache to generate a fatal error.
13	l2t5_fee	0	RW	Allow Bank 5 of the L2 cache to generate a fatal error.
12	l2t4_fee	0	RW	Allow Bank 4 of the L2 cache to generate a fatal error.
11	l2t3_fee	0	RW	Allow Bank 3 of the L2 cache to generate a fatal error.
10	l2t2_fee	0	RW	Allow Bank 2 of the L2 cache to generate a fatal error.

TABLE 13-10 Reset Fatal Error Enable Register – RESET_FEE ($89\ 0000\ 0820_{16}$)

Bit	Field	Initial Value	R/W	Description
9	l2t1_fee	0	RW	Allow Bank 1 of the L2 cache to generate a fatal error.
8	l2t0_fee	0	RW	Allow Bank 0 of the L2 cache to generate a fatal error.
7:0	—	0	RO	*Reserved*

13.2.4 Subsystem Reset

The Subsystem Reset Generation register, shown in TABLE 13-11, allows software to reset selected I/O subsystems.

TABLE 13-11 Subsystem Reset Generation Register – SSYS_RESET ($89\ 0000\ 0838_{16}$)

Bit	Field	Initial Value	R/W	Description
63:7	—0	0	RO	*Reserved*
6	—	0	RW	
5	—	0	RW	
4	—	0	RW	*Reserved* (was set to 1 to protect the FBDIMM interfaces of each MCU from being reset by either warm reset or debug reset).
3:2	—2	0	RO	*Reserved*
1	—	0	RW	
0	—	0	RW	

13.2.5 Reset Status

The chip Reset Status register, shown in TABLE 13-12, is maintained for all chipwide reset and power management commands. The reset source bits in this register are writable to allow software to clear them after the chip reset sequence is complete, in order for virtual processor warm resets to be distinguished from chip resets. Hardware will copy the current reset status into a shadow status whenever a warm reset occurs.

TABLE 13-12 Reset Status Register – RESET_STAT $(89\ 0000\ 0810)_{16}$

Bit	Field	Initial Value	R/W	Description
63:12	—0	0	RO	*Reserved*
11	freq_s	0	RO	Shadow status of FREQ
10	por_s	0	RO	Shadow status of POR
9	wmr_s	0	RO	Shadow status of WMR

TABLE 13-12 Reset Status Register – RESET_STAT (89 0000 0810)$_{16}$

Bit	Field	Initial Value	R/W	Description
8:5	—	0	RO	*Reserved*
4	—2	0	RW	*Reserved*
3	freq	0	RW	Set to 1 if the reset is a warm reset that changed frequency.
2	por	1	RW	Set to 1 if the reset is from PWRON_RST_L pin.
1	wmr	0	RW	Set to 1 if the reset is from the PB_RST_L pin.
0	—	0	RO	*Reserved*

13.2.6 Lock Time

The Lock Time register determines the length of time the Reset Unit in OpenSPARC T2 waits for all the PLLs in OpenSPARC T2 to lock. The initial value is an estimated time only that software can reprogram during the warm reset sequence. Moreover, software can enable the pre-WMR boot code to perform warm reset with the same PLL configuration register values, obviating the need to wait for the PLLs to relock.

TABLE 13-13 Lock Time Register – LOCK_TIME (89 0000 0870$_{16}$)

Bit	Field	Initial Value	R/W	Description
63:16	—	0	RO	*Reserved*
15:0	lock_time	1400$_{16}$	RW	The length of time the Reset Unit in OpenSPARC T2 waits for all the PLLs in OpenSPARC T2 to lock.

13.2.7 Propagation Time

The Propagation Time register indicates how long it takes for the longest scan chain to flush. The register initializes to the estimated longest time needed (assuming the highest planned reference clock frequency), that software can reprogram during warm reset sequence.

TABLE 13-14 Propagation Time Register – PROP_TIME (89 0000 0880$_{16}$)

Bit	Field	Initial Value	R/W	Description
63:16	—	0	RO	*Reserved*
15:0	prop_time	C00$_{16}$	RW	Time taken for longest scan chain to flush.

13.3 Reset Overview

A reset is anything that causes an entry to RED_state. Two classes of resets exist: chipwide and virtual processor.

Chipwide resets are power-on reset (POR), warm reset (WMR), and debug reset (DBR). POR affects all subsystems in OpenSPARC T2, while WMR affects all subsystems and DBR affects all subsystems except PIU. Chipwide resets are generated from the PWRON_RST_L input pin (POR), PB_RST_L pin (WMR), by software writing a 1 to the wmr_gen field of the RESET_GEN register (WMR) or to the dbr_gen field of the RESET_GEN register (DBR) and from fatal errors in the processor (WMR).

Virtual processor resets are XIR, WDR, SIR and are generated by the BUTTON_XIR_ pin (XIR), software resets (SIR), software writing a 1 to the xir_gen field of the RESET_GEN register (XIR), and error conditions (WDR), and only affect the operation of a single virtual processor (or for the case of XIR, only the virtual processors specified in XIR_STEERING as described in *ASI_XIR_STEERING* on page 181). In addition to forcing entry to RED_state, various resets cause different effects in initializing processor state, as discussed in the following sections. Reset priorities from highest to lowest are: POR, WMR, DBR, XIR, WDR, SIR. Resets are not maskable (that is, resets ignore PSTATE.ie).

Programming Note Chipwide resets cause the virtual processors to enter RED_state and initialize some, but not all, of the processor state. Significant programming effort is required to take the OpenSPARC T2 chip from a chipwide reset to the point where the operating system can be loaded.

13.4 Chipwide Resets

Chipwide resets affect all virtual processors in a chip, as well as all I/O, cache, and DRAM subsystems and are categorized as power-on or warm reset. Power-on reset is used when the chip power and clock inputs are outside their operating specifications. Warm reset is used when the power and clock inputs are stable. Warm reset is typically used to modify clock frequencies or ratios, or to reinitialize the chip after an unrecoverable hardware or software failure. Warm reset resets all subsystems in OpenSPARC T2.

13.4.1 Power-on Reset (POR)

A power-on reset occurs when the PWRON_RST_L pin is asserted and then deasserted. The PWRON_RST_L pin must be asserted until 1 clock after the CPU voltages and input clocks reach their operating specifications. When the PWRON_RST_L pin is asserted, all other resets and traps are ignored. Power-on reset has a trap type of 001_{16} at physical address offset 20_{16}. Since POR and warm reset share the same trap type and trap vector, the RSET_STAT register described in Section 13.2.5 has separate POR and warm reset bits to allow software to distinguish between POR and warm resets. All pending transactions are cancelled. Strand 0 of the first available physical core begins executing at the `RED_State_Trap_Vector` base plus POR offset, while the remaining strands start out inactive. BIST testing may optionally be initiated by software as part of the chip initialization sequence.

After a power-on reset, software must initialize values specified as *unknown* in *Machine State After Reset and in RED_State* on page 164.

Note | Each unknown register must be initialized before it is used. Failure to initialize registers or states properly before use may result in unpredictable or incorrect results.

13.4.2 Warm Reset (WMR)

A warm reset occurs when the PB_RST_L pin is asserted and then deasserted, when software writes a 1 to the wmr_gen field of the RESET_GEN register, or when a fatal error is detected in the processor. When a warm reset is received, all other resets and traps except POR are ignored. Warm reset has the same trap type and vector as power-on reset: a trap type of 001_{16} at physical address offset 20_{16}. Software can distinguish between POR and the various sources of warm reset by checking the RSET_STAT register. The memory controller places DRAM in self-refresh mode prior to warm reset only if SSYS_RESET.mcu_selfrsh bit is set by software prior to the warm reset. Otherwise, on a warm reset, the memory controller does not put the DRAM in self-refresh.Warm reset can be programmed to do BIST testing. After warm reset, strand 0 of the first available physical core begins executing at the `RED_State_Trap_Vector` base plus POR offset, while the remaining strands start out inactive.

After a warm reset, software must initialize values specified as unknown in *Machine State After Reset and in RED_State* on page 164. If there was a clean shutdown, the primary instruction, primary data, L2 caches, and main memory are still valid. Otherwise, I-cache tags, D-cache tags, and L2 cache tags should be initialized before enabling the caches. The iTLB and dTLB also must be initialized before enabling memory management.

Note that if a warm reset is received without software first placing the chip in a quiescent state, the hardware will still maintain the state of the primary instruction, primary data, L2 caches, main memory, and all error registers/logs. However, the caches and main memory may no longer be completely coherent after the warm reset, because any transactions in flight when the warm reset was received will have been lost. In particular, dirty lines in the process of being written back to main memory may have been dropped.

13.4.3 Debug Reset (DBR)

A debug reset occurs when software writes a 1 to the dbr_gen field of the RESET_GEN register. DBR behaves the same as warm reset, except that the PIUs are not reset.

13.5 Virtual Processor Resets

Virtual processors can receive reset traps. Virtual processor reset traps do not set any bits in the RSET_STAT register.

13.5.1 Externally Initiated Reset (XIR)

An externally initiated reset can be generated either from the BUTTON_XIR_ pin or by writing a 1 to the xir_gen field of the RESET_GEN register. When either of these events occurs, an XIR reset trap is sent to all virtual processors specified in the XIR_STEERING register. This trap causes a SPARC V9 XIR, which has a trap type of 003_{16} at physical address offset 60_{16}. It has higher priority than all other virtual processor core resets. XIR is used for system debug.

13.5.2 Watchdog Reset (WDR) and `error_state`

A SPARC V9 WDR is generated when a virtual processor encounters a trap when TL = *MAXTL*, it passes through `error_state` and signals itself internally to take a WDR trap. Window traps that cause watchdog traps still update CWP if they would have done so with no watchdog trap being generated.

13.5.3 Software-Initiated Reset (SIR)

A SPARC V9 SIR interrupt can be generated on a virtual processor by issuing a SIR instruction while operating in hyperprivileged mode. This virtual processor reset has a trap type of 004_{16} at physical address offset 80_{16}.

13.6 RED_state

RED_state is an acronym for Reset, Error, and Debug State. RED_state is described in the UltraSPARC Architecture 2007 specification.

13.7 RED_state Trap Vector

When a SPARC V9 virtual processor processes a reset or trap that enters RED_state, it takes a trap at an offset relative to the RED_state_trap_vector base address (*RSTVADDR*). The trap offset depends on the type of red mode trap and takes the values:

- POR, WMR, or DBR 20_{16}
- XIR 60_{16}
- WDR 40_{16}
- SIR 80_{16}
- other $A0_{16}$

In OpenSPARC T2 the RSTV base address is FFFF FFFF F000 0000_{16} if ASI_RST_VEC_MASK.vec_mask = 0. If ASI_RST_VEC_MASK.vec_mask = 1, RSTV base address is 0000 0000 0000 0000_{16}. See *ASI_RST_VEC_MASK* on page 454 for more details on the ASI_RST_VEC_MASK register.

13.8 Machine State After Reset and in RED_State

TABLE 13-15 shows CPU state created as a result of any reset, or after entering RED_state.

TABLE 13-15 CPU State After Reset and in `RED_state` *(1 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
Integer Registers		0	Unchanged				
Floating Point Registers		0	Unchanged				
iTLB/dTLB	Mappings	All invalid	Unchanged				
PSTATE	tct	0 (Trap on control transfer)					
	mm	0 (TSO)					
	red	0 (`RED_state` bit is in HPSTATE register)					
	pef	1 (FPU on)					
	am	0 (Full 64-bit addresses)					
	priv	1					
	ie	0 (Disable interrupts)					
	ag	0 (Alternate globals always 0)					
	cle	0 (Current not little endian)					
	tle	0 (Trap not little endian)		Unchanged			
	ig	0 (Interrupt globals always 0)					
	mg	0 (MMU globals always 0)					
HPSTATE	ibe	0 (Instruction breakpoint disabled)					
	red	1 (RED_state)					
	hpriv	1 (Hyperprivileged mode)					
	tlz	0 (tlz traps disabled)					
TBA{63:15}		0	Unchanged				
HTBA{63:15}		0	Unchanged				
Y		0	Unchanged				
PIL		0	Unchanged				
CWP		0	Unchanged (except for window traps)				
PC		RSTV | 20_{16}	RSTV | 20_{16}	RSTV | 40_{16}	RSTV | 60_{16}	RSTV80_{16}	RSTV | $A0_{16}$
NPC		RSTV | 24_{16}	RSTV | 24_{16}	RSTV | 44_{16}	RSTV | 64_{16}	RSTV | $_{16}84$	RSTV | $A4_{16}$
TT[TL]		1	1	2 or Trap type	3	4	Trap type
TPC[TL]			PC				
TNPC[TL]		0	NPC				
Store Buffer		Empty		Unchanged			Empty
CCR		0	Unchanged				
ASI		0	Unchanged				
TL		*MAXTL*		**min**(TL+1,*MAXTL*)			
GL		*MAXGL*		**min**(GL+1,*MAXGL*)			

TABLE 13-15 CPU State After Reset and in `RED_state` *(2 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
TSTATE[TL]	GL	0	pre-WMR[1]	GL			
	CCR	0	pre-WMR[1]	CCR			
	ASI	0	pre-WMR[1]	ASI			
	PSTATE	0	0_{16}	PSTATE			
	CWP	0	pre-WMR[1]	CWP			
HTSTATE[TL]	ibe	0	0	HPSTATE.ibe			
	red	0	0	HPSTATE.red			
	hpriv	0	0	HPSTATE.hpriv			
	tlz	0	0	HPSTATE.tlz			
TICK	npt	1	1	Unchanged			
	counter	0	Count	Count			
CANSAVE		6_{16}	Unchanged				
CANRESTORE		0_{16}	Unchanged				
OTHERWIN		0_{16}	Unchanged				
CLEANWIN		7_{16}	Unchanged				
WSTATE	other	0_{16}	Unchanged				
	normal	0_{16}	Unchanged				
VER	manuf	$003E_{16}$					
	impl	0024_{16}					
	mask	Mask-dependent (4 bits major, 4 bits minor)					
	maxtl	6					
	maxgl	3					
	maxwin	7					
FSR	all	0	Unchanged				
FPRS	all	4_{16}	Unchanged				
GSR	all	0	Unchanged				
PERF_CONTROL (PCR)	all	0 (off)	Unchanged				
PIC		0	Unchanged				
TICK_CMPR	int_dis	1	Unchanged				
	tick_cmpr	0	Unchanged				
STICK_CMPR	int_dis	1	Unchanged				
	stick_cmpr	0	Unchanged				
HSTICK_CMPR	int_dis	1	Unchanged				
	hstick_cmpr	0	Unchanged				
HINTP		0	Unchanged				
SOFTINT		0	Unchanged				
`ASI_SCRATCHPAD_0_REG`		0	Unchanged				

TABLE 13-15 CPU State After Reset and in RED_state *(3 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
ASI_SCRATCHPAD_1_REG		0	Unchanged				
ASI_SCRATCHPAD_2_REG		0	Unchanged				
ASI_SCRATCHPAD_3_REG		0	Unchanged				
ASI_SCRATCHPAD_6_REG		0	Unchanged				
ASI_SCRATCHPAD_7_REG		0	Unchanged				
ASI_PRIMARY_CONTEXT_0		0	Unchanged				
ASI_SECONDARY_CONTEXT_0		0	Unchanged				
ASI_PRIMARY_CONTEXT_1		0	Unchanged				
ASI_SECONDARY_CONTEXT_1		0	Unchanged				
ASI_CPU_MONDO_QUEUE_HEAD		0	Unchanged				
ASI_CPU_MONDO_QUEUE_TAIL		0	Unchanged				
ASI_DEVICE_QUEUE_HEAD		0	Unchanged				
ASI_DEVICE_QUEUE_TAIL		0	Unchanged				
ASI_RES_ERROR_QUEUE_HEAD		0	Unchanged				
ASI_RES_ERROR_QUEUE_TAIL		0	Unchanged				
ASI_NONRES_ERROR_QUEUE_HEAD		0	Unchanged				
ASI_NONRES_ERROR_QUEUE_TAIL		0	Unchanged				
ASI_CORE_AVAILABLE		FFFFFFFFFFFFFFFF$_{16}$ (if all cores available)	Unchanged				
ASI_CORE_ENABLE_STATUS		ASI_CORE_AVAILABLE	ASI_CORE_ENABLE	Unchanged			
ASI_CORE_ENABLE		ASI_CORE_AVAILABLE	Unchanged				
ASI_XIR_STEERING		ASI_CORE_AVAILABLE	ASI_CORE_ENABLE	Unchanged			
ASI_CMT_TICK_ENABLE		0	Unchanged				
ASI_CORE_RUNNING_RW		1_{16} (or lowest enabled strand)		Unchanged			
ASI_CORE_RUNNING_STATUS		1_{16} (or lowest enabled strand)		Unchanged			
ASI_INST_MASK_REG		0		Unchanged			
ASI_LSU_DIAG_REG		0		Unchanged			
ASI_ERROR_INJECT_REG		0		Unchanged			
ASI_LSU_CONTROL_REG		0					
ASI_DECR		0		Unchanged			
ASI_RST_VEC_MASK		0	Unchanged				

TABLE 13-15 CPU State After Reset and in RED_state *(4 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
ASI_DESR		0	Unchanged				
ASI_DFESR		0	Unchanged				
ASI_CERER		0		Unchanged			
ASI_CETER		0		Unchanged			
ASI_clesr		0	Unchanged				
ASI_CLFESR		0	Unchanged				
ASI_SPARC_PWR_MGMT		0		Unchanged			
ASI_HYP_SCRATCHPAD_0_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_1_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_2_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_3_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_4_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_5_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_6_REG		0	Unchanged				
ASI_HYP_SCRATCHPAD_7_REG		0	Unchanged				
ASI_IMMU_TAG_TARGET		0	Unchanged				
ASI_IMMU_SFSR		0	Unchanged				
ASI_IMMU_TAG_ACCESS		0	Unchanged				
ASI_IMMU_VA_WATCHPOINT		0	Unchanged				
ASI_MMU_REAL_RANGE_0		0	Unchanged				
ASI_MMU_REAL_RANGE_1		0	Unchanged				
ASI_MMU_REAL_RANGE_2		0	Unchanged				
ASI_MMU_REAL_RANGE_3		0	Unchanged				
ASI_MMU_PHYSICAL_OFFSET_0		0	Unchanged				
ASI_MMU_PHYSICAL_OFFSET_1		0	Unchanged				
ASI_MMU_PHYSICAL_OFFSET_2		0	Unchanged				
ASI_MMU_PHYSICAL_OFFSET_3		0	Unchanged				
ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_0		0	Unchanged				
ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_1		0	Unchanged				
ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_2		0	Unchanged				
ASI_MMU_ZERO_CONTEXT_TSB_CONFIG_3		0	Unchanged				
ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_0		0	Unchanged				
ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_1		0	Unchanged				

TABLE 13-15 CPU State After Reset and in RED_state *(5 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_2		0	Unchanged				
ASI_MMU_NONZERO_CONTEXT_TSB_CONFIG_3		0	Unchanged				
ASI_MMU_ITSB_PTR_0		0	Unchanged				
ASI_MMU_ITSB_PTR_1		0	Unchanged				
ASI_MMU_ITSB_PTR_2		0	Unchanged				
ASI_MMU_ITSB_PTR_3		0	Unchanged				
ASI_MMU_DTSB_PTR_0		0	Unchanged				
ASI_MMU_DTSB_PTR_1		0	Unchanged				
ASI_MMU_DTSB_PTR_2		0	Unchanged				
ASI_MMU_DTSB_PTR_3		0	Unchanged				
ASI_PENDING_TABLEWALK_CONTROL		0	Unchanged				
ASI_PENDING_TABLEWALK_STATUS		0	Unchanged				
ASI_DMMU_TAG_TARGET		0	Unchanged				
ASI_DMMU_SFSR		0	Unchanged				
ASI_DMMU_SFAR		0	Unchanged				
ASI_DMMU_TAG_ACCESS		0	Unchanged				
ASI_DMMU_WATCHPOINT		0	Unchanged				
ASI_HWTW_CONFIG		0	Unchanged				
ASI_PARTITION_ID		0	Unchanged				
ASI_CMT_CORE_INTR_ID		COREID					
ASI_CMT_CORE_INTR_ID		$7003F0000_{16}$ | COREID					
ASI_INTR_RECEIVE		0	Unchanged				
PLL_CTL		$1000204E1_{16}$	Unchanged				
I/D cache tags		All invalid	Unchanged if BISI not run, else invalid				
L2 tags and data		Unknown	Unchanged if BISI not run, else invalid				
L2 directory		All invalid	Unchanged if BISI not run, else invalid				
L2 Error En Reg	all	0 (reporting disabled)	Unchanged				
L2 Error Status Reg	synd	Unknown	Unchanged				
	Other fields	0	Unchanged				
L2 Error Address		Unknown	Unchanged				
L2 NotData Error	vcid	Unknown	Unchanged				
	rw	Unknown	Unchanged				
	address	Unknown	Unchanged				
	Other fields	0	Unchanged				

TABLE 13-15 CPU State After Reset and in `RED_state` *(6 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
L2 Error Inject		0			Unchanged		
L2 Mask Reg		0			Unchanged		
L2 Address Compare Reg		0			Unchanged		
L2 Bank Available		FF_{16} (if all banks available)			Unchanged		
L2 Bank Enable		FF_{16} (if all banks available)			Unchanged		
L2 Bank Enable Status		$F00_{16}$ (if all banks available)	$F0F_{16}$ (if all banks available)	Unchanged			
L2 Index Hash Enable		0_{16}			Unchanged		
L2 Control Reg		1_{16}			Unchanged		
DRAM Refresh Counter		0			Unchanged		
DRAM Error Status Register	`synd`	Unknown			Unchanged		
	Other fields	0			Unchanged		
DRAM Error Address		Unknown			Unchanged		
DRAM Error Inject		0			Unchanged		
DRAM Error Counter		0			Unchanged		
DRAM FBD Error Syndrome		0			Unchanged		
DRAM Error Location		Unknown			Unchanged		
DRAM Debug Enable Trigger	`dbg_en`	0			Unchanged		
	Other fields	1_{16}			Unchanged		
DRAM CAS Address Width		B_{16}			Unchanged		
DRAM RAS Address Width		F_{16}			Unchanged		
DRAM CAS Latency		3_{16}			Unchanged		
DRAM Scrub Frequency		FFF_{16}			Unchanged		
DRAM Refresh Frequency		514_{16}			Unchanged		
DRAM Open Bank Max		0			Unchanged		
DRAM Scrub Enable		0			Unchanged		
DRAM Programmable Time Counter		0			Unchanged		
DRAM RAS to RAS Different Banks Delay		0			Unchanged		
DRAM RAS to RAS Same Bank Delay		C_{16}			Unchanged		
DRAM RAS to CAS Delay		3			Unchanged		
DRAM Write to Read CAS Delay		0			Unchanged		

TABLE 13-15 CPU State After Reset and in `RED_state` *(7 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
DRAM Read to Write CAS Delay		0			Unchanged		
DRAM Internal Read to Precharge Delay		2_{16}			Unchanged		
DRAM Active to Precharge Delay		9_{16}			Unchanged		
DRAM Precharge Command Period		3_{16}			Unchanged		
DRAM Write Recover Period		3_{16}			Unchanged		
DRAM Autorefresh to Active Period		27_{16}			Unchanged		
DRAM Mode Register Set Command Period		2_{16}			Unchanged		
DRAM Four Activate Window		2_{16}			Unchanged		
DRAM Internal Write to Read Command Delay		2_{16}			Unchanged		
DRAM DIMM Stacked		0			Unchanged		
DRAM Extended Mode (2)		0			Unchanged		
DRAM Extended Mode (1)		18_{16}			Unchanged		
DRAM Extended Mode (3)		0			Unchanged		
DRAM 8 Bank Mode		1_{16}			Unchanged		
DRAM Branch Disabled		0			Unchanged		
DRAM Select Low Order Address Bits		0			Unchanged		
DRAM Single Channel Mode		0			Unchanged		
DRAM DIMMs Present		1_{16}			Unchanged		
DRAM Fail-Over Status		0			Unchanged		
DRAM Fail-Over Mask		0			Unchanged		
FBD Channel State		0			Unchanged		
FBD Fast Reset Flag		0			Unchanged		
FBD Channel Reset		0			Unchanged		
TS1 Southbound to Northbound Mapping		0			Unchanged		
TS1 Test Parameter		0			Unchanged		
TS3 Failover Configuration		$FFFF_{16}$			Unchanged		
Disable State Period		$3F_{16}$			Unchanged		
Calibrate State Period		0			Unchanged		
Training State Minimum Time		FF_{16}			Unchanged		
Training State Timeout		FF_{16}			Unchanged		
Testing State Timeout		FF_{16}			Unchanged		
Polling State Timeout		FF_{16}			Unchanged		

TABLE 13-15 CPU State After Reset and in `RED_state` *(8 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
DRAM Per-Rank CKE		$FFFF_{16}$	Unchanged				
L0s Duration		$2A_{16}$	Unchanged				
Channel Sync Frame Frequency		$2A_{16}$	Unchanged				
SerDes Configuration Bus		0	Unchanged				
SerDes Transmitter and Receiver Differential Pair Inversion		0	Unchanged				
SerDes Test Configuration Bus		$C000_{16}$	Unchanged				
DRAM FBD Injected Error Source		0	Unchanged				
DRAM FBR Count		0	Unchanged				
IMU Error Log Enable Register		$7FFF_{16}$	Unchanged				
IMU Error Status Clear Register		0_{16}	Unchanged				
IMU Error Status Set Register		0_{16}	Unchanged				
IMU RDS Error Log Register		0_{16}	Unchanged if any Primary Error bit in IMU Error Status Clear Register in RDS Group is set				
IMU SCS Error Log Register		0_{16}	Unchanged if any Primary Error bit in IMU Error Status Clear Register in SCS Group is set				
IMU EQS Error Log Register		0_{16}	Unchanged if any Primary Error bit in IMU Error Status Clear Register in EQS Group is set				
MMU Error Log Enable Register		$1FFFFF_{16}$	Unchanged				
MMU Error Status Clear Register		0_{16}	Unchanged				
MMU Translation Fault Address Register		0_{16}	Unchangedonly if any Primary Error bit is set inMMU Error Status Clear Register				
MMU Translation Fault Status Register		0_{16}	Unchangedonly if any Primary Error bit is set inMMU Error Status Clear Register				
MMU TTE Cache Data Registers		0_{16}	Unchanged				
MMU DEV2IOTSB Registers		0_{16}	Unchanged				
MMU IOTSBDESC Registers		0_{16}	Unchanged				
ILU Error Log Enable Register		$F0_{16}$	Unchanged				
ILU Error Status Clear Register		0_{16}	Unchanged				
ILU Error Status Set Register		0_{16}	Unchanged				
DMU ILU Diagnostic Register		0_{16}	Unchanged (only bits 3:2)				
IBIST Registers		0	Unchanged				
NCU Debug Trigger Enable		0	Unchanged				
SOC DECR		0	Unchanged				
SOC Error Status		0	Unchanged				
SOC Pending Error Status		0	Unchanged				
SOC SII Error Syndrome		0	Unchanged				
SOC NCU Error Syndrome		0	Unchanged				
Debug Port Configuration		0	Unchanged				

TABLE 13-15 CPU State After Reset and in `RED_state` *(9 of 9)*

Name	Fields	POR	WMR|DBR	WDR	XIR	SIR	RED_State[2]
IO Quiesce Control	niu_stall	0	Unchanged				
	dmu_stall	0	Unchanged				
	Other fields	Unknown	Unchanged				
Serial Number		Unique Value					
EFUSE Status		FFFF FFFF FFFF $FFFF_{16}$					
OVERLAP_MODE		0	Unchanged				
CLK_SCKSEL		0	Unchanged				
SSI_LOG		0_{16}	Unchanged				
MBIST_MODE	Bits{3:0}	0	Unchanged				
MBIST_BYPASS	Bits {47:0}	0	Unchanged				
MBIST_RESULT	Bits {1:0}	0	Unchanged				
MBIST_DONE	Bits {47:0}	0	Unchanged				
MBIST_FAIL	Bits {47:0}	0	Unchanged				
LBIST_MODE	Bits {1:0}	0	Unchanged				
LBIST_BYPASS	Bits {7:0}	0	Unchanged				
LBIST_DONE	Bits {7:0}	0	Unchanged				
DCR	Bits {2:0}	0	Unchanged				
TRIGOUT		0	Unchanged				
CYCLE_COUNTER	Bits {63:0}	0	Unchanged				
CLOCK STOP DELAY counter	Bits{6:0}	0	Unchanged				
MBIST_START		0	Unchanged				
MBIST_ABORT		0	Unchanged				
MBIST_START_WMR		0	Unchanged				
LBIST_START		0	Unchanged				
LOCK TIME		1400_{16}	Unchanged				
PROP TIME		$C00_{16}$	Unchanged				

1. The TSTATE fields are sampled from the relevant registers on WMR. Since these registers are preserved through WMR, the value in the TSTATE fields after a WMR are the pre-WMR values from the relevant registers.
2. This column applies to all other traps that take the processor into `RED_state` (i.e., traps while at *MAXTL* − 1).

13.9 Boot Sequence

A high-level overview of the typical OpenSPARC T2 power-up reset sequence is as follows:

1. On power-up of the system, the system controller asserts PWRON_RST_L and RST asserts all other reset signals. This causes (1) all OpenSPARC T2 internal states to reset, including all control registers and memory refresh state machines, (2) causes IO outputs to reset, and (3) protects the internal tristate muxes. The main CPU PLL will be locking to the default clock ratio during this time. The other PLLs, in the PIU SerDes, also will be locking to their frequencies during this time.
2. Once power is up in the system, the system controller then deasserts PWRON_RST_L and the CPU fetches reset configuration programming code from the boot PROM where configuration registers (clock ratios, etc.) are programmed. RST must deassert all reset signals simultaneously and synchronously to their respective clocks.
3. A second, warm reset caused by writing to wmr_gen bit is used to reset most of OpenSPARC T2 (everything except items reset only by PWRON_RST_L above), relock the CPU PLL, and restart instruction fetch of boot code running at the reprogrammed clock ratio. The main PLL is locking during this time if the ratio is updated.
4. Subsequent warm resets may take place later via writing to wmr_gen bit, which do not disturb states which are reset only by PWRON_RST_L. Warm reset may also be caused by assertion of the PB_RST_L pin.

13.9.1 Assumed POR Software Initialization Sequence

This is the sequence we envision a machine in normal use would follow. During debug, an engineer may wish to forego some steps, such as the warm reset.

Guaranteed by hardware:

- L2 Tag, Data, and VUAD arrays, when BISTed to zeros, are initialized to empty with good parity and good ECC.
- L2 Directory (of L1 tags) is marked invalid on reset.
- L1 I-cache, L1 D-cache, when BISTed to zeros, initialized to good parity

Assumptions:

- Main Memory – Fast ECC initialization with Bzero ASI.

- SPD (Serial Presence Detect) – SPD is launched by software, and can take up to ~1/12 of a second to complete.

Sequence:

1. Service processor asserts TRST_L and PWRON_RST_L.
2. Power ramps up.
3. PLLs start up, and clocks start toggling.
4. Service processor asserts TMS and applies one TCK pulse.
5. Service processor deasserts TRST_L.
6. Service processor issues 5 TCK clock pulses - TMS still asserted. At this point the JTAG logic is reset.

Once TRST_L is deasserted, registers in the JTag portion of the TCU may be accessed via the JTag TAP using TCK while the PLLs in UltraSPARC {N2} have not locked yet. To do this, the user needs to execute TAP_JTPOR_ACCESS after deasserting TRST_L but before deasserting PWRON_RST_L. The status of TCU can then be checked with TAP_JTPOR_STATUS; a status of 1 indicates that the TCU is paused and the JTAG programming window is active. JTAG instructions can be executed during this window. To continue with POR, the user should execute TAP_JTPOR_CLEAR after deasserting PWRON_RST_L, which will cause TCU to continue with the POR sequence.

7. Service processor deasserts PWRON_RST_L. Note that it must deassert PWRON_RST_L after deasserting TRST_L.
8. PLLs lock.
9. The lowest-numbered available virtual processor begins fetching and executing instructions at $RSTV_{ADDR} \mid\mid 20_{16}$. The MMUs are turned off, in bypass mode, with default mapping. At first, only PROM working. Software has to enable everything else.
10. Read RSET_STAT register, which indicates POR.
11. Initialize CLK_DIV register with desired ratios.
12. Write 1 to wmr_gen bit of RESET_GEN, to initiate warm reset.
13. The lowest-numbered available virtual processor begins fetching and executing instructions at $RSTV_{ADDR} \mid\mid 20_{16}$. The MMUs are turned off, in bypass mode, with default mapping. At first, only PROM working. Software has to enable everything else.
14. Read RSET_STAT register, which indicates warm reset, with clock change.

15. Enable error detection on L1 and L2 caches.
16. Enable L1 and L2 caches.
17. Copy bootstrap into L2 cache, using `ASI_BLK_INIT_ST_*`.
18. Branch to bootstrap (now executing from cache).
19. Copy/decode code segments from PROM space to cacheable space, using `ASI_BLK_INIT_ST_*`.
20. Initialize DRAM interface blocks.
21. Force refresh asynchronicity, by zeroing out the refresh counters on each DRAM controller, at precise intervals ($n/4$ of the refresh interval value).
22. Initialize main memory using `ASI_BLK_INIT_ST_*`.
23. Initialize rest of blocks on the chips.
24. Slowly unpark the other enabled virtual processors via the `ASI_CORE_RUNNING_W1S` or `ASI_CORE_RUNNING_RW` registers. Additional virtual processors should be unparked one at a time, with at least 20 microseconds elapsing between successive unpark operations to avoid surges in power consumption.
25. All virtual processors initialize strand-specific state.
26. Strand 0 in each available physical core initializes physical core state (such as enable L1$).
27. Jump into hypervisor.

13.9.2 Assumed Warm Reset Software Initialization Sequence

Assumptions:

- Need to check whether error reset or software-generated reset.

Sequence:

1. Read RSET_STAT register, which indicates warm reset.
2. Check local error logs.
3. If errors, go to crash-dump handling.
4. If no errors, initialize/clear L1 caches.

5. Turn on caches.
6. Initialize strand-specific state.
7. Slowly unpark the other enabled virtual processors via the `ASI_CORE_RUNNING_W1S` or `ASI_CORE_RUNNING_RW` registers. Additional virtual processors should be unparked one at a time, with at least 20 microseconds elapsing between successive unpark operations to avoid surges in power consumption.
8. All virtual processors initialize strand-specific state.
9. Strand 0 in each available physical core initializes physical core state (such as clear & enable L1$).
10. Continue, as desired by SW.
11.

CHAPTER **14**

CMT

OpenSPARC T2 implements the *Sun Microsystems Standard CMT* Programming Model Specification **version 2.2.1** (as incorporated into the CMT chapter of the *UltraSPARC Architecture 2007* specification), with the exception that the `ASI_CMT_ERROR_STEERING` register is not supported. Please refer to that document for details of the operation of the CMT registers listed in this chapter.

14.1 CMT Registers

14.1.1 ASI_CORE_AVAILABLE

All virtual processors share a single `ASI_CORE_AVAILABLE` register at ASI 41_{16}, VA{63:0} = 0_{16}.

TABLE 14-1 defines the format of this register.

TABLE 14-1 Strand Available – `ASI_CORE_AVAILABLE` (ASI 41_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:0	core_avail	FFF FFFF FFFF FFFF$_{16}$[1]	RO	Bits are set to 1 if the virtual processor is available, 0 if unavailable. Each physical core in OpenSPARC T2 (represented by a byte) will either be all 0s or all 1s.

1. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores unavailable may contain 00_{16} bytes in the initial value.

14.1.2 ASI_CORE_ENABLE_STATUS

All virtual processors share a single `ASI_CORE_ENABLE_STATUS` register at ASI 41_{16}, VA{63:0} = 10_{16}.

TABLE 14-2 defines the format of this register.

TABLE 14-2 Strand Enable Status – ASI_CORE_ENABLE_STATUS (ASI 41_{16}, VA 10_{16})

Bit	Field	Initial Value	R/W	Description
63:0	enable_status	FFF FFFF FFFF FFFF$_{16}$[1]	RO	Bits are set to 1 if the virtual processor is enabled, 0 if disabled. Loaded with the contents of ASI_CORE_AVAILABLE upon completion of a power-on reset. This register is loaded with the contents of ASI_CORE_ENABLE upon completion of a warm reset. Each physical core in OpenSPARC T2 (represented by a byte) will either be all 0s or all 1s.

1. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores unavailable may contain 00_{16} bytes in the initial value.

Notes The CMT spec uses CORE_ENABLE_STATUS instead of ASI_CORE_ENABLE_STATUS. ASI_CORE_ENABLE_STATUS is used in this document to avoid confusion between the two very similar register names in the CMT spec.

If ASI_CORE_ENABLE is all zeros, the lowest available physical core will remain enabled.
An interrupt sent to a disabled virtual processor will be ignored by the disabled virtual processor and will not have any effect on the virtual processor sending the interrupt.

14.1.3 ASI_CORE_ENABLE

All virtual processors share a single ASI_CORE_ENABLE register at ASI 41_{16}, VA{63:0} = 20_{16}.

TABLE 14-3 defines the format of this register.

TABLE 14-3 Strand Enable – ASI_CORE_ENABLE (ASI 41_{16}, VA 20_{16})

Bit	Field	Initial Value	R/W	Description
63:0	enable	FFF FFFF FFFF FFFF$_{16}$[1]	RW	Set bit to 1 to enable the virtual processor on the next warm reset. Set bit to 0 to disable the virtual processor on the next warm reset. Loaded with the contents of ASI_CORE_AVAILABLE upon completion of a power-on reset. Each physical core in OpenSPARC T2 (represented by a byte) will either be all zeros or all ones. When written to, if any bit within a byte is a zero, the whole byte will be set to 00_{16}.

1. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores unavailable may contain 00_{16} bytes in the initial value.

14.1.4 ASI_XIR_STEERING

All virtual processors share a single ASI_XIR_STEERING register at ASI 41_{16}, VA{63:0} = 30_{16}.

TABLE 14-4 defines the format of this register.

TABLE 14-4 XIR Steering – ASI_XIR_STEERING (ASI 41_{16}, VA 30_{16})

Bit	Field	Initial Value	R/W	Description
63:0	xirsteering	FFF FFFF FFFF FFFF$_{16}$[1]	RW	Bits are set to 1 if the virtual processor will receive an XIR when the external XIR pin is asserted. Loaded with the contents of ASI_CORE_AVAILABLE upon completion of a power-on reset. This register is loaded with the contents of ASI_CORE_ENABLE upon completion of a warm reset.

1. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores unavailable may contain 00_{16} bytes in the initial value.

Note | OpenSPARC T2 allows the ASI_XIR_STEERING register to be set to 0_{16}. When set to 0_{16}, assertion of the external XIR pin will have no effect.

14.1.5 ASI_CMT_TICK_ENABLE

All virtual processors share a single ASI_CMT_TICK_ENABLE register at ASI 41_{16}, VA{63:0} = 38_{16}. This register is preserved across warm reset.

TABLE 14-5 defines the format of this register.

TABLE 14-5 Tick Enable – ASI_CMT_TICK_ENABLE (ASI 41_{16}, VA 38_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	0_{16}	RO	*Reserved*
0	tick_enable	0_{16}	RW	Set to 1 to enable incrementing of TICK register in all available, enabled physical cores

The ASI_CMT_TICK_ENABLE register synchronizes the tick registers in all physical cores of N2. Each physical core contains one tick register. A physical core's TICK register increments only when it is 1) available, 2) enabled, and 3) ASI_CMT_TICK_ENABLE is set to 1. The output of the ASI_CMT_TICK_ENABLE register is distributed synchronously and with the same delay to all physical cores. Thus, when ASI_CMT_TICK_ENABLE changes, all available, enabled physical cores start or stop incrementing their tick register at the same system clock cycle.

Programming Note Hyperprivileged software can synchronize the tick registers across all available, enabled physical cores as follows. First, one strand writes 0 to `ASI_CMT_TICK_ENABLE`. Then it initializes a mutex-protected counter (`c`) to the number of available, enabled physical cores. Then, one strand on each available, enabled physical core writes the desired tick value to its core's tick register and decrements `c`. Each strand checks the value of the counter. If `c` is 0, that strand writes a 1 to the `ASI_CMT_TICK_ENABLE` register.

14.1.6 ASI_CMT_ERROR_STEERING

OpenSPARC T2 does not implement the `ASI_CMT_ERROR_STEERING` (ASI 41_{16}, VA 40_{16}) register.

14.1.7 ASI_CORE_RUNNING_RW

All virtual processors share a single `ASI_CORE_RUNNING_RW` register at ASI 41_{16}, VA{63:0} = 50_{16}.

TABLE 14-6 defines the format of this register.

TABLE 14-6 Strand Running RW – `ASI_CORE_RUNNING_RW` (ASI 41_{16}, VA 50_{16})

Bit	Field	Initial Value	R/W	Description
63:0	running_rw	1_{16}[1,2]	RW	Bits are set to 0 to park a virtual processor and set to 1 to unpark a virtual processor.

1. Initial value listed is that seen by software. After a power-on reset, the register contains 0_{16} until reset is complete and the initial strand is unparked, and an external agent viewing this register (through the tap controller) may see the zero value.
2. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores enabled will have a single bit corresponding to the lowest enabled virtual processor set.

Notes As per the *CMT Programming Model Specification*, OpenSPARC T2 prevents software from parking all strands and forces one hardware strand (the one performing the parking) to keep running when an attempt is made to park all strands. However, software must allow for a change in `ASI_CORE_RUNNING_RW` to propagate by waiting for the value of `ASI_CORE_RUNNING_STATUS` to match the value written to `ASI_CORE_RUNNING_RW` before making another update, otherwise hardware operation is unpredictable. In particular, a strand or all strands may become parked or unparked and remain unresponsive to further unpark or park commands, until a warm reset or POR is performed.

If a strand parks itself, the strand is guaranteed to not execute any instructions beyond the instruction that parked it (i.e. there is no skid following the parking instruction).

WARNINGS! Software must not attempt to park a strand that is not completely unparked (that is, the strand's bit in `ASI_CORE_RUNNING_STATUS` must be 1 before clearing the strand's bit in `ASI_CORE_RUNNING_RW`). Operation of OpenSPARC T2 is undefined when a strand that is not unparked has its bit in `ASI_CORE_RUNNING_RW` cleared. In particular, the strand may become unparked and remain unresponsive to further park commands, until a warm reset or POR is performed.

Software must not attempt to unpark a strand that is not completely parked (that is, the strand's bit in `ASI_CORE_RUNNING_STATUS` must be 0 before setting the strand's bit in `ASI_CORE_RUNNING_RW`). Operation of OpenSPARC T2 is undefined when a strand that is not parked has its bit in `ASI_CORE_RUNNING_RW` set. In particular, the strand may become parked and remain unresponsive to further unpark commands, until a warm reset or POR is performed.

14.1.8 ASI_CORE_RUNNING_STATUS

All virtual processors share a single `ASI_CORE_RUNNING_STATUS` register at ASI 41_{16}, VA{63:0} = 58_{16}.

TABLE 14-7 defines the format of this register.

TABLE 14-7 Strand Running Status – ASI_CORE_RUNNING_STATUS (ASI 41_{16}, VA 58_{16})

Bit	Field	Initial Value	R/W	Description
63:0	running_status	1_{16}[1,2]	RO	Bits are set to 0 if the virtual processor is currently parked and set to 1 if the virtual processor is currently running.

1. Initial value listed is that seen by software. After a power-on reset, the register contains 0_{16} until reset is complete and the initial strand is unparked, and an external agent viewing this register (through the tap controller) may see the zero value.
2. Initial value listed is for a fully available OpenSPARC T2. An OpenSPARC T2 with some physical cores enabled will have a single bit corresponding to the lowest enabled virtual processor set.

Note | While a virtual processor is parked, interrupt and XIR events targeting the virtual processor will be held pending and will be taken once the virtual processor is unparked.

14.1.9 ASI_CORE_RUNNING_W1S

All virtual processors share a single ASI_CORE_RUNNING_W1S register at ASI 41_{16}, VA{63:0} = 60_{16}.

TABLE 14-8 defines the format of this register.

TABLE 14-8 Strand Running W1S – ASI_CORE_RUNNING_W1S (ASI 41_{16}, VA 60_{16})

Bit	Field	Initial Value	R/W	Description
63:0	running_w1s	0_{16}	WO	Writing 1 to a bit will set the corresponding bit in ASI_CORE_RUNNING_RW. Writing 0 to a bit will leave the corresponding bit in ASI_CORE_RUNNING_RW unchanged.

Software must not attempt to unpark a strand that is not completely parked (that is, the strand's bit in ASI_CORE_RUNNING_STATUS must be 0 before using ASI_CORE_RUNNING_W1S to set the strand's bit in ASI_CORE_RUNNING_RW). Operation of OpenSPARC T2 is undefined when a strand that is not parked has its bit in ASI_CORE_RUNNING_RW set. In particular, the strand may become unparked and remain unresponsive to further park commands, until a warm reset or POR is performed.

14.1.10 ASI_CORE_RUNNING_W1C

All virtual processors share a single ASI_CORE_RUNNING_W1C register at ASI 41_{16}, VA{63:0} = 68_{16}.

TABLE 14-9 defines the format of this register.

TABLE 14-9 Strand Running W1C – `ASI_CORE_RUNNING_W1C` (ASI 41_{16}, VA 68_{16})

Bit	Field	Initial Value	R/W	Description
63:0b	running_w1c	0_{16}	WO	Writing 1 to a bit will clear the corresponding bit in `ASI_CORE_RUNNING_RW`. Writing a zero to a bit will leave the corresponding bit in `ASI_CORE_RUNNING_RW` unchanged.

Notes

As per the *CMT Programming Model Specification*, OpenSPARC T2 prevents software from parking all strands and forces one hardware strand (the one performing the parking) to keep running when an attempt is made to park all strands. However, software must allow for a change in `ASI_CORE_RUNNING_W1C` to propagate by waiting for the value of `ASI_CORE_RUNNING_STATUS` to match the value written to `ASI_CORE_RUNNING_W1C` before making another update, otherwise hardware operation is unpredictable. In particular, the strand may become parked and remain unresponsive to further unpark commands, until a warm reset or POR is performed.

If a strand parks itself, the strand is guaranteed to not execute any instructions beyond the instruction that parked it (i.e. there is no skid following the parking instruction).

Software must not attempt to park a strand that is not completely unparked (that is, the strand's bit in `ASI_CORE_RUNNING_STATUS` must be 1 before using `ASI_CORE_RUNNING_W1C` to clear the strand's bit in `ASI_CORE_RUNNING_RW`). Operation of OpenSPARC T2 is undefined when a strand that is not unparked has its bit in `ASI_CORE_RUNNING_RW` cleared. In particular, the strand may become parked and remain unresponsive to further unpark commands, until a warm reset or POR is performed.

14.2 ASI_CMT_CORE Registers

14.2.1 ASI_CMT_CORE_INTR_ID

Each virtual processor has a read-only `ASI_CMT_CORE_INTR_ID` register at ASI 63_{16}, VA{63:0} = 0_{16}.

TABLE 14-10 defines the format of this register.

TABLE 14-10 Strand Interrupt ID – `ASI_CMT_CORE_INTR_ID` (ASI 63_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15:6	intr_id_hi	0_{16}	RO	Upper bits of Interrupt ID are all 0.
5:0	intr_id_lo	coreid	RO	Matches `ASI_CMT_STRAND_ID` bits 5:0.

14.2.2 `ASI_CMT_STRAND_ID`

Each virtual processor has a read-only `ASI_CMT_STRAND_ID` register at ASI 63_{16}, VA{63:0} = 10_{16}.

TABLE 14-11 defines the format of this register.

TABLE 14-11 Strand ID – `ASI_CMT_STRAND_ID` (ASI 63_{16}, VA 10_{16})

Bit	Field	Initial Value	R/W	Description
63:38	—	0_{16}	RO	*Reserved*
37:32	max_strand_id	7_{16}	RO	Each physical core on OpenSPARC T2 consists of 8 strands.
31:22	—	0_{16}	RO	*Reserved*
21:16	max_strand_id	$3F_{16}$	RO	OpenSPARC T2 contains 64 virtual processors
15:6	—	0_{16}	RO	*Reserved*
5:0	strand_id	coreid	RO	Physical strand ID in 5:3, strand ID in 2:0.

Note The strand ID in OpenSPARC T2 is fixed based on physical core and strand numbers. This implies that an OpenSPARC T2 with unavailable cores will have holes in the strand ID space (for example, if physical core 1 is unavailable, there will be no strand_id 8_{16}–F_{16}).

CHAPTER 15

(Placeholder chapter)

CHAPTER 16

(Placeholder chapter)

CHAPTER 17

(Placeholder chapter)

CHAPTER 18

(Placeholder chapter)

CHAPTER 19

(Placeholder chapter)

CHAPTER 20

(Placeholder chapter)

CHAPTER 21

(Placeholder chapter)

CHAPTER 22

(Placeholder chapter)

CHAPTER 23

(Placeholder chapter)

CHAPTER 24

Noncacheable Unit (NCU) and Boot ROM Interfaces

24.1 Noncacheable Unit (NCU)

The main functions of the NCU are to route PIO accesses from the CMP virtual processors to the I/O subsystem and to vector interrupts from the I/O subsystem to the CMP virtual processors. The NCU provides CSRs for NCU management and mondo interrupt management.

The NCU decodes the I/O physical address space. OpenSPARC T2 supports 40-bit physical addresses, where the MSB (bit 39) is 0 for cacheable accesses (memory system) and 1 for noncacheable accesses (I/O subsystem).

NCU determines the destination of a PIO access by examining the 8 MSB (bit 39:32) of the physical address. All accesses received by NCU have bit 39 of the physical address set to 1. The address range of each IO subsystem block can be found in the following table.

TABLE 24-1 Global Physical Address Assignments

MSB Address Range{39:32}	Assignment
00_{16} ~ $7F_{16}$	Not supported by NCU (memory)
80_{16}	NCU
82_{16}	*Reserved*
83_{16}	CCU
84_{16}	MCUs 13:12 = 00_2 for MCU0, 13:12 = 01_2 for MCU1 13:12 = 10_2 for MCU2, 13:12 = 11_2 for MCU3
85_{16}	TCU (JTAG / TAP unit)
86_{16}	DBG
87_{16}	*Reserved*

TABLE 24-1 Global Physical Address Assignments

MSB Address Range{39:32}	Assignment
88_{16}	DMU
89_{16}	RST
$8A_{16}$ ~ $8F_{16}$	*Reserved*
90_{16}	ASI CPU shared registers (directly accessible only by JTAG/TAP unit)
91_{16} ~ $9F_{16}$	*Reserved*
$A0_{16}$ ~ BF_{16}	Not supported by NCU (L2 control and status registers)
$D0_{16}$ ~ FE_{16}	*Reserved*
FF_{16}	SSI (boot ROM)

24.2 NCU Management Registers

The NCU provides a unique serial number for each OpenSPARC T2 chip. In addition, the NCU contains registers showing the eFuse, Sparc core, and L2 bank status.

24.2.1 Serial Number

The serial number register format is show in TABLE 24-2.

TABLE 24-2 Processor Serial Number – SER_NUM (80 0000 1000_{16})

Bit	Name	Initial Value	R/W	Description
63:60	delta_vdd	X	RO	Delta_Vdd[3] is a sign bit (increase or decrease with respect to nominal). Bits [2:0] specify one of 8 increments.
59:50	delta_t	X	RO	Indicates the temperature offset for the thermal diode, in increments of 20 mV.
49	reserved	0	RO	reserved for testinfo
48:46	fab	X	RO	
45:41	reserved	0	RO	reserved for testinfo
40	bin	X	RO	
39:16	lot	X	RO	
15:10	wafer	X	RO	
9:5	column	X	RO	
4:0	row	X	RO	

24.2.2 eFuse Status

The eFuse Status register format is show in TABLE 24-3.

TABLE 24-3 eFuse Status – EFU_STAT ($80\ 0000\ 1008_{16}$)

Bit	Name	Initial Value	R/W	Description
63:0	efu_status	$FFFF\ FFFF\ FFFF\ FFFF_{16}$	RO	eFuse status programmed by eFuse block

24.2.3 Strand Available

The Strand Available register format is show in TABLE 24-4. This register is an alias for the ASI_CORE_AVAILABLE register described in 14.1.1 ON PAGE 179.

TABLE 24-4 Strand Available – CORE_AVAIL ($80\ 0000\ 1010_{16}$)

Bit	Name	Initial Value	R/W	Description
63:0	avail	$FFFF\ FFFF\ FFFF\ FFFF_{16}$	RO	Strand available programmed by eFuse. Note that all strands within a physical core will be programmed to the same value (all available or all unavailable).

24.2.4 L2 Configuration Control and Status Registers

The NCU contains several L2 Configuration Control and Status registers.

- The L2 Bank Available register is described in Section 28.14.2, *L2 Bank Available*, on page 438.
- The L2 Bank Enable register is described in Section 28.14.3, *L2 Bank Enable*, on page 439.
- The L2 Bank Enable Status Register is described in Section 28.14.4, *L2 Bank Enable Status*, on page 441.
- The L2 Index Hash Enable register is described in Section 28.14.5, *L2 Index Hash Enable*, on page 442.
- The L2 Index Hash Enable Status register is described in Section 28.14.6, *L2 Index Hash Enable Status*, on page 443.

24.2.5 Physical Address Partitioning

The 64-Gbyte region of noncacheable physical memory, from $C0\ 0000\ 0000_{16}$ to $CF\ FFFF\ FFFF_{16}$, is reserved for mapping NCU to access PCIE address space.NCU ASI Registers

Several CMP and Interrupt registers are located in the NCU and made accessible to the JTAG/TAP controller, as listed below. In addition, all ASI registers are made accessible to the JTAG/TAP controller through the addressing shown in TABLE 24-5.

TABLE 24-5 NCU ASI Register Physical Address Map

Bit	Field	Value	Description
39:32	ncu	90_{16}	Identifies NCU space.
31:29	core	0_{16}–7_{16}	Identifies physical core being targeted (set to 0_{16} for a shared ASI register).
28:26	strand	0_{16}–7_{16}	Identifies strand being targeted (set to 0_{16} for a shared ASI register).
25:18	asi	0_{16}–FF_{16}	Identifies ASI being targeted.
17:3	va	0_{16}–$7FFFF_{16}$	Identifies va{17:13} being targeted.
2:0	—	0_{16}	Always zero for 64-bit access.

Note For a shared ASI register, different virtual processor, thread may access the same ASI VA register in NCU. For those registers, NCU ignores PA{31:26} when PA{39:32} = 90_{16}.

24.2.6 Strand Available Register (ASI 41_{16} VA 00_{16})

This register is described in Section 14.1.1, *ASI_CORE_AVAILABLE*, on page 179. It is available to the JTAG/TAP controller at address 90 0104 0000_{16}.

24.2.7 Strand Enable Status Register (ASI 41_{16} VA 10_{16})

This register is described in Section 14.1.2, *ASI_CORE_ENABLE_STATUS*, on page 179. It is available to the JTAG/TAP controller at address 90 0104 0010_{16}.

24.2.8 Strand Enable Register (ASI 41_{16} VA 20_{16})

This register is described in Section 14.1.3, *ASI_CORE_ENABLE*, on page 180. It is available to the JTAG/TAP controller at address 90 0104 0020_{16}.

24.2.9 XIR Steering Register (ASI 41_{16} VA 30_{16})

This register is described in Section 14.1.4, *ASI_XIR_STEERING*, on page 181. It is available to the JTAG/TAP controller at address 90 0104 0030_{16}.

24.2.10 CMP Tick Enable Register (ASI 41_{16} VA 38_{16})

This register is described in Section 14.1.5, *ASI_CMT_TICK_ENABLE*, on page 181. It is available to the JTAG/TAP controller at address 90 0104 0038_{16}.

24.2.11 Strand Running RW Register (ASI 41_{16} VA 50_{16})

This register is described in Section 14.1.7, *ASI_CORE_RUNNING_RW*, on page 182. It is available to the JTAG/TAP controller at address 90 0104 0050_{16}.

24.2.12 Strand Running Status Register (ASI 41_{16} VA 58_{16})

This register is described in Section 14.1.8, *ASI_CORE_RUNNING_STATUS*, on page 183. It is available to the JTAG/TAP controller at address 90 0104 0058_{16}.

24.2.13 Strand Running W1S Register (ASI 41_{16} VA 60_{16})

This register is described in Section 14.1.9, *ASI_CORE_RUNNING_W1S*, on page 184. It is available to the JTAG/TAP controller at address 90 0104 0060_{16}.

24.2.14 Strand Running W1C Register (ASI 41_{16} VA 68_{16})

This register is described in Section 14.1.10, *ASI_CORE_RUNNING_W1C*, on page 184. It is available to the JTAG/TAP controller at address 90 0104 0068_{16}.

24.2.15 SOC Error Steering Register (90 0104 1000_{16})

This register is described in Section 25.23.4, *SOC Error Steering Register*, on page 350. It is available to the JTAG/TAP controller at address 90 0104 1000_{16}.

24.2.16 Warm Reset Vector Mask Register (ASI 45_{16} VA 18_{16})

This register is described in Section 29.1.4, *ASI_RST_VEC_MASK*, on page 454. It is available to the JTAG/TAP controller at address 90 0114 0018_{16}.

24.2.17 Interrupt Vector Dispatch Register (ASI 73_{16} VA 0_{16})

This register is described in Section 7.3.3, *Interrupt Vector Dispatch Register*, on page 55. It is available to the JTAG/TAP controller at address 90 01CC 0000_{16}.

24.3 Boot ROM Address Region

The format of the Boot ROM Range register is defined in TABLE 24-6.

TABLE 24-6 Address Range Definition Boot ROM Range (FF FXXX $XXXX_{16}$)

Bit	Field	Value	Description
39:32	bootspace	FF_{16}	Identifies BOOT ROM space
31:28	bootrom	F_{16}	Identifies BOOT ROM range
27:0	romaddr		Byte address sent to Boot ROM (256 Mbyte available)

Addresses within the Boot ROM address range (FF F000 0000_{16} to FF FFFF $FFFF_{16}$) are issued to the Boot ROM, aliasing the Boot ROM to cover the top 16 Mbytes of the available space. The only transactions that are supported directly to the Boot ROM are

- 1, 2, 4, 8 byte-aligned reads
- 1, 2, 4, 8 byte-aligned writes

Since the Boot ROM is predominantly used for instructions, which are explicitly always big-endian, all accesses to the Boot ROM are treated as big-endian.

24.3.1 Boot ROM Interface Registers

All of the BOOT ROM Interface registers other than SSI Clock Select register are defined elsewhere, specifically in Chapter 25, *Error Handling*.

24.3.1.1 SSI Clock Select Register

The SSI Clock Select register controls the SSI clock frequency as a fraction of the core clock frequency. The format of the SSI Clock Select register is as shown in TABLE 24-7.

TABLE 24-7 SSI Clock Select Register – SSI_CLKSEL ($80\ 0000\ 3040_{16}$)

Bit	Field	Initial Value	R/W	Description
63:2	*Reserved*	0_{16}	RO	*Reserved*
1:0	ssi_scksel	0_{16}	RW	0_{16}: SSI Clock = iol2clk/8 1_{16}: SSI Clock = iol2clk/4 2_{16}: SSI Clock = iol2clk/8 3_{16}: SSI Clock = iol2clk/8 Preserved on warm reset. Value programmed takes effect on next warm reset.

CHAPTER 25

Error Handling

25.1 Error Classes

Errors on OpenSPARC T2 fall into three main classes: fatal (FE), hardware uncorrected (UE), and hardware corrected (CE). Hardware uncorrected errors have two subclasses: software recoverable and software unrecoverable. Hardware corrected errors have two subclasses: hardware corrected and cleared, and hardware corrected but not cleared.

Fatal errors are generated whenever the hardware detects a condition where an error has occurred and the extent to which the error may have propagated is unbounded. An example of a fatal error is an uncorrectable VUAD corruption in the L2 cache; this case implies that global cache coherence has been lost. Since a fatal error may have corrupted key operating system or hypervisor data structures, fatal errors generate an immediate warm reset to the OpenSPARC T2 chip.

Uncorrected errors are errors for which the hardware does not take corrective action. Uncorrected errors fall into two classes: software recoverable and software unrecoverable. For software unrecoverable errors, the extent to which the error may have propagated is tightly bounded. Examples of uncorrected errors include a data parity error on the DTLB, and a double-bit ECC error in the L2 cache data. The data parity error in the DTLB may be recoverable in software by forcing the TLB entry with the bad parity to be invalidated, which will then be reloaded over with a new TTE entry and good parity on the subsequent TLB miss. A double-bit L2 ECC error is unrecoverable as the data cannot be corrected by software, but the extent to which the error could have propagated is limited to the address space of any process that has access to that memory location, and software may be able to keep the system running by killing all processes that could be affected by the error and then scrubbing the bad memory location. Uncorrected errors are reported through several traps, including precise, deferred, and disrupting traps.

Deferred errors consist solely of uncorrectable store buffer errors in the SPARC virtual processor. They cause a nonmaskable, deferred *store_error* trap. NotData errors (NDE) are a specific case of uncorrected errors, where the virtual processor encountering the NotData error is not the first virtual processor to encounter the error.

Corrected errors are errors that are corrected by OpenSPARC T2 in hardware. Corrected errors can either be cleared by hardware or require software to be cleared. Examples of correctable errors that are cleared by hardware are a data parity error in the instruction cache or a tag parity error in the data cache. Corrected errors that are cleared by hardware generate a disrupting *hw_corrected_error* trap. Examples of corrected errors that require clearing by software are a single-bit error in the L2 cache access for an instruction fetch or data fetch. Corrected errors that are not cleared by hardware generate a disrupting *sw_recoverable_error* trap.

25.2 NotData Overview

NotData is used to flag a data value that has an uncorrectable error, after an uncorrectable error trap has been signaled to one and only one virtual processor. Subsequent accesses by virtual processors to NotData also take an uncorrectable error trap, but their Error Status register flags the trap as having come from access to NotData. Support for NotData prevents the problem of multiple virtual processors attempting to fix the same error, and also prevents the problem of a single uncorrectable error appearing as multiple errors as it enters different subsystems or is accessed multiple times.

In OpenSPARC T2, NotData is implemented for items as they enter into the L2 cache only. This handles the most significant sources of uncorrectable errors, namely, UEs in main memory, UEs generated by the processor, or UEs coming in from the I/O subsystem through the SIU. UEs arising in the L2 itself are kept to a very low probability through hardware scrubbing.

On a cache fill, OpenSPARC T2 signals a UE trap to one and only one virtual processor and loads each 16-byte chunk with a UE into the L2 with the NotData indication (inverted check bits on the four 4-byte chunks). Likewise, on a processor request with a UE, each 4-byte chunk of data in error is loaded into the L2 with the NotData indication. The UE trap for processor requests has already been generated by the processor subsystem.

The N2 implementation of NotData protects for the most frequent UEs but does not provide full NotData protection. The following very low probability cases are not protected by NotData and can result in multiple UE traps:

- Writeback of a line with a UE or NotData is written back to main memory with an ECC encoding that is not distinguishable from a normal memory UE. Thus, a refetch of the line from memory will generate a second UE trap.
- A multiple-bit error in the L2 data array. The UE is *not* converted to NotData on an access, scrubber read, etc., because this was too large a change from the OpenSPARC T1 implementation and provided negligible impact on FIT rate. Multiple accesses of the UE before software has a chance to clean up the line will result in multiple UE traps.

25.3 CMP Error Overview

Errors detected in the CMP and memory subsystem are reported in three sets of error registers: Core, L2 cache, and DRAM. Precise and deferred errors are reported in the Core register set in program order. The Core Error Recording Enable register (CERER) controls whether errors are recorded in the virtual processor. The Strand Error Trap Enable register (SETER) controls whether the strand takes a trap as a result of any reported errors. Errors are reported in the L2 cache and DRAM error sets in the order the errors occur. The L2 cache Error Enable register controls whether errors associated with the L2 cache and DRAM are reported back to the initiator (logging of the errors in the L2 cache and DRAM registers is always performed). If the L2 cache error enable bit (ceen or nceen, depending on whether the error is correctable or uncorrectable/NotData) is not set, error information is not reported back to the initiator. For the uncorrectable/NotData case (nceen clear), this means that bad data will be executed/used, and thus the nceen bit is only intended to be cleared during heavily controlled phases of diagnostic operation.

The OpenSPARC T2 core records errors in five error status registers (ESRs). All of the ESRs provide one copy per strand (virtual processor). OpenSPARC T2 records instruction-fetch-related errors in the I-SFSR (Instruction Synchronous Fault Status register) and precise data-access-related errors in the D-SFSR (Data Synchronous Fault Status register) and D-SFAR (Data Synchronous Fault Address register). The D-SFAR is shared by the OpenSPARC T2 implementation between errors and MMU processing to avoid the need for a separate address register for precise error logging. OpenSPARC T2 also has a disrupting ESR (DESR) to log disrupting errors. Finally, the OpenSPARC T2 core detects store buffer errors. Some of these are handled as nonmaskable, deferred traps and are recorded in the deferred ESR (DFESR).

The L2 error registers can log detailed information for a single error. A dedicated error register is provided for NotData errors, while correctable, uncorrectable, and fatal errors share a single register. The L2 error registers have bits to indicate if multiple errors have occurred.

The DRAM error registers can log detailed information for a single error. A single error register is provided for the two different error classes: uncorrectable and correctable. The DRAM error registers have bits to indicate if multiple errors have occurred.

For the L2 and DRAM registers, fatal and uncorrectable errors overwrite earlier correctable error information. The error registers have bits to indicate if multiple errors have occurred. There are two bits for multiple errors: meu (multiple uncorrectable errors) and mec (multiple correctable errors). TABLE 25-1 lists the multiple error logging and overwrite behavior of the error registers (a CE for the main logged error with the meu bit set is not possible due to FE and UE being higher priority than CE in logging).

TABLE 25-1 Multiple Error Logging in the L2 and DRAM Registers

Main Logged Error	meu	mec	Description
FE	0	0	Single FE encountered and logged.
FE	0	1	Single FE encountered and logged. One or more correctable errors encountered but details on the correctable errors not logged.
FE	1	0	Single FE encountered and logged. One or more uncorrectable errors encountered but details on the uncorrectable errors not logged.
FE	1	1	Single FE encountered and logged. One or more uncorrectable errors encountered but details on the uncorrectable errors not logged. One or more correctable errors encountered but details on the correctable errors not logged.
UE	0	0	Single UE encountered and logged.
UE	0	1	Single UE encountered and logged. One or more correctable errors encountered but details on the correctable errors not logged.
UE	1	0	Two or more UEs encountered, details on the first logged.
UE	1	1	Two or more UEs encountered, details on the first logged. One or more corrected errors encountered but details on the corrected errors not logged.
CE	0	0	Single CE encountered and logged.
CE	0	1	Two or more CEs encountered; details on the first logged.

25.4 Error Trap Vectors

The following table describes the trap vectors used in OpenSPARC T2 to handle hardware errors.

TABLE 25-2 OpenSPARC T2 Error Traps

Trap Vector	Trap Type	Trap Class	N2-specific	Trap Priority Level	Remarks
power_on_reset (warm reset)	01_{16}	Reset	N	0	Used for fatal errors. Error state captured in L2 ESR/EAR.
store_error	07_{16}	Deferred	Y	2.1	Used for store buffer errors. Error state captured in DFESR.
instruction_access_error	$0A_{16}$	Precise	N	4	Used for instruction access errors. Error state captured in ISFSR or L2 ESR/EAR/NDER.
internal_processor_error	29_{16}	Precise	N	8.2	Used for all IRF/FRF errors (IRFU/IRFC/FRFU/FRFC) except those on the second or subsequent passes of a multicycle operation (partial store, compare and swap, block store). Used for store buffer bypass errors (SBDLU/SBDLC) and register array errors (MRAU/TSAU/TSAC/SCAU/SCAC/tcup/TCCP). Error state captured in DSFSR/DSFAR.
internal_processor_error	29_{16}	Precise	N	12.10	Used for IRF/FRF errors (IRFU/IRFC/FRFU/FRFC) on the second or subsequent passes of a multicycle operation (partial store, compare and swap, block store). Error state captured in DSFSR/DSFAR.
data_access_error	32_{16}	Precise	N	12.9	Used for data access errors. Error state captured in DSFSR or L2 ESR/EAR/NDER.
sw_recoverable_error	40_{16}	Disrupting	N	33.1	Used for disrupting errors not corrected and not cleared by hardware. Error state captured in DESR or L2 ESR/EAR/NDER.
hw_corrected_error	63_{16}	Disrupting	N	33.2	Used for disrupting errors corrected and cleared by hardware. Error state captured in DESR or L2 ESR/EAR/NDER.
instruction_access_MMU_error	71_{16}	Precise	N	2.7	Used for IMMU errors. Error state captured in ISFSR or L2 ESR/EAR/NDER.
data_access_MMU_error	72_{16}	Precise	N	12.2	Used for DMMU errors. Error state captured in DSFSR/DSFAR or L2 ESR/EAR/NDER.

To conform to the latest RAS specification, OpenSPARC T2 uses a different trap vector for each type of hardware error (disrupting, precise, or deferred). There is no unique trap vector for NotData errors. Instead NotData is indicated by an I-SFSR, D-SFSR, or DESR encoding.

25.5 Error Barrier

A MEMBAR #Sync acts as an error barrier on OpenSPARC T2. Before a MEMBAR #Sync completes, all previous, enabled precise and deferred error traps will be taken by the strand. Disrupting errors related to the instruction execution stream also treat MEMBAR #Sync as a memory barrier (with a possible skid of one instruction as described below). Disrupting errors unrelated to the instruction stream do not have a memory barrier operation on OpenSPARC T2.

The following lists the behavior for errors that treat MEMBAR #Sync as an error barrier:

- Any precise error that occurs (and is not masked) will cause a trap before the MEMBAR #Sync executes, and TPC will be the PC of the instruction with the precise error.
- For deferred errors, if the error is detected before the MEMBAR #Sync executes, the trap will be taken before the MEMBAR #Sync executes, and the TPC will be the PC of the MEMBAR #Sync or the PC of some earlier instruction. If the deferred error is detected during the execution of the MEMBAR #Sync (a MEMBAR #Sync causes the store buffer to drain), then the trap will be taken after execution of the MEMBAR #Sync, and the TPC will be the NPC of the MEMBAR #Sync.
- The disrupting errors listed below are related to the instruction stream. If the error is detected before the MEMBAR #Sync executes and the trap is not masked, the trap will be taken either before the MEMBAR #Sync executes, or on the first instruction after the MEMBAR #Sync. The TPC will be the PC of the MEMBAR #Sync, the PC of some earlier instruction, or the PC of the instruction after the MEMBAR #Sync. If the error is detected during the execution of the MEMBAR #Sync and the trap is not masked, then the trap will be taken after executing the instruction following the MEMBAR #Sync, and the TPC will be the NPC of the instruction following the MEMBAR #Sync.
  - Instruction Cache Valid bit Parity
  - Instruction Cache Tag Parity
  - Instruction Cache Tag Multiple
  - Instruction Cache Data Parity
  - Data Cache Valid bit Parity
  - Data Cache Tag Parity
  - Data Cache Tag Multiple

- Data Cache Data Parity
- Store Buffer Data PCX read Correctable ECC
- Store Buffer Data PCX read Uncorrectable ECC
- IT L2 Correctable
- IC L2 Correctable
- DT L2 Correctable
- DC L2 Correctable

The following disrupting errors are not related to the instruction stream. So MEMBAR #Sync has no relationship to these errors and therefore doesn't act as a barrier for these errors. If the error is detected before the MEMBAR #Sync executes and the trap is not masked, the trap will be taken either before the MEMBAR #Sync executes or on the first instruction after the MEMBAR #Sync. The TPC will be the PC of the MEMBAR #Sync, the PC of some earlier instruction, or the PC of the instruction after the MEMBAR #Sync. If the error is detected during the execution of the MEMBAR #Sync and the trap is not masked, then the trap will be taken after executing the instruction following the MEMBAR #Sync, and the TPC will be the NPC of the instruction following the MEMBAR #Sync. If the error is detected after the MEMBAR #Sync completes, the trap (if not masked) will be taken on some later instruction.

- Tick_compare correctable disrupting
- Tick_compare uncorrectable disrupting
- L2 correctable ECC error
- L2 uncorrectable ECC error
- L2 NotData error
- SOC correctable
- SOC uncorrectable

25.6 Virtual Processor Error Handling Overview

The core RAS architecture has two objectives:

1. Provide a FIT rate sufficient for the target market at minimal core cost.
2. Conform to the architecture described by the SPARC SWG RAS & Error Handling Working Group, with implementation suggestions from UltraSPARC Architecture 2007 specification.

To meet both these objectives, OpenSPARC T2 generally does not correct or retry correctable errors detected by hardware. Any hardware-detected error that can be attributed to an instruction's execution results in a precise trap if enabled. Software at the trap handler diagnoses the error. If the error is correctable, it attempts

recovery, for example by correcting a single-bit error in a register file. Software then retries the instruction. If another failure occurs, software can determine the appropriate action to take. This approach minimizes the hardware dedicated to error-specific handling flows in OpenSPARC T2. Software is provided with sufficient information to enable error recovery.

25.6.1 Error Status Registers

The OpenSPARC T2 core records errors in five per-strand error status registers (ESRs): I-SFSR (Instruction Synchronous Fault Status register), D-SFSR (Data Synchronous Fault Status register), D-SFAR (Data Synchronous Fault Address register), DESR (disrupting ESR), and DFESR (deferred ESR).

OpenSPARC T2 cores detect errors as they occur but pipeline precise error indications along with the instruction until commit time. At this time a precise error is recorded in either the I-SFSR or D-SFSR and D-SFAR. The error is only recorded if the appropriate CERER bit is set. In addition, for certain precise errors, the SETER.pscce bit must also be set for the error to be recorded.

Disrupting errors are recorded only if the appropriate CERER bit is set.

Deferred errors are always recorded.

25.6.2 Error Summary

TABLE 25-3 lists the errors the OpenSPARC T2 core detects and also lists salient information about each error. Columns in TABLE 25-3 are explained in the following table.

Column	Terminology
Error Type	Specifies the type of error, as follows: **CE:** hardware-corrected error **UEr:** hardware-uncorrected error that is software recoverable **UEU**: hardware-uncorrected error that is software unrecoverable (**Note:** Software unrecoverable does not imply that the error is fatal).
Trap Type	Specifies the type of trap that is caused, as follows: **P:** precise—logged in either the I-SFSR or the D-SFSR **D:** disrupting—logged in the DESR **Df:** deferred error—results in a *store_error* trap to the virtual processor that detected the error.
Trap Vector	Describes the trap vector to which the processor will be directed when an error occurs. **IAME:** *instruction_access_MMU_error* **IA:** *instruction_access_error* **IPE:** *internal_processor_error* **DAM:** *data_access_MMU_error* **DAE:** *data_access_error* **HCE:** *hw_corrected_error* **SRE:** *sw_recoverable_error* **SE:** *store_error.* OpenSPARC T2 directs disrupting traps and precise traps to different vectors. Software can then inspect the I-SFSR, D-SFSR, or DESR to determine more information regarding the failure.
Maskable	Defines which errors can be masked by SETER.pscce{62}, de{61} or dhcce{60} bits, as follows: **N:** Errors are not maskable by SETER bits. **Number** (62, 61, or 60)**:** Specifies which bit of SETER masks the error trap.

TABLE 25-3 OpenSPARC T2 Processor Error Types *(1 of 5)*

Error	Access Type or Unit	Error Type	Trap Type	Trap Vector	Maskable	ESR ErrorAddr Info	HW Corrected	HW Cleared	Notes
IT tag multiple hit (ITTM)	IFetch	UER	P	IAME	N	ISFSR.ittm	N	N	HW does not invalidate entry; SW should demap all; cannot distinguish between same or different contexts.
ITLB tag parity (ITTP)	IFetch	UER	P	IAME	N	ISFSR.ittp	N	N	HW does not invalidate entry; SW should log tag and data for all entries and demap page.
ITLB data parity (ITDP)	IFetch	UER	P	IAME	N	ISFSR.itdp	N	N	HW does not invalidate entry; SW should demap page.

TABLE 25-3 OpenSPARC T2 Processor Error Types *(2 of 5)*

Error	Access Type or Unit	Error Type	Trap Type	Trap Vector	Maskable	ESR ErrorAddr Info	HW Corrected	HW Cleared	Notes
ITLB MRA uncorrectable (ITMU)	IFetch	UER	P	IAME	N	MRA index{2:0}	N	N	SW should correct MRA by reloading.
ITLB L2 correctable (ITL2C)	IFetch	UER	D	SRE	61	Recorded in L2 ESR	Y	N	SW should correct L2 error.
ITLB L2 uncorrectable (ITL2U)	IFetch	UEU	P	IAME	N	Recorded in L2 ESR	N	N	SW should correct L2 error if possible.
ITLB L2 NotData (ITL2ND)	IFetch	UEU	P	IAME	N	Recorded in L2 ESR	N	N	NotData should have been preceded by a UE.
Icache valid bit (ICVP)	IFetch	CE	D	HCE	60	Index{5:0}, way{2:0}	Y	Y	HW invalidates all ways of I$ index; HW refetches.
Ic tag multiple hit (ICTM)	IFetch	CE	D	HCE	60	Index{5:0}, way{2:0}	Y	Y	HW invalidates all ways of I$ index; HW refetches.
Icache tag parity (ICTP)	IFetch	CE	D	HCE	60	Index{5:0}, way{2:0}	Y	Y	HW invalidates all ways of I$ index; HW refetches.
ICache data parity (ICDP)	IFetch	CE	D	HCE	60	Index{5:0}	Y	Y	HW invalidates all ways of I$ index; HW refetches.
Icache L2 correctable (ICL2C)	IFetch	UER	D	SRE	61	Recorded in L2 ESR	Y	N	SW should correct L2 error.
Icache L2 uncorrectable (ICL2U)	IFetch	UEU	P	IAE	62	Recorded in L2 ESR	N	N	SW should correct L2 error if possible.
Icache L2 NotData (ICL2ND)	IFetch	UEU	P	IAE	62	Recorded in L2 ESR	N	N	NotData should have been preceded by a UE.
IRF correctable ECC error (IRFC)	EXU	UER	P	IPE	62	GL{1:0}, Index{4:0}, ECC Syndrome{7:0}	N	N	SW should correct and retry.
IRF uncorrectable ECC error (IRFU)	EXU	UEU	P	IPE	62	GL{1:0}, Index{4:0}, ECC Syndrome{7:0}	N	N	
FRF correctable ECC error (FRFC)	FGU	UER	P	IPE	62	Index{5:0}, (2) ECC Syndrome{6:0}	N	N	SW should correct and retry.

TABLE 25-3 OpenSPARC T2 Processor Error Types *(3 of 5)*

Error	Access Type or Unit	Error Type	Trap Type	Trap Vector	Maskable	ESR ErrorAddr Info	HW Corrected	HW Cleared	Notes
FRF uncorrectable ECC error (FRFU)	FGU	UEU	P	IPE	62	Index{5:0}, (2) ECC Syndrome{6:0}	N	N	
DTLB tag parity (DTTP)	Load	UER	P	DAME	N	VA{47:0} in D-SFAR	N	N	HW does not invalidate entry; SW should demap page
DT tag multiple hit (DTTM)	Load	UER	P	DAME	N	VA{47:0} in D-SFAR	N	N	HW does not invalidate all entries, SW should demap all; can not distinguish between same or different contexts
DTLB data parity (DTDP)	Load	UER	P	DAME	N	VA{47:0} in D-SFAR	N	N	HW does not invalidate entry; SW should demap page
DTLB MRA uncorrectable (DTMU)	Load	UER	P	DAME	N	MRA index{2:0}	N	N	SW can reload MRA
DTLB L2 correctable (DTL2C)	Load	UER	D	SRE	61	Recorded in L2 ESR	Y	N	SW should correct L2 error
DTLB L2 uncorrectable (DTL2U)	Load	UEU	P	DAME	N	Recorded in L2 ESR	N	N	SW should correct L2 error if possible
DTLB L2 NotData (DTL2ND)	Load	UEU	P	DAME	N	Recorded in L2 ESR	N	N	NotData should have been preceded by a UE
Dcache valid bit (DCVP)	Load	CE	D	HCE	60	Index{6:0}, way{1:0}	Y	Y	HW invalidates all ways of D$ index; HW refetches data; if trap taken, SW can log
Dcache tag parity (DCTP)	Load	CE	D	HCE	60	Index{6:0}, way{1:0}	Y	Y	HW invalidates all ways of D$ index; HW refetches data; if trap taken, SW can log
Dcache tag multiple hit (DCTM)	Load	CE	D	HCE	60	Index{6:0}, way{1:0}	Y	Y	HW invalidates all ways of D$ index; HW refetches data; if trap taken, SW can log
Dcache data parity (DCDP)	Load	CE	D	HCE	60	Index{6:0}, way{1:0}	Y	Y	HW invalidates all ways of D$ index; HW refetches data; if trap taken, SW can log

TABLE 25-3 OpenSPARC T2 Processor Error Types *(4 of 5)*

Error	Access Type or Unit	Error Type	Trap Type	Trap Vector	Mask able	ESR ErrorAddr Info	HW Cor-rected	HW Cleared	Notes
Dcache L2 Correctable (DCL2C)	Load	UER	D	SRE	61	Recorded in L2 ESR	Y	N	SW should correct L2 error.
Dcache L2 Uncorrectable (DCL2U)	Load	UEU	P	DAE	62	Recorded in L2 ESR	N	N	SW should correct L2 error if possible.
Dcache L2 NotData (DCL2ND)	Load	UEU	P	DAE	62	Recorded in L2 ESR	N	N	NotData should have been preceded by a UE.
Store buffer data load hit correctable ECC (SBDLC)	RAW (Read-After-Write)	UER	P	IPE	62	STB index{2:0}	N	N	SW should MEMBAR #Sync, then retry.
Store buffer data load hit uncorrectable ECC (SBDLU)	RAW	UEU	P	IPE	62	STB index{2:0}	N	N	SW should MEMBAR #Sync, then retry (likely a nonrecoverable error).
Store buffer data PC*X* read or ASI store correctable (SBDPC)	PCX	CE	D	HCE	60	STB index{2:0}	Y	Y	SW to log.
Store buffer data PC*X* read (SBDPU)	PCX	UEU	D	SRE	61	STB index{2:0}	N	N	HW generates NotData.
Store buffer address PC*X* read or ASI store parity (SBAPP)	PCX	UEU	DF	SE	N	DFESR bit 61, privilege level, STB index{2:0}	N	N	Nonmaskable, causes *store_error* trap.
Store buffer data I/O read and ASI store uncorrectable (SBDIOU)	PCX	UEU	DF	SE	N	DFESR bit 60, privilege level, STB index{2:0}	N	N	Nonmaskable, causes *store_error* trap.
TSA correctable (TSAC)	TLU	UER	P	IPE	62	TSA index{2:0}, syndrome	N	N	SW can correct or reload TSA.
TSA uncorrectable (TSAU)	TLU	UER	P	IPE	62	TSA index{2:0}, syndrome	N	N	SW can correct or reload TSA.

TABLE 25-3 OpenSPARC T2 Processor Error Types *(5 of 5)*

Error	Access Type or Unit	Error Type	Trap Type	Trap Vector	Maskable	ESR ErrorAddr Info	HW Corrected	HW Cleared	Notes
MRA uncorrectable (MRAU)	MMU	UER	P	IPE (ASI read or read-modify-write)	62	MRA index{2:0}	N	N	SW can correct or reload MRA.
SCA correctable (SCAC)	MMU	UER	P	IPE	62	SCA index{2:0}, syndrome	N	N	SW can correct SCA.
SCA uncorrectable (SCAU)	MMU	UEU	P	IPE	62	SCA index{2:0}, syndrome	N	N	
TICK_CMPR correctable precise (TCCP)	TLU	UER	P	IPE	62	TCA index{1:0}, syndrome	N	N	SW can correct or reload TICK_CMPR.
TICK_CMPR correctable disrupting (TCCD)	TLU	UER	D	SRE	61	TCA index{1:0}, syndrome	N	N	SW can correct or reload TICK_CMPR.
TICK_CMPR uncorrectable precise (TCUP)	TLU	UER	P	IPE	62	TCA index{1:0}, syndrome	N	N	SW can reload TICK_CMPR.
TICK_CMPR uncorrectable disrupting (TCUD)	TLU	UER	D	SRE	61	TCA index{1:0}, syndrome	N	N	SW can reload TICK_CMPR.
L2C	L2	CE	D	HCE	61	Recorded in L2 ESR	Y	N	See Section 25.9.
L2U	L2	UEU	D	SRE	61	Recorded in L2 ESR	N	N	See Section 25.9.
L2ND	L2	UEU	D	SRE	61	Recorded in L2 ESR	N	N	See Section 25.9.
SOC Correctable (SOCC)	SOC	CE	D	HCE	61	Recorded in SOC Error Status Register	Y	Y	Software must read SOC ESRs to determine details of the error.
SOC Uncorrectable (SOCU)	SOC	UEU	D	SRE	61	Recorded in SOC Error Status Register	N	N	Software must read SOC ESRs to determine details of the error.

25.7 SPARC Error Descriptions

25.7.1 ITLB Errors

The priority of ITLB errors is ITTM → ITTP → ITDP → ITMU.

25.7.1.1 ITLB Tag Multiple Hit Error (ITTM)

The OpenSPARC T2 ITLB checks for multiple tag hits on each access if CERER.ittm is set. A multiple tag hit error has higher priority than a tag parity error, and parity is not factored into the tag hit determination. A multiple hit can occur as a result of a hardware failure or a software error. Each ITLB tag entry is a tuple consisting of partition ID, real address indicator, context, and VA, adjusted for page size. A hardware error in any of these fields can cause a multiple tag hit. A multiple hit can also occur if software maps the same virtual address using different contexts[1], loads the ITLB with those mappings simultaneously, and sets each context register to point to one of the two contexts. A multiple hit can also occur if software loads pages with differing pages sizes that do not cause the first page to be autodemapped1.

When a multiple hit error is detected, hardware records the error in the I-SFSR, and a precise *instruction_access_MMU_error* trap is taken. The VA of the instruction fetch is recorded in TPC[TL].

Programming Notes

To handle an ITTM error, software at the trap handler logs the error, and issues either a demap_all or a sequence of demap_pages to all active contexts. Then software issues a RETRY instruction. OpenSPARC T2 refetches the instruction and reaccesses the ITLB. This time either a hit will occur (if the translation was reloaded by another strand) or a miss will occur. If an ITLB miss occurs, hardware retranslates the address and reloads the ITLB.

OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

1. The autodemap feature of the TLB, described in *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142, prevents translations with identical page sizes, VA, and context from existing in the TLB simultaneously. However, software can generate multiple matches by inserting overlapping translations of differing page sizes, or by inserting translations that differ only in context, and then programming the context 0 and context 1 registers to match the pair of translations.

25.7.1.2 ITLB Tag Parity Error (ITTP)

Each ITLB tag entry is protected with parity. The tag entry bits covered by parity are listed in *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142.

ITLB tag parity is checked with each instruction translation if CERER.ittp is set. ITLB tag parity is not checked on loads to ASI_ITLB_TAG_READ_REG. When a parity error is detected, the error is logged in the I-SFSR and the strand takes a precise *instruction_access_MMU_error* trap. The VA of the instruction fetch is recorded in TPC[TL]. Software at the trap handler logs the error and issues a demap_page to the TPC[TL]. Then software issues a RETRY instruction. OpenSPARC T2 refetches the instruction and reaccesses the ITLB. This time either a hit will occur (if the translation was reloaded by another strand), or a miss will occur. If an ITLB miss occurs, hardware retranslates the address and reloads the ITLB.

Implementation Notes

Parity is only checked for translations that hit in the TLB. This implies that a page that would otherwise hit in the TLB may experience a TLB miss instead (for example., if a VA bit has flipped). Normal replacement will eventually remove such a TLB entry, or a page that would otherwise miss in the TLB may experience an ITTP error instead, due to a VA bit flip.

If a parity error occurs in the page size stored in the TTE data, a TLB tag parity error may result. This can occur since an error in the page size will compute incorrect parity for the tag entry. In this case, both tag and data parity errors are detected, but since tag parity errors have higher priority than data parity errors, only the tag parity error exception is taken and recorded.

Programming Note

OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

25.7.1.3 ITLB Data Parity Error (ITDP)

Each ITLB data entry is protected with parity. The data entry bits covered by parity are listed in *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142.

ITLB data parity is checked with each instruction translation if CERER.itdp is set. ITLB data parity is not checked on loads to ASI_ITLB_DATA_ACCESS_REG. When a parity error is detected, hardware records the error in the I-SFSR, and takes a precise *instruction_access_MMU_error* trap. The VA of the instruction fetch is recorded in TPC[TL]. Software at the trap handler logs the error and issues a

demap_page to the TPC[TL]. Then software issues a RETRY instruction. OpenSPARC T2 refetches the instruction and reaccesses the ITLB. This time either a hit will occur (if the translation was reloaded by another strand) or a miss will occur. If an ITLB miss occurs, hardware retranslates the address and reloads the ITLB.

Programming Note OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

25.7.1.4 ITLB MMU Register Array Uncorrectable Error (ITMU)

The MMU register array (MRA) is protected by parity. When hardware tablewalk is enabled, MRA parity is checked for all translations that do not have an entry present in the ITLB during the ITLB reload process. If CERER.hwtwmu is set and a parity error is encountered, hardware records the error in the I-SFSR and takes a precise *instruction_access_MMU_error* trap. The VA of the instruction fetch is recorded in TPC[TL], while the failing MRA index is recorded in the D-SFAR. To recover from an ITMU error, software can reload the MRA register with a "clean" copy of the MRA data which it has stored elsewhere.

25.7.2 DTLB Errors

The priority of DTLB errors is DTTM → DTTP → DTDP → DTMU.

25.7.2.1 DTLB Tag Multiple Hit Error (DTTM)

The OpenSPARC T2 DTLB checks for multiple tag hits on each access (including prefetch instructions) if CERER.dttm is set. A multiple tag hit error has higher priority than a tag parity error, and parity is not factored into the tag hit determination. A multiple hit can occur as a result of a hardware failure or a software error. Each DTLB tag entry is a tuple consisting of partition ID, real address indicator, context, and VA, adjusted for page size. A hardware error in any of these fields can cause a multiple tag hit. A multiple hit can also occur if software maps the same virtual address using different contexts,[1] loads the DTLB with those mappings simultaneously, and sets each context register to point to one of the two contexts. A multiple hit can also occur if software loads pages with differing pages sizes that do not cause the first page to be autodemapped.[1]

1. The autodemap feature of the TLB, described in Section 12.10.15, prevents translations with identical page sizes, VA, and context from existing in the TLB simultaneously. However, software can generate multiple matches by inserting overlapping translations of differing page sizes, or by inserting translations that differ only in context and then programming the context 0 and context 1 registers to match the pair of translations.

When a multiple hit error is detected, hardware records the error in the D-SFSR, and takes a precise *data_access_MMU_error* trap. The VA of the data access is recorded in D-SFAR.

Programming Notes

To handle a DTTM error, software at the trap handler logs the error, and issues either a demap_all or a sequence of demap_pages to all active contexts. Then software issues a retry instruction. OpenSPARC T2 refetches the instruction and reaccesses the DTLB. This time either a hit will occur (if the translation was reloaded by another strand) or a miss will occur. If a DTLB miss occurs, hardware retranslates the address and reloads the DTLB.

OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

25.7.2.2 DTLB Tag Parity Error (DTTP)

Each DTLB tag entry is protected with parity. The tag entry bits covered by parity are listed in *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142.

DTLB tag parity is checked with each data translation (including prefetch instructions) if CERER.dttp is set. DTLB tag parity is not checked on loads to `ASI_DTLB_TAG_READ_REG`. When a parity error is detected, hardware records the error in the D-SFSR and takes a precise *data_access_MMU_error* trap. The VA of the data access is recorded in the D-SFAR. Software at the trap handler logs the error and issues a demap_page to the D-SFAR. Then software issues a RETRY instruction. OpenSPARC T2 refetches the instruction and reaccesses the DTLB. This time either a hit will occur (if the translation was reloaded by another strand) or a miss will occur. If a DTLB miss occurs, hardware retranslates the address and reloads the DTLB.

Implementation Notes

Parity is only checked for data translations that hit in the TLB. This implies that a page that would otherwise hit in the TLB may experience a TLB miss instead (for example, if a VA bit has flipped). Normal replacement will eventually remove such a TLB entry, or a page that would otherwise miss in the TLB may experience a DTTP error instead due to a VA bit flip.

If a parity error occurs in the page size stored in the TTE data, a TLB tag parity error may result. This can occur since an error in the page size will compute incorrect parity for the tag entry. In this case, both tag and data parity errors are detected, but since tag parity errors have higher priority than data parity errors, only the tag parity error exception is taken and recorded.

Programming Note

OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

25.7.2.3 DTLB Data Parity Error (DTDP)

Each DTLB data entry is protected with parity. The data entry bits covered by parity are listed in *I-/D-TLB Data-In/Data-Access/Tag-Read Registers* on page 142.

DTLB data parity is checked with each data translation (including prefetch instructions) if CERER.dtdp is set. DTLB data parity is not checked on loads to ASI_DTLB_DATA_ACCESS_REG. When a parity error is detected, hardware records the error in the D-SFSR, and takes a precise *data_access_MMU_error* trap. The VA of the data access is recorded in D-SFAR. Software at the trap handler logs the error, and issues a demap_page to the D-SFAR. Then software issues a retry instruction. OpenSPARC T2 refetches the instruction and reaccesses the DTLB. This time either a hit will occur (if the translation was reloaded by another strand), or a miss will occur. If a DTLB miss occurs, hardware retranslates the address and reloads the DTLB.

Programming Note

OpenSPARC T2 provides no special hardware to deal with permanent single-bit failures in a TLB entry. To work around a permanent single-bit failure, software must identify the bad TLB entry, then disable hardware tablewalking and manage the TLB in software to avoid the bad TLB entry.

25.7.2.4 DTLB MMU Register Array Uncorrectable Error (DTMU)

The MMU register array (MRA) is protected by parity. When hardware tablewalk is enabled, MRA parity is checked for all translations that do not have an entry present in the DTLB during the DTLB reload process. If CERER.hwtwmu is set and a parity error is encountered, hardware records the error in the D-SFSR, and takes a precise *data_access_MMU_error* trap. The failing MRA index is recorded in the D-SFAR. To recover from an DTMU error, software can reload the MRA register with a "clean" copy of the MRA data which it has stored elsewhere.

25.7.3 Icache Errors

The priority of Icache errors is ICVP > ICTP > ICTM > ICDP.

25.7.3.1 Icache Valid Parity Error (ICVP)

The Icache tag valid bits are protected by duplication: there is a *master* copy and a *slave* copy. The valid bits are checked for each instruction cache access for all ways in the accessed set if CERER.icvp is set.

If CERER.icvp is not set, the *master* copy of the valid bit simply determines whether a cache hit or miss occurred. If a valid bit mismatch occurs and CERER.icvp is set, hardware invalidates all ways in the accessed Icache set.

If the DESR.f bit is clear, hardware records the index and the way[1] with the error in the DESR ErrorAddress field, and sets the DESR.f and DESR.icvp bits. If the DESR.f bit is set, hardware sets the DESR.me bit but does not record the index and way in the DESR.

A disrupting *hw_corrected_error* trap is generated to the requesting virtual core when SETER.dhcce is set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap. The earliest instruction on which the disrupting trap can be taken is the instruction after the one that encountered the error.

Implementation Note | Due to an implementation bug, the normal forwarding of the instruction around the Icache (used to guarantee forward progress with multiple strands) does **not** guarantee forward progress for the case where the valid bit has a permanent error.

1. The way information captured may not be correct for an instruction in a control transfer delay slot.

25.7.3.2 Icache Tag Parity Error (ICTP)

The Icache tag is protected with parity. The Icache tag parity is checked with each instruction fetch if CERER.ictp is set. The parity is checked for each of the possible valid ways in the set. If a parity error is found in any of the ways, hardware encodes ictp and captures the way[1] and set information in the DESR register. Hardware invalidates all ways in the accessed Icache set.

A disrupting *hw_corrected_error* trap is generated to the requesting virtual core when SETER.dhcce is set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap. The earliest instruction on which the disrupting trap can be taken is the instruction after the one that encountered the error.

Implementation Note The normal forwarding of the instruction around the Icache (used to guarantee forward progress with multiple strands) will also guarantee forward progress for the case where the tag has a permanent error.

25.7.3.3 Icache Tag Multiple Hit Error (ICTM)

Multiple errors in the tags can lead to multiple hits within a set. If CERER.ictm is set, each instruction fetch is checked for multiple hits. If multiple hits are detected on an instruction fetch, hardware encodes ictm and captures one of the ways that hit[2] and the set information in the DESR register. Hardware invalidates all ways in the accessed Icache set.

A disrupting *hw_corrected_error* trap is generated to the requesting virtual core when SETER.dhcce is set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap. The earliest instruction on which the disrupting trap can be taken is the instruction after the one that encountered the error.

Implementation Note The normal forwarding of the instruction around the Icache (used to guarantee forward progress with multiple strands) will also guarantee forward progress for the case where the tag has a permanent error.

25.7.3.4 Icache Data Parity Error (ICDP)

Each 32-bit instruction in the Icache data array and the instruction buffers is protected with parity. The instruction data parity is checked with each instruction decode if CERER.icdp is set. Hardware can not distinguish between a parity error that occurs in the instruction cache data array and a parity error that occurs in the instruction buffer.

1. The way information captured may not be correct for an instruction in a control transfer delay slot.
2. The way information captured may not be correct for an instruction in a control transfer delay slot.

When a parity error is detected, hardware encodes icdp and captures the set information in the DESR register. Hardware invalidates all ways in the accessed Icache set. If the parity error actually occurred in the instruction buffer and not in the data array, the set information in the DESR is irrelevant.

A disrupting *hw_corrected_error* trap is generated to the requesting virtual core when SETER.dhcce is set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap. The earliest instruction on which the disrupting trap can be taken is the instruction after the one that encountered the error.

Implementation Notes The normal forwarding of the instruction around the Icache (used to guarantee forward progress with multiple strands) will also guarantee forward progress for the case where the data has a permanent error.

25.7.4 Dcache Errors

The priority of Dcache errors is DCVP > DCTP > DCTM > DCDP.

Programming Note If the D$ is in direct-mapped mode, and one of the following parity errors occurs, the way replaced will be the way which experienced the parity error, not the way specified by address bits [12:11].

25.7.4.1 Dcache Valid Parity Error (DCVP)

The Dcache tag valid bits are protected by duplication: there is a *master* copy and a *slave* copy. The valid bits are checked for each data load for all ways in the accessed set if CERER.dcvp is set. If CERER.dcvp is not set, the *master* copy of the valid bit simply determines whether a cache hit or miss occurred. If a valid bit mismatch occurs and CERER.dcvp is set, hardware invalidates all ways in the accessed set. If the DESR.f bit is clear, hardware records the index and the way with the error in the DESR ErrorAddress field, and sets the DESR.f and DESR.dcvp bits. If the DESR.f bit is set, hardware sets the DESR.me bit but does not record the index and way in the DESR. In addition, if the SETER.dhcce bit is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor for logging of the error. If the *hw_corrected_error* trap is not taken, the error is remembered via DESR.f and a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor when SETER.dhcce is later set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap.

Implementation Notes

The normal forwarding of the data around the Dcache guarantees forward progress for the case where the valid bit has a permanent error.

Because stores do not read the Dcache they do not check the Dcache valid parity. Stores do not read the Dcache since coherency is maintained by the L2 directory.

25.7.4.2 Dcache Tag Parity Error (DCTP)

The Dcache tag is protected with parity. The Dcache tag parity is checked with each data load if CERER.dctp is set. If CERER.dctp is not set, the error is ignored. The parity is checked for each of the possible valid ways in the set. If a parity error is found in any of the ways, hardware invalidates all ways in the accessed set, encodes dctp and captures the way and set information in the DESR register. In addition, if the SETER.dhcce bit is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor for logging of the error. If the *hw_corrected_error* trap is not taken, the error is remembered via DESR.f and a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor when SETER.dhcce is later set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher-priority trap.

Implementation Notes

The normal forwarding of the data around the Dcache guarantees forward progress for the case where the tag has a permanent error.

Because stores do not read the Dcache they do not check the Dcache tag parity. Stores do not read the Dcache since coherency is maintained by the L2 directory.

25.7.4.3 Dcache Tag Multiple Hit Error (DCTM)

Multiple errors in the tags can lead to multiple hits within a set. If CERER.dctm is set, each data load checks for multiple hits. If multiple hits are detected on an data load, hardware invalidates all ways in the accessed set, encodes dctm and captures one of the ways that hit and the set information in the DESR register. In addition, if the SETER.dhcce bit is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor for logging of the error. If the *hw_corrected_error* trap is not taken, the error is remembered via DESR.f and a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor when SETER.dhcce is later set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher priority trap.

Implementation Notes

The normal forwarding of the data around the Dcache guarantees forward progress for the case where the tag has a permanent error.

Because stores do not read the Dcache, they do not check the Dcache for multiple tag hits. Stores do not read the Dcache since coherency is maintained by the L2 directory.

25.7.4.4 Dcache Data Parity Error (DCDP)

Each byte in the Dcache data array is protected with parity. The Dcache data parity is checked for each of the valid ways with each data load if CERER.dcdp is set. When a parity error is detected, hardware invalidates all ways in the accessed set, encodes dcdp and captures the set information in the DESR register. In addition, if the SETER.dhcce bit is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor for logging of the error. If the *hw_corrected_error* trap is not taken, the error is remembered via DESR.f and a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor when SETER.dhcce is later set (and PSTATE.ie is set or HPSTATE.hpriv is clear) and there is no higher-priority trap.

Implementation Notes

The normal forwarding of the data around the Dcache guarantees forward progress for the case where the data has a permanent error.

Because stores do not read the Dcache, they do not check Dcache data parity. Stores do not read the Dcache since coherency is maintained by the L2 directory.

25.7.5 IRF ECC Error (IRFC and IRFU)

Each IRF entry is protected by SEC/DED ECC. Up to three operands can be read from the IRF at a time. Hardware checks each operand's ECC independently. Hardware prioritizes operand errors in the order rs1 > rs2 > rs3. This means that an uncorrectable error which occurs on a lower-priority operand (e.g., rs2) simultaneously with a correctable error on a higher-priority operand (e.g., rs1) will not be reported (until the correctable error is cleared by software and the instruction is retried).

Hardware can only detect an IRF ECC error if CERER.irf is set. If hardware detects either a correctable or uncorrectable error for any valid operand, what happens depends upon the setting of SETER.pscce.

If SETER.pscce is set, hardware records the error type in the D-SFSR by encoding IRFC or IRFU as appropriate, records the IRF index and the ECC syndrome in the D-SFAR (see Table 25-11 on page 253), and generates a precise *internal_processor_error* trap.

Programming Note

Software should correct an IRFC error before issuing a retry instruction. Software can correct the error as follows.

- It reads the D-SFAR contents and decodes the failing address location in the IRF, turns off the SETER.pscce bit, and reads the data. If desired for logging, software can also read the ECC bits by using an LDXA to the `ASI_IRF_ECC_REG` register (See *ASI_IRF_ECC_REG* on page 425).
- It decodes the ECC syndrome. If the error is in the data bits, it **xor**s the correction mask with the data read from the IRF.
- It then writes the corrected data into the IRF using a normal integer instruction (hardware generates the proper ECC prior to writing the IRF entry). In the process of reading the failing location, another IRF ECC error will occur, but will not be recorded or trapped as SETER.pscce is clear.

After correcting the data, software should turn the SETER.pscce bit back on. Then software can retry the original failing instruction.

An IRFU error is generally not recoverable.

If the SETER.pscce bit is not set, the error is not recorded, and hardware continues executing, using the uncorrected data read from the IRF. This will lead to data corruption.

Since up to three operands can be read for each instruction, software may define an appropriate error threshold for the instruction before considering the IRF broken.

25.7.6 FRF ECC Error (FRFC and FRFU)

Each FRF entry is protected by SEC/DED ECC. Up to two operands can be read from the FRF at a time. Hardware checks each operand's ECC independently. Note: PDIST reads 3 operands over 2 cycles, so hardware still prioritizes three operands for reporting errors. Hardware prioritizes operand errors in the order rs1 > rs2 > rs3. This means that an uncorrectable error which occurs on a lower-priority operand (e.g., rs2) simultaneously with a correctable error on a higher priority operand (e.g., rs1) will not be reported (until the correctable error is cleared by software and the instruction is retried).

Hardware can only detect an FRF ECC error if CERER.frf is set. If hardware detects either a correctable or uncorrectable error for any valid operand, what happens depends upon the setting of SETER.pscce.

If SETER.pscce is set, hardware records the error type by encoding FRFC or FRFU in the D-SFSR as appropriate, records the failing FRF index and ECC syndrome in the D-SFAR (see Table 25-11 on page 253), and generates a precise *internal_processor_error* trap.

Programming Note

Software should correct an FRFC error before issuing a retry instruction. Handling of an FRFC error is similar to an IRFC error. The additional complication is that each FRF entry contains two ECC words (due to the single-precision FP registers). So two corrections may have to be performed. Software can correct a correctable error as follows:

- It reads the failing FRF index and ECC syndrome from the D-SFAR. It decodes the failing address location in the FRF, turns off the SETER.pscce bit, and reads the data. If desired for logging, software can also read the ECC bits by using an LDXA to the ASI_FRF_ECC_REG register (See *ASI_FRF_ECC_REG* on page 426).
- Hardware reports 2 syndromes in the D-SFAR, even for single-precision FP operations (the syndromes correspond to the 2 SP registers of an even/odd SP pair). For a double-precision operation, both syndromes are pertinent. Hardware does not indicate which of the syndromes are pertinent for an SP operation. Both syndromes may in fact indicate an ECC error. It is also possible for one of the syndromes (the one corresponding to the even or odd SP register which was not read) to be incorrect.[1] Software should inspect the instruction at %tpc. If it was a double-precision operation, it should decode both syndromes. If it was a single-precision operation, it decodes the register numbers of the sources accordingly. That identifies which of the syndromes is pertinent.
- If an error occurred in the data bits, software **xor**s the correction mask with the data.
- It then writes the corrected data into the FRF using a normal FP operation (hardware generates the proper ECC before writing the FRF). In the process of reading the failing FRF location, another FRF ECC error will occur, but will not be recorded or trapped as SETER.pscce is clear.

After correcting the data, software should turn the SETER.pscce bit back on. Then software can retry the original failing instruction.

1. Hardware captures the syndromes in the D-SFAR at the time an instruction reads its operands from the FRF, without regard to previous single-precision operations in the pipeline which may update the value of one of the SP registers of an even/odd pair. If both the even and odd SP registers of an even-odd pair have ECC errors, hardware reports an error on the register being read, and the other syndrome is unneeded. Consider the following example. If the odd SP register of an even/odd pair is being overwritten by a single-precision instruction still in the pipeline when the instruction reading from the even register reads the FRF, hardware takes a trap on the instruction reading the even register. Since at the time of the trap and the capture of the syndromes in the D-SFAR, the instruction writing the odd register has not updated the register value, it still shows as having an ECC error.

An FRFU error is generally not recoverable.

If SETER.pscce is not set, the error is not recorded, and hardware continues executing, using the uncorrected data read from the FRF. This will lead to data corruption.

Software may define an appropriate error threshold for the instruction before considering the FRF broken.

25.7.7 Store Buffer

The Store Buffer (STB) is organized as a CAM which contains the tag portion of the address and a RAM which contains the data and status bits. The status bits consist of the privilege level of the store. OpenSPARC T2 implements ECC in the data array for data bits, and the Cam bits are protected by a single parity bit. The store buffer is accessed on data loads (to check for RAW (Read-After-Write) hits) and on PCX reads[1] and ASI ring stores.[2] It can also be accessed with diagnostic reads, but these accesses do not cause parity or ECC errors.

25.7.7.1 Correctable Data ECC Error on a Load (SBDLC)

Hardware detects this error only if CERER.sbdlc is set. If a load which results in a full RAW hit in the STB gets a single-bit data error, hardware does not correct the load data.

If SETER.pscce is set, hardware records the error in the D-SFSR by encoding sbdlc, writes the store buffer index of the entry in error in the D-SFAR, and generates a precise *internal_processor_error* trap. Since hardware will correct the data before writing the store data to memory, this error is likely recoverable; software can issue a retry to reexecute the load.

1. A PCX read occurs when the store is sent to the L2 cache or NCU. PCX stands for Processor to Cache Xbar.

2. Stores to ASI space which go over the ASI ring internal to the processor are referred to as ASI ring stores.

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

25.7.7.2 Uncorrectable Data ECC Error on a Load (SBDLU)

Hardware detects this error only if CERER.sbdlu is set. If a load which results in a full RAW hit in the STB gets an uncorrectable data ECC error, the following flow occurs.

If SETER.pscce is set, the error is recorded in the D-SFSR by encoding sbdlu, and writing the store buffer index of the entry in error to the D-SFAR, and generates a precise *internal_processor_error* trap. Another uncorrectable error will likely occur when hardware reads the store buffer entry to write the store data to memory. (See *Uncorrectable Data ECC Error on a PCX Read to Memory (SBDPU)* below.)

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

25.7.7.3 STB Address Parity Error on a Load

OpenSPARC T2 does not check CAM parity for load accesses.

25.7.7.4 Correctable Data ECC Error on a PCX Read to Memory or I/O or Read for an ASI Ring Store (SBDPC)

If CERER.sbdpc is set, on a PCX read to memory or I/O space or a read for an ASI ring store which results in a single bit ECC error, hardware corrects the error before forwarding the data to the crossbar or the ASI ring. Hardware encodes sbdpc and writes the failing store buffer index to the DESR.

If SETER.dhcce is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware presents a disrupting *hw_corrected_error* trap to the core.

If SETER.dhcce is not set, hardware continues executing. Assuming software has not reset DESR.f, a disrupting trap will be presented to the core when software sets SETER.dhcce (and PSTATE.ie is set or HPSTATE.hpriv is clear).

25.7.7.5 Uncorrectable Data ECC Error on a PCX Read to Memory (SBDPU)

If CERER.sbdpu_sbdiou is set, on a PCX read to memory which results in an uncorrectable ECC error, hardware generates NotData before forwarding the data to the crossbar. The error is recorded in the DESR by encoding sbdpu, and writing the store buffer index to the DESR.

If SETER.de is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware presents a disrupting *sw_recoverable_error* trap to the core.

If SETER.de is not set, hardware continues executing. Note that if SETER.de is not set, hardware has performed a bad store which will not be detected until the bad value is loaded. When software sets SETER.de (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware will present a disrupting *sw_recoverable_error* trap to the core.

25.7.7.6 Uncorrectable Data ECC Error on a PCX Read to I/O or Read for an ASI Ring Store (SBDIOU)

If CERER.sbdpu_sbdiou is set, on a PCX read to I/O space or read for ASI Ring Store which results in an uncorrectable ECC error, hardware suppresses the store and all subsequent stores then in the store buffer for that strand. The error is recorded in the DFESR by encoding sbdiou, and writing the store buffer index and privilege level to the DFESR. The privilege level recorded is the highest privilege level for any store in the store buffer at the time of the error.

Hardware presents a deferred *store_error* trap to the core. Software can decide what termination action is appropriate. Software at the trap handler should read the store buffer using store buffer diagnostic reads (see *Store Buffer — ASI_STB_ACCESS* on page 427) before issuing any stores which will overwrite the store buffer.

25.7.7.7 Address Bit Parity Error on a PCX read or Read for an ASI Ring Store (SBAPP)

If CERER.sbapp is set, on a PCX read or an ASI ring store read which exposes a parity error on the address bits, hardware suppresses the store, and logs the error in the DFESR by setting sbapp. Hardware suppresses other (younger, subsequent) stores in the store buffer. The highest privilege level for any store in the store buffer at the time of the error is also recorded.

Hardware presents a deferred *store_error* trap to the core.

If a user process was running, software may be able to avoid taking down the entire chip; similarly if an operating system was running, software may be able to kill only the partition the operating system was running in.

25.7.8 Scratchpad Array (SCAC and SCAU)

The Scratchpad array contains the scratchpad registers. It can be accessed only via normal ASI loads and stores or diagnostic ASI loads. The array is protected via SEC/DED ECC.

Hardware detects correctable Scratchpad errors only if CERER.scac is set, and uncorrectable errors only if CERER.scau is set.

ECC is not checked for diagnostic reads of the array, so a diagnostic read can not result in an error.

If a normal ASI read of the array results in a correctable ECC error, hardware corrects neither the returned data nor the error in the array.

If the SETER.pscce bit is set, hardware records the error in the D-SFSR by encoding scac, and records the array index with the error and syndrome in the D-SFAR. Hardware signals a precise *internal_processor_error* trap to the core. When software takes the trap, it can correct the data in the array. It decodes the syndrome, issues a diagnostic ASI read to read the data and ECC check bits, computes the correct data, and writes the corrected data back using a normal ASI store. Hardware generates the proper ECC before writing to the array.

If the SETER.pscce bit is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

If a normal ASI read of the array results in an uncorrectable ECC error, and SETER.pscce is set, hardware records the error in the D-SFSR by encoding SCAU. The array index with the error is stored in the D-SFAR. Hardware signals a precise *internal_processor_error* trap to the core.

If the SETER.pscce bit is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

25.7.9 Tick_compare (TCCP, TCUP, TCCD, TCUD)

The Tick_compare arrays are also protected via SEC/DED ECC. The Tick_compare array stores TICK_CMPR, STICK_CMPR, and HSTICK_CMPR. The arrays have three access means. The first is via ASR reads and writes. The second is via diagnostic ASI loads. The third, compare access, is implicit as hardware cycles through the entries to compare the TICK/STICK register with the TICK_CMPR registers.

Hardware detects correctable Tick_compare precise/disrupting errors only if CERER.tccp/tccd is set, and uncorrectable precise/disrupting errors only if CERER.tcup/tcud is set.

ECC is checked for a normal ASR read. If a correctable error occurs, hardware corrects neither the returned data nor the array location.

If SETER.pscce is set, hardware records the error in the D-SFSR by encoding tccp and the failing array index and syndrome is stored in the D-SFAR. Hardware generates a precise *internal_processor_error* trap to the core. For a correctable error, software at the trap handler can rewrite the array location and retry the failing instruction. It decodes the syndrome, issues a diagnostic ASI read to read the data

and check bits, computes the correct data, and writes the corrected data back using a normal ASR write. Hardware generates the proper ECC before writing the data to the array.

If the SETER.pscce bit is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

If an uncorrectable error occurs on a normal ASR read, and SETER.pscce is set, hardware records the error in the D-SFSR by encoding tcup and writes the failing index and syndrome to the D-SFAR. Hardware takes a precise *internal_processor_error* trap. Software may be able to recover from this error by picking a reasonable value to load the TICK_CMPR register with, and retrying the ASR read.

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

ECC is not checked for a diagnostic ASI load, so no error is recorded and no trap can occur for this access type.

ECC is checked for a compare access. If a correctable or uncorrectable error occurs, hardware does not correct the data in the array, and suppresses any compare operation. Hardware records the error by encoding either tccd or tcud, and writing the failing array index in the DESR. The DESR is updated irrespective of the setting of SETER.de.

If SETER.de is set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware presents a disrupting *sw_recoverable_error* trap to the core.

Programming Note For a TCCD error, software can attempt recovery by using diagnostic array ASI accesses to correct the data as described for TCCP processing above. For a TCUD error, software may be able to recover from the error by setting the appropriate softened bit, and reloading the TICK_CMPR register after processing of the interrupt completes.

If SETER.de is not set, hardware continues executing without regard to the error. This can result in HW taking an *hstick_match* or *interrupt_level_n* trap based upon a faulty comparison. When software sets SETER.de (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware will present a disrupting *sw_recoverable_error* trap to the core.

25.7.10 Trap Stack Array (TSAC and TSAU)

The TSA array is protected via SEC/DED ECC. It contains the Trap Stack array and the mondo interrupt queue registers. It can be accessed via privileged register reads and writes or diagnostic ASI accesses. Privileged register writes require a read-

modify-write operation, so privileged writes can generate an ECC error. The TSA is also accessed during Done and Retry instructions. Diagnostic accesses to the TSA do not generate ECC errors.

Hardware detects correctable TSA errors only if CERER.tsac is set. It detects uncorrectable TSA errors only if CERER.tsau is set.

If hardware detects a correctable error during an ASR read or write, or a DONE or RETRY instruction, hardware corrects neither the data returned by the read nor the array location. If the access was an ASI write, hardware suppresses the array write.

If SETER.pscce is set, hardware records the error in the D-SFSR by encoding tsac, and writes the failing TSA index and syndrome in the D-SFAR. Hardware presents the core with a precise *internal_processor_error* trap. Software can attempt recovery as follows. First note that each TSA array location contains several logical registers (see *Trap Stack Array (TSA)* on page 430) for details). Also, certain bits in each array location are unused (and assumed to be 0 during ECC generation). First, it turns off TSA error reporting by setting CERER.tsac to 0. The decoded syndrome from the D-SFAR indicates whether the error was in one of the unused bits in the array. If not, based upon the decoded syndrome, it reads the architected register which spans the failing bit using a RDPR/HPR instruction. After flipping the erroneous bit, it writes the new value back to the architected register using a WRPR/HPR instruction. If the decoded syndrome pointed to an unused bit, then software can read any register in the failing location using a RDPR/HPR instruction, and write the same value back using a WRPR/HPR instruction. Hardware will regenerate the correct ECC. Software then sets CERER.tsac to 1 to reenable TSAC error reporting.

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

If hardware detects an uncorrectable error during the read access for a privileged register read or write, or a DONE or RETRY instruction, and SETER.pscce is set, it records the uncorrectable error to the D-SFSR by encoding tsau, and writes the failing array index to the D-SFAR. It then presents the core with a precise *internal_processor_error* trap.

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

25.7.11 MMU Register Array (MRAU)

The MRA (MMU Register Array) contains various pointers and configuration registers used by hardware tablewalk and the MMU. Software can maintain a clean copy of the MRA contents; hardware does not update or store any MRA contents without software participation. (See *MMU Register Array (MRA)* on page 434 for details of the information stored in the MRA).

Each MRA location is protected by parity. The MRA is accessed by normal ASI reads and writes, diagnostic ASI reads and writes, and for hardware tablewalks. It is also read-modify-write for ASI writes. Diagnostic accesses to the MRA do not generate parity errors.

Hardware detects MRA parity errors only if CERER.mrau is set.

If hardware detects a parity error during a normal ASI access, hardware signals an MRAU error to the trap unit. If the ASI access was a normal ASI write, hardware suppresses the array update.

If SETER.pscce is set, hardware records the error in the D-SFSR by encoding mrau, and writes the failing array index to the D-SFAR. Hardware presents a precise *internal_processor_error* to the strand. Software can attempt recovery from the error by reloading the entry from its clean copy of the MRA contents and retrying the ASI access.

If SETER.pscce is not set, hardware continues executing using the uncorrected, and possibly erroneous, data. The error is not recorded.

Parity is not checked for diagnostic ASI reads and writes.

If an MRA location gets a parity error during a hardware tablewalk, the MRA error results in a precise *instruction_access_MMU_error* or *data_access_MMU_error* trap to the strand (see previous error handling sections). Software can attempt recovery from an error as above for an ASI access.

25.8 SPARC Error Registers

CERER (Core Error Recording Enable) register enables recording of an individual hardware error in either the I-SFSR, D-SFSR and D-SFAR, DESR, or DFESR. Bits in SETER (Core Error Trap Enable) register further control whether or not a trap occurs for a recorded error. A trap can never occur for an error that is not recorded; thus, a trap will be taken only if both CERER and associated SETER enable bits are set. Additionally, precise traps which are masked off (either by the appropriate CERER bit or SETER.pscce) will not update the I-SFSR or D-SFSR and D-SFAR. Hardware only updates the I-SFSR or D-SFSR and D-SFAR if a precise trap is taken.

25.8.1 ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER

The strands on a physical core share a hyperprivileged ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER (CERER) at ASI $4C_{16}$, VA{63:0} = 10_{16}. This register controls the reporting of errors to the virtual processor, and is intended only for use during debug. Software may also use this register to selectively disable error recording and trap generation in special circumstances. The format of the CERER register is shown in TABLE 25-4.

TABLE 25-4 CERER – ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER (ASI $4C_{16}$, VA 10_{16}) *(1 of 3)*

Bit	Field	Initial Value	R/W	Description
63	ittp	0	RW	If set to 0, mask ITTP errors. If set to 1, enable ITTP errors.
62	itdp	0	RW	If set to 0, mask ITDP errors. If set to 1, enable ITDP errors.
61	ittm	0	RW	If set to 0, mask ITTM errors. If set to 1, enable ITTM errors.
60	—	0	RO	*Reserved*
59	hwtwmu	0	RW	If set to 0, mask all uncorrectable MRA errors on hwtw. If set to 1, enable all uncorrectable MRA errors on hwtw.
58	hwtwl2	0	RW	If set to 0, mask all L2 errors on hwtw. If set to 1, enable all L2 errors on hwtw.
57	—	0	RO	*Reserved*
56	—	0	RO	*Reserved*
55	icl2c	0	RW	If set to 0, mask ICL2C errors. If set to 1, enable ICL2C errors.
54	icl2u	0	RW	If set to 0, mask ICL2U errors. If set to 1, enable ICL2U errors.
53	icl2nd	0	RW	If set to 0, mask ICL2ND errors. If set to 1, enable ICL2ND errors.
52	irf	0	RW	If set to 0, mask all IRF errors. If set to 1, enable all IRF errors.
51	—	0	RO	*Reserved*
50	frf	0	RW	If set to 0, mask all FRF errors. If set to 1, enable all FRF errors.
49	—	0	RO	*Reserved*
48	dttp	0	RW	If set to 0, mask DTTP errors. If set to 1, enable DTTP errors.
47	dttm	0	RW	If set to 0, mask DTTM errors. If set to 1, enable DTTM errors.
46	dtdp	0	RW	If set to 0, mask DTDP errors. If set to 1, enable DTDP errors.
45	—	0	RO	*Reserved*
44	—	0	RO	*Reserved*
43	—	0	RO	*Reserved*
42	—	0	RO	*Reserved*
41	—	0	RO	*Reserved*
40	dcl2c	0	RW	If set to 0, mask DCL2C errors. If set to 1, enable DCL2C errors.
39	dcl2u	0	RW	If set to 0, mask DCL2U errors. If set to 1, enable DCL2U errors.

TABLE 25-4 CERER – ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER (ASI $4C_{16}$, VA 10_{16}) *(2 of 3)*

Bit	Field	Initial Value	R/W	Description
38	dcl2nd	0	RW	If set to 0, mask DCL2ND errors. If set to 1, enable DCL2ND errors.
37	sbdlc	0	RW	If set to 0, mask SBDLC errors. If set to 1, enable SBDLC errors.
36	sbdlu	0	RW	If set to 0, mask SBDLU errors. If set to 1, enable SBDLU errors.
35	—	0	RO	*Reserved*
34	—	0	RO	*Reserved*
33	mrau	0	RW	If set to 0, mask MRAU errors. If set to 1, enable MRAU errors.
32	tsac	0	RW	If set to 0, mask TSAC errors. If set to 1, enable TSAC errors.
31	tsau	0	RW	If set to 0, mask TSAU errors. If set to 1, enable TSAU errors.
30	scac	0	RW	If set to 0, mask SCAC errors. If set to 1, enable SCAC errors.
29	scau	0	RW	If set to 0, mask SCAU errors. If set to 1, enable SCAU errors.
28	tccp	0	RW	If set to 0, mask TCCP errors. If set to 1, enable TCCP errors.
27	tcup	0	RW	If set to 0, mask TCUP errors. If set to 1, enable TCUP errors.
26:24	—0	0	RO	*Reserved*
23	sbapp	0	RW	If set to 0, mask SBAPP errors. If set to 1, enable SBAPP errors.
22	—	0	RO	*Reserved*
21	l2c_socc	0	RW	If set to 0, mask L2C and SOCC errors. If set to 1, enable L2C and SOCC errors.
20	l2u_socu	0	RW	If set to 0, mask L2U and SOCU errors. If set to 1, enable L2U and SOCU errors.
19	l2nd	0	RW	If set to 0, mask L2ND errors. If set to 1, enable L2ND errors.
18	icvp	0	RW	If set to 0, mask ICVP errors. If set to 1, enable ICVP errors.
17	ictp	0	RW	If set to 0, mask ICTP errors. If set to 1, enable ICTP errors.
16	ictm	0	RW	If set to 0, mask ICTM errors. If set to 1, enable ICTM errors.
15	icdp	0	RW	If set to 0, mask ICDP errors. If set to 1, enable ICDP errors.
14	dcvp	0	RW	If set to 0, mask DCVP errors. If set to 1, enable DCVP errors.
13	dctp	0	RW	If set to 0, mask DCTP errors. If set to 1, enable DCTP errors.
12	dctm	0	RW	If set to 0, mask DCMH errors. If set to 1, enable DCMH errors.
11	dcdp	0	RW	If set to 0, mask DCDP errors. If set to 1, enable dcDP errors.
10	sbdpc	0	RW	If set to 0, mask SBDPC errors. If set to 1, enable SBDPC errors.
9	sbdpu_ sbdiou	0	RW	If set to 0, mask SBDPU and SBDIOU errors. If set to 1, enable SBDPU and SBDIOU errors.
8	mamu	0	RW	If set to 0, mask MAMU errors. If set to 1, enable MAMU errors.
7	tccd	0	RW	If set to 0, mask TCCD errors. If set to 1, enable TCCD errors.
6	tcud	0	RW	If set to 0, mask tcud errors. If set to 1, enable tcud errors.
5	mal2c	0	RW	If set to 0, mask MAL2C errors. If set to 1, enable MAL2C errors.
4	mal2u	0	RW	If set to 0, mask MAL2U errors. If set to 1, enable MAL2U errors.
3	mal2nd	0	RW	If set to 0, mask MAL2ND errors. If set to 1, enable MAL2ND errors.

TABLE 25-4 CERER – ASI_CORE_ERROR_RECORDING_ENABLE_REGISTER (ASI $4C_{16}$, VA 10_{16}) *(3 of 3)*

Bit	Field	Initial Value	R/W	Description
2	—	0	RW	—
1	—	0	RW	—
0	—	0	RW	—

Each bit of the enable fields for precise errors (CERER bits 63:61, 59:58, 54:53, 52, 50, 48:46, 39:36, 33:27) controls whether a corresponding precise error is recorded in the I-SFSR or D-SFSR. A 1 in the CERER bit position enables the corresponding error to be recorded if that error occurs. Otherwise, that error is not recorded. Similarly, each bit of the DESR enable field For disrupting errors (CERER bits 55, 40, 21:10, 8:0) control whether the corresponding disrupting error is recorded in the DESR. Bits 23 and 9 oF the CERER control whether the associated error is logged in the DFESR.

Logically the CERER bit is **and**ed with error sources before an error is encoded and recorded in the I-SFSR, D-SFSR, DESR, or DFESR. A masked enable in the CERER Precise Error enable field cannot cause the I-SFSR or D-SFSR to be updated if the associated error occurs. Similarly a masked enable in the CERER DESR enable field cannot cause the DESR.f or the DESR.me bits to be set, nor can a masked enable in the DFESR enable field cause a DFESR bit to be set.

25.8.2 ASI_STRAND_ERROR_TRAP_ENABLE_REGISTER

Each virtual processor has a hyperprivileged ASI_STRAND_ERROR_TRAP_ENABLE_REGISTER (SETER) at ASI $4C_{16}$, VA{63:0} = 18_{16}. This register controls the generation of traps for errors that are reported to the virtual processor.

Each of the SETER bits apply to a particular class of maskable errors. Nonmaskable errors always result in traps regardless of the setting of the SETER bits (assuming that the CERER bit for the error is set).

Bit 62, pscce, controls whether a trap will be taken if a maskable, precise software correctable error is detected during the execution of an instruction. Since hardware performs neither correction nor clearing of the error, software must take action to avoid unpredictable execution and possible data loss. This bit should always be set for normal operation. An example of this type of error is a correctable IRF ECC error. Furthermore, if pscce is 0, no maskable, precise software correctable errors will be recorded in the I-SFSR or the D-SFSR and D-SFAR.

Bit 61, de, controls whether a *sw_recoverable_error* disrupting trap will be taken if the DESR.f is set for a maskable disrupting error which is not hardware corrected and cleared. (Note that a disrupting trap is also conditioned by the setting of the PSTATE.ie bit when HPSTATE.hpriv is set; PSTATE.ie must be set in this case to take a disrupting trap).

Bit 60, dhcce, controls whether a *hw_corrected_error* disrupting trap will be taken for a maskable disrupting hardware error that was corrected and cleared. In this case, software should log the error. (Note that a disrupting trap is also conditioned by the setting of the PSTATE.ie bit when HPSTATE.hpriv is set; PSTATE.ie must be set in this case to take a disrupting trap).

Note | In normal operation, pscce and de should *always* be set. Otherwise, the OpenSPARC T2 core will continue to execute in the face of hardware errors, leading to unpredictable behavior.

The format of the SETER register is shown in TABLE 25-5.

TABLE 25-5 SETER – ASI_STRAND_ERROR_TRAP_ENABLE_REGISTER (ASI $4C_{16}$, VA 18_{16})

Bit	Field	Initial Value	R/W	Description
63	—	0	RO	*Reserved*
62	pscce	0	RW	If set to 1, trap on maskable, precise software corrected and cleared errors
61	de	0	RW	If set to 1, trap on maskable, disrupting errors which are not hardware corrected and cleared
60	dhcce	0	RW	If set to 1, trap on maskable, disrupting hardware corrected and cleared errors.
59:0	—1	0	RO	*Reserved*

25.8.3 IMMU Synchronous Fault Status Register

OpenSPARC T2 uses the I-SFSR to record precise hardware errors encountered during the instruction fetch process. For most instruction fetch errors, the TPC[TL] records the relevant VA. For an IT MRA[1]correctable or uncorrectable error, the D-SFAR records the failing MRA index. Using the D-SFAR obviates the need for an ISFAR. If the error occurred for a fetch to L2 (either during an hwtw or instruction fetch), an L2 ESR contains more information about the error.

1. MRA stands for the MMU Register Array; it contains various configuration registers required for the MMU and hardware tablewalk.

Each virtual processor has one hyperprivileged, read/write, ASI_IMMU_SFSR register located at ASI 50_{16}, VA 18_{16}. The format of the ISFSR register is shown in TABLE 25-6.

TABLE 25-6 ISFSR – ASI_IMMU_SYNCHRONOUS_FAULT_STATUS_REGISTER (ASI 50_{16}, VA 18_{16})

Bit	Field	Initial Value	R/W	Description
63:3	—	0	RO	*Reserved*
2:0	errtype	0	RW	Error type information, format defined in TABLE 25-7.

Note | ISFSR is preserved across warm resets to allow software to determine the cause of an error that required a warm reset.

TABLE 25-7 describes the hardware errors that are recorded in the ISFSR register. If multiple error conditions occur for the same instruction, only the highest priority is logged. If multiple exceptions occur for the same instruction, the ISFSR will only be updated if the error is the highest-priority exception for the instruction. For example, if an access to a privileged page in user mode occurs in conjunction with an Icache tag parity error, the I-SFSR will not be updated. Since ISFSR only logs precise errors, a write clearing this register cannot be simultaneous to an error that would set the I-SFSR.

Note | There is no “latching” of the first error. If another error occurs before the I-SFSR has been examined, the new error information is captured in the I-SFSR, overwriting the old information.

TABLE 25-7 Hardware Errors Recorded in the I-SFSR Register

Trap Type	Error Name	Error Description	Relative Priority	Contents of I-SFSR ErrorType Field	Contents of D-SFAR (see TABLE 25-11)
instruction_access_MMU_error	ITTM	IT Tag Multiple hit	1	1	TPC[TL] contains VA
	ITTP	IT Tag Parity	2	2	TPC[TL] contains VA
	ITDP	IT Data Parity	3	3	TPC[TL] contains VA
	ITMU	IT MRA Uncorrectable	4	4	2
	ITL2U	IT L2 Uncorrectable	5	5	TPC[TL] contains VA
	ITL2ND	IT L2 NotData	5	6	TPC[TL] contains VA
instruction_access_error	ICL2U	IC L2 Uncorrectable	6	1	TPC[TL] contains VA
	ICL2ND	IC L2 NotData	6	2	TPC[TL] contains VA

The Trap Type column describes the trap type which will result if the error occurs and the trap is enabled (see SETER in *ASI_STRAND_ERROR_TRAP_ENABLE_REGISTER* on page 247). The Error Name column names the hardware error, and the Error Description column provides a brief description of the error. If multiple errors occur for the same instruction, the error that is recorded in the I-SFSR is determined by the Relative Priority column, with 1 being the highest priority. The error information recorded in the I-SFSR is indicated by the encoding specified in the errortype field while the format of the address information recorded in the D-SFAR is described by the entry in the last column for ITMU errors, with reference to TABLE 25-11 on page 253.

25.8.4 DMMU Synchronous Fault Status and Address Registers

OpenSPARC T2 records data-related precise hardware errors in the D-SFSR and D-SFAR. Since a hardware error has higher priority than a normal program error, OpenSPARC T2 saves hardware by using the D-SFSR and D-SFAR for precise data hardware errors.

The error types recorded in the D-SFSR and D-SFAR are listed in TABLE 25-8. TSA stands for Trap Stack Array; the TSA contains the V9 trap stack and mondo queue interrupt registers. SCA stands for Scratchpad Array; the SCA contains the

scratchpad registers. If multiple error conditions occur for the same access, only the highest priority is logged. If multiple exceptions occur for the same instruction, the D-SFSR will only be updated if the error is the highest priority exception for the instruction. For example, if an FRF ECC error occurs in conjunction with an privileged opcode exception, the D-SFSR will not be updated. Since the D-SFSR and D-SFAR log precise errors, a write clearing the D-SFSR cannot be simultaneous to an error that would set the D-SFSR and D-SFAR.

Notes

On a block store instruction which gets multiple FRF ECC errors, only the first error is logged in the D-SFSR and D-SFAR.

There is no "latching" of the first error in the D-SFSR and D-SFAR. If another error occurs before the D-SFSR and D-SFAR have been examined, the new error information is captured in the D-SFSR and D-SFAR, overwriting the old information.

TABLE 25-8 Hardware Errors Recorded in the D-SFSR and D-SFAR Registers

Trap Type	Error Name	Error Description	Relative Priority	Contents of D-SFSR ErrorType Field	Contents of D-SFAR (see TABLE 25-11
internal_processor_error	IRFU	Integer register file uncorrectable	1	1	8
	IRFC	Integer register file correctable	2	2	8
	FRFU	Floating-point register file uncorrectable	3	3	9
	FRFC	Floating-point register file correctable	4	4	9
	SBDLC	Store buffer data load hit correctable	10	5	6
	SBDLU	Store buffer data load hit uncorrectable	10	6	6
	MRAU	MRA uncorrectable	11	7	10
	TSAC	TSA correctable	11	8	11
	TSAU	TSA uncorrectable	11	9	11
	SCAC	SCA correctable	11	10	12
	SCAU	SCA uncorrectable	11	11	12
	TCCP	Tick compare correctable precise	11	12	13
	tcup	Tick compare uncorrectable precise	11	13	13

TABLE 25-8 Hardware Errors Recorded in the D-SFSR and D-SFAR Registers *(Continued)*

Trap Type	Error Name	Error Description	Relative Priority	Contents of D-SFSR ErrorType Field	Contents of D-SFAR (see TABLE 25-11
data_access_MMU_error	DTTM	DT tag multiple hit	5	1	1
	DTTP	DT tag parity	6	2	1
	DTDP	DT data parity	7	3	1
	DTMU	DT MRA uncorrectable	8	4	2
	DTL2U	DT L2 uncorrectable	9	5	3
	DTL2ND	DT L2 NotData	9	6	3
data_access_error	DCL2U	dc L2 uncorrectable	11	1	3
	DCL2ND	dc L2 NotData	11	2	3
	SOCU	SOC uncorrectable	12	4	See page 324

25.8.4.1 DMMU Synchronous Fault Status Register

Each virtual processor has one hyperprivileged, read/write, ASI_DMMU_SFSR register located at ASI 58_{16}, VA 18_{16}. The format of the DSFSR register is shown in TABLE 25-9.

TABLE 25-9 DSFSR – ASI_DMMU_SYNCHRONOUS_FAULT_STATUS_REGISTER (ASI 58_{16}, VA 18_{16})

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	errtype	0	RW	Error type information, format defined in TABLE 25-8.

Note | DSFSR is preserved across warm resets to allow software to determine the cause of an error that required a warm reset.

25.8.4.2 DMMU Synchronous Fault Address Register

Each virtual processor has one hyperprivileged, read-only, ASI_DMMU_SFAR register located at ASI 58_{16}, VA 20_{16}. The format of the DSFAR register is shown in TABLE 25-10.

TABLE 25-10 DSFAR – ASI_DMMU_SYNCHRONOUS_FAULT_ADDRESS_REGISTER (ASI 58_{16}, VA 20_{16})

Bit	Field	Initial Value	R/W	Description
63:48	—	0	RO	*Reserved*
47:0	erraddr	0	RO	Error address information, format defined in TABLE 25-11.

Note | DSFAR is preserved across warm resets to allow software to determine the cause of an error that required a warm reset.

TABLE 25-11 further describes the contents of the DSFAR or DESR registers, which contain information about either the address or syndrome of the error, for each type of error which can occur.

TABLE 25-11 Contents of D-SFAR or DESR Error Address Field for Various Error Types

Error Class in I-SFSR, D-SFSR, or DESR	Bit Index in D-SFAR or DESR	ErrorAddr Information	Remarks
1 (ITLB/DTLB)	47:0	VA{47:0} for D-SFAR	Undefined for ITLBE: VA of ITLB error is recorded in TPC[TL]; VA of DTLB error is recorded in D-SFAR.
2 (MRA access for HWTW)	47:3	—	Undefined.
	2:0	MRA index{2:0}	
3 (Memory access)	47:0	—	Undefined; L2 ESR contains more information about the error.
4 (IC access)	47:9	—	Undefined.
	8:6	Way{2:0}	One of the ways if a multiple-way hit occurred; not recorded for Icache data parity errors.
	5:0	IC index{5:0}	
5 (DC access)	47:9	—	Undefined.
	8:7	Way{1:0}	One of the ways if a multiple-way hit occurred.
	6:0	DC Index{6:0}	
6 (Store Buffer Data ECC)	47:3	—	Undefined.
	2:0	STB Index{2:0}	

TABLE 25-11 Contents of D-SFAR or DESR Error Address Field for Various Error Types *(Continued)*

Error Class in I-SFSR, D-SFSR, or DESR	Bit Index in D-SFAR or DESR	ErrorAddr Information	Remarks
8 (IRF ECC)	47:15	—	Undefined.
	14:7	Syndrome{7:0}	ECC Syndrome; see Appendix E, *ECC Codes* for ECC syndrome decode.
	6:5	GL{1:0}	GL field when error occurred.
	4:0	IRF Index{4:0}	Physical array index where the error occurred. For indices which point to %in or %out registers, the interpretation depends upon whether %cwp is pointing to an even or an odd window, as follows: For an even window: index 0 --> %g0 index 8 --> %o0 index 16 --> %l0 index 24 --> %i0 For an odd window, the indices which map to %in and %out registers swap: index 0 --> %g0 index 24 --> %o0 index 16 --> %l0 index 8 --> %i0
9 (FRF ECC)	47:20	—	Undefined.
	19:13	Even Syndrome{6:0}	Syndrome of lower 32 bits (bits 31:0); see Appendix E, *ECC Codes* for ECC syndrome decode.
	12:6	Odd Syndrome{6:0}	Syndrome of upper 32 bits (bits 63:32); see Appendix E, *ECC Codes* for ECC syndrome decode.
	5:0	FRF Index{5:0}	
10 (MRA)	47:3	—	Undefined. (No syndrome since parity is stored in MRA.)
	2:0	MRA Index{2:0}	
11 (TSA)	47:19	—	Undefined.
	18:11	Odd Syndrome{7:0}	
	10:3	Even Syndrome{7:0}	
	2:0	TSA Index{2:0}	
12 (Scratchpad)	47:3	—	Undefined.
	10:3	Syndrome{7:0}	
	2:0	SCPD Index{2:0}	

TABLE 25-11 Contents of D-SFAR or DESR Error Address Field for Various Error Types *(Continued)*

Error Class in I-SFSR, D-SFSR, or DESR	Bit Index in D-SFAR or DESR	ErrorAddr Information	Remarks
13 (Tick_compare)	47:2	—	Undefined.
	9:2	Syndrome{7:0}	
	1:0	Tick_compare Index{1:0}	

25.8.5 Disrupting Error Status Register (DESR)

Each virtual processor has a hyperprivileged, read-only DESR located at ASI $4C_{16}$, VA 0_{16}. The DESR records all disrupting errors. A read of DESR clears all fields. The format of the DESR register is shown in TABLE 25-12.

TABLE 25-12 DESR – ASI_DISRUPTING_ERROR_STATUS_REGISTER (ASI $4C_{16}$, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63	f	0	RC	Full, register contains valid data.
62	me	0	RC	Multiple errors detected.
61	s	0	RC	Error trap type. If 1, a *sw_recoverable_error* was logged. If 0, a *hw_corrected_error* was logged.
60:56	errtype	0	RC	Error type, format defined in TABLE 25-13.
55:11	—	0	RC	*Reserved*
10:0	erraddr	0	RC	Error address information, format defined in TABLE 25-11.

Note | DESR is preserved across warm resets to allow software to determine the cause of an error that required a warm reset.

DESR captures information on both *sw_recoverable_error* and *hw_corrected_error* errors. The s bit denotes the type of error whose information is logged in the errortype and erroraddr fields. If set to 1, the errortype and erroraddr fields contain information regarding a *sw_recoverable_error*; else, the errortype and erroraddr fields contain information regarding a *hw_corrected_error.* The s bit is required since OpenSPARC T2 can detect both *hw_corrected_error* and *sw_recoverable_error* simultaneously. Furthermore, in the event DESR contains error information regarding a *hw_corrected_error* and a *sw_recoverable_error* occurs, the *sw_recoverable_error* must take precedence, to allow software the opportunity to recover from the error.

Hardware sets the f bit when a disrupting error occurs and the corresponding CERER bit is set; the f bit signifies that the register contains valid data (for example, is full). The s bit is set when a *sw_recoverable_error* occurs and the s bit is not

already set. The me bit is set when an error occurs and the f bit is already set, since disrupting errors can occur while the f bit is set from a previous error. If the f bit was set from a previous *hw_corrected_error* and the current error is a *sw_recoverable_error,* the error information is captured for the *sw_recoverable_error*. Otherwise, no additional information is captured about an error that sets the me bit.

If the f bit is set and the s bit is cleared (which implies only hardware corrected and cleared errors have occurred), a *hw_corrected_error* trap will be generated if SETER.dhcce is set and either PSTATE.ie is set or HPSTATE.hpriv is clear. If not enabled, the trap remains pending while the f bit is set and the s bit is cleared until SETER.dhcce is set and either PSTATE.ie is set or HPSTATE.hpriv is clear.

If the f bit is set and the s bit is set (which implies an error that was not hardware corrected and cleared has occurred), a *sw_recoverable_error* trap will be generated if SETER.de is set and either PSTATE.ie is set or HPSTATE.hpriv is clear. If not enabled, the trap remains pending while the f bit is set and the s bit is set until SETER.de is set and either PSTATE.ie is set or HPSTATE.hpriv is clear.

A more detailed description of the DESR semantics is as follows:

1. If the f bit is clear and a *sw_recoverable_error* occurs, hardware sets the s and f bits to 1 and captures the *sw_recoverable_error* information in the errortype and erroraddr fields. If another *hw_corrected_error* or *sw_recoverable_error* occurs before software reads the DESR and clears the s and f bits, hardware sets the me bit and leaves the errortype and erroraddr fields unchanged.

2. If the f bit is clear and a *hw_corrected_error* occurs and there is no simultaneous *sw_recoverable_error,* hardware sets DESR.s to 0, DESR.f to 1, and captures the error information about the *hw_corrected_error*.

3. If the f bit is set and a *sw_recoverable_error* occurs:

   a. If the s bit is set, hardware sets the me bit, and leaves all other DESR fields unchanged.

   b. If the s bit is clear, hardware overwrites the DESR as follows. It sets the s, f, and me bits to 1 and overwrites the contents of the errortype and erroraddr fields with the information about the *sw_recoverable_error*. In this case, software could find that hardware took a *hw_corrected_error* trap and upon reading the DESR, realize that a *sw_recoverable_error* occurred which overwrote the *hw_corrected_error*. All detailed information about the *hw_corrected_error* is lost.

4. If the f bit is set, and a *hw_corrected_error* occurs, hardware sets the me bit and leaves the other DESR fields unchanged.

5. If a *hw_corrected_error* and a *sw_recoverable_error* occur simultaneously, the *sw_recoverable_error* takes precedence. Hardware sets the f, s, and me bits to 1 and records the *sw_recoverable_error* in the DESR errortype and erroraddr fields. The *sw_recoverable_error* exception becomes pending. The pending *hw_corrected_error* exception is cleared.

6. If software performs an ASI read of the DESR simultaneously with hardware reporting a new disrupting exception:

   a. Hardware returns the value previously stored in the DESR to the ASI read.

   b. Hardware stores the new disrupting exception in the DESR as if the f bit were clear.

Programming Notes

ASI reads of the DESR clear the DESR. Consequently, hardware clears the corresponding f bit in the CLESR as well.

The DESR is read-only. To test error recovery code, software must inject an error (using the `ASI_CORE_ERROR_INJECT` register), then cause the erroneous data to be used, thereby taking a *hw_corrected_error* or *sw_recoverable_error* trap.

The DESR errortype field contains information that describes the error and is detailed in TABLE 25-13 below. If multiple disrupting errors occur at the same time, hardware prioritizes the errors and stores the syndrome for the highest-priority error in the errortype field, according to the column labeled Relative Priority, with 1 being the highest priority.

TABLE 25-13 DESR Error Type Bits

Trap Type	Error Name	Description	Relative Priority	Contents of DESR errortype Field (bits 60:56)	Contents of DESR erroraddr Field (see TABLE 25-11)
hw_corrected_error	ICVP	Instruction cache valid bit parity.	11.1	1	4
	ICTP	Instruction cache tag parity.	11.2	2	4
	ICTM	Instruction cache tag multiple.	11.3	3	4
	ICDP	Instruction cache data parity.	11.4	4	4
	DCVP	Data cache valid bit parity.	12.1	5	5
	DCTP	Data cache tag parity.	12.2	6	5
	DCTM	Data cache tag multiple.	12.3	7	5
	DCDP	Data cache data parity.	12.4	8	5
	L2C	L2 cache correctable.	13	9	3
	SBDPC	Store buffer data PCX read correctable ECC.	14	10	6
	SOCC	SOC correctable.	15	11	(see SOC)

TABLE 25-13 DESR Error Type Bits *(Continued)*

Trap Type	Error Name	Description	Relative Priority	Contents of DESR errortype Field (bits 60:56)	Contents of DESR erroraddr Field (see TABLE 25-11)
	SBDPU	Store buffer data PCX read uncorrectable ECC.	1	6	6
	TCCD	Tick_compare correctable disrupting.	2	14	13
	TCUD	Tick_compare uncorrectable disrupting.	2	15	13
	L2C	L2 correctable ECC error.	5	20	3
	L2U	L2 uncorrectable ECC error.	5	16	3
	L2ND	L2 NotData error.	5	17	3
	ITL2C	IT L2 correctable.	6	1	3
	ICL2C	ic L2 correctable.	6	2	3
	DTL2C	DT L2 correctable.	6	3	3
sw_recoverable_error	DCL2C	dc L2 correctable.	6	4	3
	SOCU	SOC uncorrectable.	7	19	See Section 25.17

25.8.6 Deferred Error Status Register (DFESR)

Each virtual processor has a hyperprivileged, read-only-and-clear DFESR at ASI $4C_{16}$, VA 8_{16}. A read of DFESR clears all fields to 0. DFESR records deferred errors detected by the OpenSPARC T2 core. The format of the Deferred Error Status register is shown in TABLE 25-14.

TABLE 25-14 DFESR – ASI_DEFERRED_ERROR_STATUS_REGISTER (ASI $4C_{16}$, VA 8_{16})

Bit	Field	Initial Value	R/W	Description
63:62	—	0	RO	*Reserved*
61:60	type	0	RC	Error type, format defined in TABLE 25-15.
59:58	priv	0	RC	Privilege level, format defined in TABLE 25-16.
57:55	stbindex	0	RC	Store buffer index.
54:0	—	0	RO	*Reserved*

Note | DFESR is preserved across warm resets to allow software to determine the cause of an error that required a warm reset.

The content of the error type field is as follows:

TABLE 25-15 DFESR ErrorType

Bit Index in FESR	Field Name	Error	Remarks
61	sba	SBAPP	Store buffer address parity error.
60	sbdio	SBDIOU	Store buffer data I/O uncorrectable error.

The content of the privilege-level field is as follows:

TABLE 25-16 Privilege Level Field Contents

Bit Index in FESR	Field Name	Interpretation	Remarks
59:58	priv	Highest privilege level of any store in the store buffer	00 - User (hpriv = priv = 0). 01 - Supervisor (hpriv = 0, priv = 1). 10 - Hpriv (hpriv = 1). 11 - Error in privilege level field.

Using the privilege level field, software can determine an appropriate response to a fatal thread error. For example, if a user-level store received the error, software may kill only the user process, instead of taking down the entire chip or partition. An encoding of 11 in the privilege level field means that there was an error in this field in the store buffer contents, so the privilege level of the store cannot be determined.

Bits 57:55 of the DFESR contain the index in the store buffer which had the error.

Programming Note Hardware cannot support simultaneous reads of the DFESR and updates of the DFESR contents (which can occur due to a previously issued store to any address). Therefore, software must not attempt a store in the store error trap handler until it has read the DFESR (and implicitly cleared the DFESR).

25.8.7 ASI_CLESR

Each physical core has a shared, read-only, hyperprivileged Core Local Error Status register located at ASI $4C_{16}$, VA 20_{16}.

TABLE 25-17 CLESR Contents

T7	T6	T5	T4	T3	T2	T1	T0	*Reserved*
63:62	61:60	59:58	57:56	55:54	53:52	51:50	49:48	47:0

Each of the eight fields T7–T0 contains 2 bits. The high-order bit is a copy of the DESR.f bit for that strand; the low-order bit is the **or** of the two DFESR.errortype bits for that strand. For example, CLESR bit 54 is the **or** of the DFESR.errortype bits for strand 3.

25.8.8 ASI_CLFESR

Each physical core has a shared, read-only, hyperprivileged Core Local First Error Status register located at ASI $4C_{16}$, VA 28_{16}. CLFESR logs the first error recorded by a physical core's strands DESRs and DFESRs in CLESR. Thus, there is a one-to-one correspondence between the bits in the CLESR and CLFESR. When CLESR is clear, any error that is recorded in a strand's DESR or DFESR causes the error to be logged in CLESR and CLFESR. If multiple strands log an error in their respective DESR or DFESR simultaneously, then multiple bits can be set in the CLFESR.

Once the CLFESR has recorded an error, it will not log any subsequent error until software clears all f bits in all the physical core's strands' DESRs and errortype bits in the DFESRs. At this point the CLESR will be all zeroes, and hardware clears the CLFESR.

TABLE 25-18 CLFESR contents

T7	T6	T5	T4	T3	T2	T1	T0	*Reserved*
63:62	61:60	59:58	57:56	55:54	53:52	51:50	49:48	47:0

25.8.9 ASI_ERROR_INJECT_REG

Each physical core has a hyperprivileged ASI_ERROR_INJECT_REG register at ASI: 43_{16}, VA{63:0} = 0. The ASI_ERROR_INJECT register enables software to inject errors into a subset of the OpenSPARC T2 core's error protected arrays. The subset includes all error-protected arrays except for the Icache and Dcache valid bit, tag, and data arrays. The format of the Error Injection register is shown in TABLE 25-19.

TABLE 25-19 Error Injection Register – ASI_ERROR_INJECT_REGISTER (ASI 43_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31	enb_hp	0	RW	If 1, enable error injection; else disable error injection.
30	—	0	RO	*Reserved*
29	imdu	0	RW	Inject ITLB data array parity error on an ITLB update.
28	imtu	0	RW	Inject ITLB CAM parity error on an ITLB update.
27	dmdu	0	RW	Inject DTLB data array parity error on a DTLB update.

TABLE 25-19 Error Injection Register – ASI_ERROR_INJECT_REGISTER (ASI 43_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
26	dmtu	0	RW	Inject DTLB CAM parity error on a DTLB update.
25	ircu	0	RW	Inject an IRF ECC error on any IRF write.
24	frcu	0	RW	Inject an FRF ECC error on any FRF write.
23	scau	0	RW	Inject an ECC error into the Scratchpad array.
22	tcup	0	RW	Inject an ECC error into the TICK_CMPR array.
21	tsau	0	RW	Inject an ECC error into the TSA array.
20	mrau	0	RW	Inject a parity error into the MRA array.
19	stau	0	RW	Inject a parity error into the store buffer CAM.
18	—	0	RO	*Reserved*
17	stdu	0	RW	Inject an ECC error into the Store Buffer Data array.
16:8	—	0	RO	*Reserved*
7:0	eccmask	0	RW	Mask of bits to be **xor**ed with ECC bits for the IRF, FRF, SCA, TCA, TSA, or STB data arrays.

Setting more than one error source bit active at a time produces undefined results. In order to inject an error, both the enb_hp bit and one of the error source bits (bits 29:17) must be set.

Only one strand should be active at a time when injecting errors, since hardware may schedule a strand at any time and that strand may inject errors or receive the effect of errors generated by another strand, complicating diagnosis.

Writes to ASI_ERROR_INJECT are actually post-synchronizing, so no MEMBAR #Sync is required after the write. However, there are cases where writes to ASI_ERROR_INJECT are not presynchronizing (that is, an instruction *before* the write could see the effect). To guard against this, software needs to put a MEMBAR #Sync *before* a write to ASI_ERROR_INJECT.

25.9 L2 Cache Error Descriptions

The L2 cache protects its tag with SEC ECC. Each 32-bit data subline is protected with SECDED ECC. In the following L2 cache behavior descriptions, a partial store refers to a store that updates less than the full 32 bits of any 32-bit data subline.

Errors detected in the L2 cache and also the DRAMs are reported back to the requesting cores using two packet types. L2_Load_Return packets are sent by the L2 cache to the requesting core for synchronous requests such as loads and prefetches, and Error_Indication_(L2) packets for asynchronous errors such as store, writeback, and scrub errors.

The flow of errors detected in the L2 cache or detected in the DRAM and passed to the L2 cache is shown in FIGURE 25-1 for synchronous (load) requests to the L2 cache from the requesting core. Load request can be made from the SPARC core for load, instruction fetches and TTE requests. The errors are presented in the 2-bit err field of the L2_Load_Return packet, indicating correctable (01), uncorrectable (10) and NotData (11).

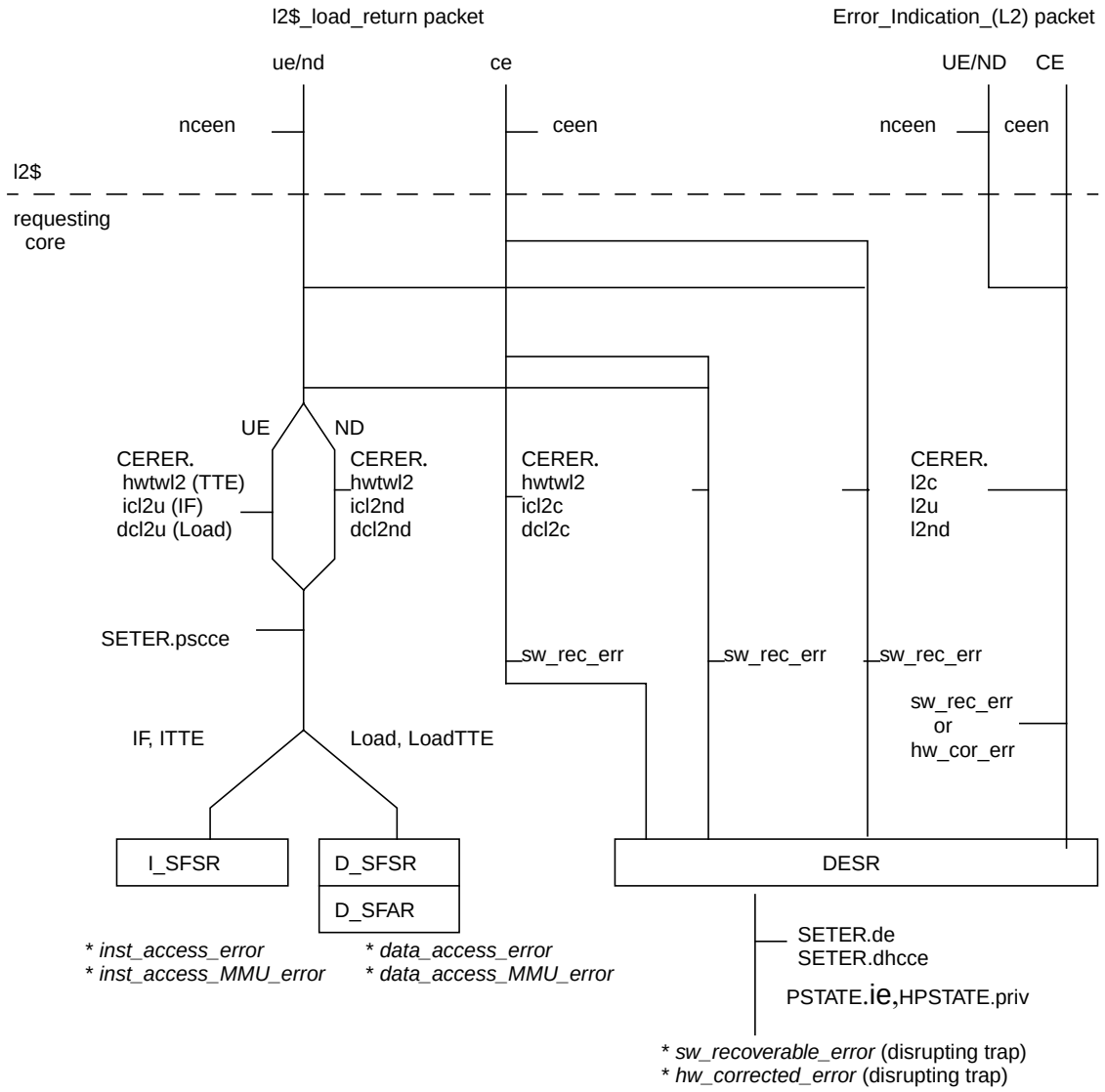


FIGURE 25-1 L2$ Error Flow for Errors Detected by L2$/DRAM

The flow of errors detected in the L2 cache or detected in the DRAM and passed to the L2 cache is shown in FIGURE 25-1 for asynchronous errors from the L2 cache to the requesting core or the L2_CSR_REG.errorsteer core. These errors include store writeback, scrub, DMA access, and prefetch errors. The errors are presented in the 2-bit err field of the Error_Indication_(L2) packet, indicating correctable (01), uncorrectable (10) and NotData (11).

Uncorrectable and NotData errors presented in the L2_Load_Return packet for loads requested by the core result in precise traps. Correctable errors on core loads result in *sw_recoverable_error* disrupting traps.

Uncorrectable errors and some correctable errors (DMA Read and Prefetch) presented in the Error_Indication_(L2) packet result in *sw_recoverable_error* disrupting traps. The other correctable errors presented in the Error_Indication_(L2) packet result in *hw_corrected_error* disrupting traps.

This section details the errors that occur from L2 hits. DRAM accesses are dealt with in Section 25.11 on page 293.

The L2 cache error detection and reporting mechanism is functional whether the L2 cache is enabled or disabled. The only difference is that L2 cache data and tag array errors are not detected since the arrays are disabled.

25.9.1 L2 Cache Data Correctable ECC Error for Access (LDAC)

Correctable data errors are detected by the L2 cache, captured in the L2 Cache Error Status register and the L2 Cache Error Address register, and then reported to the requesting virtual processor. The following control bits are used:

L2 Cache Error Enable ceen bit — When set, causes the error to be sent to the requesting virtual processor; when reset, error is ignored but still captured in the L2 Cache Error Status register and the L2 Cache Error Address register.

Associated CERER bit — When set causes L2$ error (assuming ceen = 1) to be captured in DESR register, and conditions reporting of disrupting trap condition; when disabled, L2$ error is ignored, and desr is not changed.

SETER.de bit — When set (along with ceen and associated CERER bit), causes disrupting *sw_recoverable_error* trap condition to be presented to the virtual processor for load requests that use the L2_Load_Return packet. The disrupting trap is further conditioned by HPSTATE.hpriv and PSTATE.ie. When SETER.de/dhcce is reset, the disrupting trap condition remains pending. When SETER.de/dhcce is later set, the disrupting trap is taken, conditioned by HPSTATE.hpriv and PSTATE.ie.

SETER.dhcce bit — When set (along with ceen and associated CERER bit), causes disrupting *hw_corrected_error* trap condition to be presented to the virtual processor for store requests that use the Error Indication L2) packet. The disrupting trap is further conditioned by HPSTATE.hpriv and PSTATE.ie. When SETER.de/dhcce is reset, the disrupting trap condition remains pending. When SETER.de/dhcce is later set, the disrupting trap is taken, conditioned by HPSTATE.hpriv and PSTATE.ie.

The error information stored in the L2 Cache Error Status register is shown in TABLE 25-21 on page 283, and the physical address captured in the L2 Cache Error Address register is shown in TABLE 25-25 on page 290. They are recorded as LDAC errors.

25.9.1.1 TTE Request for ITLB (ITL2C)

When a correctable ECC error is detected, the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen, and SPARC CERER.hwtwl2 and SETER.de are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware generates a disrupting *sw_recoverable_error* trap to the requesting virtual processor. Assuming CERER.hwtwl2 is set and the DESR.f bit is clear, hardware encodes itl2c in the DESR.errortype field. The DESR.erroraddr field is undefined. Hardware corrects the error on the data being returned from the L2 cache but does not correct the L2 cache data itself. Software can correct the L2 error by invalidating the L2 line as appropriate.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.2 TTE Request for DTLB (DTL2C)

When a correctable ECC error is detected the error information (ldac, vcid, synd) is captured in the L2 Cache Error Status and PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen, and SPARC CERER.hwtwl2 and SETER.de are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware generates a disrupting *sw_recoverable_error* trap to the requesting virtual processor. Assuming CERER.hwtwl2 is set, and the DESR.f bit is clear, hardware encodes dtl2c in the DESR.errortype field. The DESR.erroraddr field is undefined. Hardware corrects the error on the data being returned from the L2 cache but does not correct the L2 cache data itself. Software can correct the L2 error by invalidating the L2 line as appropriate.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.3 Instruction Fetch Hit (ICL2C)

When a correctable ECC error is detected the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen,

and SPARC CERER.icl2c and SETER.de bits are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware generates a disrupting *sw_recoverable_error* trap to the requesting virtual processor. Assuming CERER.icl2c is set, and the DESR.f bit is clear, hardware encodes icl2c in the DESR.errortype field. The DESR.erroraddr field is undefined. Hardware corrects the error on the data being returned from the L2 cache but does not correct the L2 cache data itself. Software can correct the L2 error by invalidating the L2 line as appropriate.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.4 Load Hit (DCL2C)

When a correctable ECC error is detected the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen, and SPARC CERER.dcl2c and SETER.de bits are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware generates a disrupting *sw_recoverable_error* trap to the requesting virtual processor. Assuming CERER.dcl2c is set and the DESR.f bit is clear, hardware encodes dcl2c in the DESR.errortype field. The DESR.erroraddr field is undefined. Hardware corrects the error on the data being returned from the L2 cache, but does not correct the L2 cache data itself. Software can correct the L2 error by invalidating the L2 line as appropriate.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.5 Prefetch Hit (L2C)

When a correctable ECC data error is detected the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen and SPARC CERER.l2c_socc and SETER.de bits are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), hardware generates a disrupting *sw_recoverable_error* trap to the requesting virtual processor. Assuming the DESR.f bit is clear, hardware encodes l2c in the DESR.errortype field. The contents of the DESR.erroraddr field are undefined.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.6 Partial Store Hit (L2C)

When a correctable ECC error is detected the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status and L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen and SPARC CERER.l2c_socc and SETER.dhcce bits are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *hw_corrected_error* trap is generated to the requesting virtual processor. Assuming CERER.l2c_socc is set and the DESR.f bit is clear, hardware encodes L2C in the DESR.errortype field. The DESR.erroraddr field is undefined. Hardware corrects the error in the L2 cache and returns the corrected data.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.1.7 Atomic Hit (DCL2C)

When a correctable ECC error is detected on the read portion of the read-modify-write operation, the error information (ldac, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen and SPARC CERER.dcl2c and SETER.de bits are set (and PSTATE.ie is set or HPSTATE.hpriv is clear), a disrupting *sw_recoverable_error* trap is generated to the requesting virtual processor. Assuming CERER.dcl2c is set, and the DESR.f bit is clear, hardware encodes dcl2c in the DESR.errortype field. The DESR.erroraddr field is undefined.

Hardware may or may not have corrected the error, depending upon whether the error was completely contained in the data being updated as a result of a successful atomic operation. For example, if a CASA operation succeeds, and a bit in the 4 bytes updated by CASA had the correctable error, then hardware corrected the error simply by overwriting the data in error. The same is true for a successful LDSTUB (on the byte in error being overwritten) or SWAP (on the 4 bytes in error being overwritten) or CASXA (on the 8 bytes being overwritten).

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.2 L2 Cache Data Correctable ECC Error for Writeback (LDWC)

When a correctable ECC error is detected the error information (ldwc) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. The synd field is not captured in the L2 Cache Error Status register. Hardware corrects the error on the data being written to memory. In addition, if the L2 Cache Error Enable ceen bit is set, an L2C error is reported to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set, and the DESR.f bit is clear, hardware encodes L2C in the DESR.errortype field. The DESR.erroraddr field is undefined. If SETER.dhcce is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *hw_corrected_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

> If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.2.1 DMA Read

When a correctable ECC error is detected, the error information (ldrc, rw) is captured in the L2 Cache Error Status and the PA{39:6} is captured in the L2 Cache Error Address register. Hardware corrects the error on the data being returned from the L2 cache but does not correct the L2 cache data itself. In addition, if the L2 Cache Error Enable ceen bit is set, an L2C error is reported to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set and if the DESR.f bit is clear, hardware records the error in the DESR by encoding L2C. The contents of the DESR.erroraddr field are undefined. If SETER.de is also set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware takes a disrupting *sw_recoverable_error* trap.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.2.2 DMA Write Partial

When a correctable ECC error is detected on the read portion of the read-modify-write operation, the error information (ldrc, rw, synd) is captured in the L2 Cache Error Status register and the PA{39:4} is captured in the L2 Cache Error Address register. Hardware corrects the error in the L2 cache line. In addition, if the L2 Cache Error Enable ceen bit is set, an L2C error is reported to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set and the DESR.f bit is clear, hardware records the error in the DESR by encoding l2c. The contents of the DESR.erroraddr field are undefined. If SETER.dhcce is also set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *hw_corrected_error* trap is generated.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

L2 Data ECC is only checked for partial DMA stores. Aligned 4-byte and 8-byte (and larger) DMA writes overwrite previous contents and therefore never check data ECC.

If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.3 L2 Cache Data Correctable ECC Error for Scrub (LDSC)

When a correctable ECC error is detected on a data array scrub, the error information (ldsc) is captured in the L2 Cache Error Status register and the {way[3:0],Index[8:0]} is captured in the L2 Cache Error Address register. Hardware corrects the error in the L2 cache line by writing back the corrected data and parity (this rewrite will be unable to correct a permanently failed bit). In addition, if the L2 Cache Error Enable ceen bit is set, an L2C error is reported to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set and the DESR.f bit is clear, hardware records the error in the DESR by encoding l2c. The

contents of the DESR.erroraddr field are undefined. If SETER.dhcce is also set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *hw_corrected_error* trap is generated.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note | If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.4 L2 Cache Tag Correctable ECC Error (LTC)

OpenSPARC T2 provides SEC ECC on the L2 cache tags.

On every L2 access, ECC is checked for all 16 tags in the set. When a correctable ECC error is detected the error information (LTC) is captured in the L2 Cache Error Status register and the PA{21:6} is captured in the L2 Cache Error Address register. Note that the Syndrome is *not* captured for a L2 cache tag ECC error. Hardware corrects all errors in all the tags in the set. In addition, if the L2 Cache Error Enable ceen bit is set, hardware generates an L2C error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set and the DESR.f bit is clear, hardware records the error in the DESR by setting L2C. The contents of the DESR.erroraddr field are undefined. If SETER.dhcce is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *hw_corrected_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Implementation Notes | Hardware generates the error report to the core on detection of the error .However the error would get corrected by hardware only if the access that detected the error was a miss in L2. Since the error can be reported multiple times on L2 tag hits and not get corrected, software should force the correction by issuing Prefetch ICE instruction to that index . If the error is due to a hard failure in the tag, hardware or software will not be able to complete the correction and no further accesses will be able to be processed by the L2 (that is, the L2 bank is hung.)

Note | If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.5 L2 Cache VUAD Correctable ECC Error (LVC)

On every L2 access, ECC is checked for all 32 vd (valid, dirty) bits and 32 ua (used, allocate) bits in the set.[1] When a correctable ECC single bit error is detected, the error information (LVC, SYND) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. Note that the Syndrome is captured for a L2 cache VUAD ECC error. Hardware corrects the single bit error in all the vd and ua bits in the set. In addition, if the L2 Cache Error Enable ceen bit is set, hardware generates an L2C error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2c_socc bit is set, and the DESR.f bit is clear, hardware records the error in the DESR by setting l2c. The contents of the DESR.erroraddr field are undefined. If SETER.dhcce is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *hw_corrected_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note | If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

Implementation Note | For instructions that require multiple passes through the L2 pipeline, VUAD ECC is only checked on the first pass.

25.9.6 L2 Cache Data Uncorrectable ECC Error for Access (LDAU)

Uncorrectable data errors are detected by the L2 Cache, captured in the L2 Cache Error Status register and the L2 Cache Error Address register, and then reported to the requesting virtual processor. The following control bits are used:

1. Even though the used bits need not be ECC protected (since their value is noncritical: any error in the used bits will cause potentially different replacement order, but still functionally correct operation), since vd and ua arrays are implemented out of the same Register File array, OpenSPARC T2 protects both the used bits and the allocate bits with ECC.

L2 Cache Error Enable nceen bit — When set, causes the error to be sent to the requesting virtual processor; when reset, error is ignored, but still captured in the L2 Cache Error Status register and the L2 Cache Error Address register.

Associated CERER bit — When set (along with nceen and SETER.pscce bit) causes L2$ error to be captured in I/DSFSR register, and conditions reporting the precise error trap; when disabled, L2$ error is ignored, I/DSFSR is not changed, the precise error trap is lost, and processing continues with bad data.

SETER.pscce bit — When set (along with nceen and associated CERER bit), causes L2$ error to be captured in I/DSFSR register, and conditions reporting the precise error trap to be presented to virtual processor. When SETER.pscce is reset, the precise error trap is lost, and processing continues with bad data.

The error information stored in the L2Cache Error Status register is shown in TABLE 25-21, and the physical address captured in the L2 Cache Error Address register is shown in TABLE 25-25. They are recorded as LDAU errors.

25.9.6.1 TTE Request for ITLB (ITL2U)

When an uncorrectable ECC data error is detected, the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.hwtwl2 bits are set, a precise *instruction_access_MMU_error* trap is generated to the requesting virtual processor. The ISFSR will have the itl2u encoding set. The VA of the instruction fetch is available in TPC[TL]. Software can attempt recovery.

Note | If nceen or CERER.hwtwl2 are not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.6.2 TTE Request for DTLB (DTL2U)

When an uncorrectable ECC data error is detected, the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable ceen and SPARC CERER.hwtwl2 bits are set, a precise *data_access_MMU_error* trap is generated to the requesting virtual processor. The DSFSR will have the dtl2u encoding set. The VA of the data access is *not* logged in the D-SFAR. Bits {47:13} of the VA are available in the tag access register. Software must search the L2 ESRs to determine the failing physical address.

Note | If nceen or CERER.hwtwl2 are not set, the error is not reported and no trap is generated. Hardware continues executing with the (invalid) data read from L2.

25.9.6.3 Instruction Fetch Hit (ICL2U)

When an uncorrectable ECC data error is detected, the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. If the L2 Cache Error Enable nceen and SPARC CERER.icl2u bits are set, the line is loaded into the L1 instruction cache with bad parity. In addition, if the SPARC SETER.pscce bit is set, a precise *instruction_access_error* trap is generated to the requesting virtual processor. The ISFSR will contain icl2u, and the VA of the instruction fetch is logged in TPC[TL]. Software can attempt recovery.

Note | If nceen, CERER.icl2u, or SETER.pscce are not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.6.4 Load Hit (DCL2U)

When an uncorrectable ECC data error is detected, the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. If the L2 Cache Error Enable nceen and SPARC CERER.dcl2u bits are set, the line is loaded into the L1 data cache with bad parity. Additionally, if the SPARC SETER.pscce bit is set, a precise *data_access_error* trap is generated to the requesting virtual processor. The D-SFSR will contain dcl2u. The VA of the data access is *not* logged in the D-SFAR. Software must search the L2 ESRs to determine the failing physical address.

Note | If nceen, CERER.dcl2u, or SETER.pscce is not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.6.5 Atomic Hit (DCL2U)

When an uncorrectable ECC data error is detected, the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable nceen, and SPARC CERER.dcl2u and SETER.pscce bits are set, hardware records the error in the DSFSR by encoding dcl2u. The VA of the data access is *not* logged in the D-SFAR. Software must search the L2 ESRs to determine the failing physical address. Hardware generates a precise *data_access_error* trap to the requesting virtual processor. The atomic operation will not complete its update, leaving the original data and the bad ECC unchanged. Software can attempt recovery.

Note | If nceen, CERER.dcl2u, or SETER.pscce is not set, hardware neither records the error in the DSFSR nor takes a trap.

25.9.6.6 Prefetch Hit (L2U)

When an uncorrectable ECC data error is detected the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.l2u_socu are set and the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the requesting virtual processor.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.6.7 Partial Store Hit (L2U)

When an uncorrectable ECC data error is detected on the read portion of the read-modify-write operation (assuming NotData is not set), the error information (ldau, rw, vcid, moda, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.l2u_socu bits are set, and if the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the requesting virtual processor. The partial store does not complete its update, leaving the original data and the bad ECC unchanged. NotData will *not* be set.

If SETER.dhcce is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note | If either the L2 Cache Error Enable nceen or CERER.l2u_socu bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.7 L2 Cache Data Uncorrectable ECC Error for Writeback (LDWU)

When an uncorrectable ECC error is detected the error information (ldwu) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. The synd field is not captured in the L2 Cache Error

Status register. Hardware indicates the error on the data being written to memory, and the DRAM controller writes the data back with signaling ECC. The specific signaling ECC used is described in Appendix E, *ECC Codes*. In addition, if the L2 Cache Error Enable nceen is set, hardware generates an L2U error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2u_socu bit is set, and if the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note | If either the L2 Cache Error Enable nceen or CERER.l2u_socu bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.8 L2 Cache Data Uncorrectable ECC Error for DMA (LDRU)

25.9.8.1 DMA Read

When an uncorrectable ECC error is detected, the error information (ldru, rw) is captured in the L2 Cache Error Status register and the PA{39:4} is captured in the L2 Cache Error Address register. The synd field is not captured for DMA reads. Hardware returns the data with a UE error indicator back to the DMA requestor. In addition, if the L2 Cache Error Enable nceen bit is set, hardware generates an L2U error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2u_socu bit is set, and the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note | If either the L2 Cache Error Enable nceen or CERER.l2u_socu bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.8.2 DMA Write Partial

When an uncorrectable ECC error is detected on the read portion of the read-modify-write operation, the error information (ldru, rw, synd) is captured in the L2 Cache Error Status register and the PA{39:4} is captured in the L2 Cache Error Address register. If the L2 Cache Error Enable nceen bit is set, hardware generates an L2U error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

The DMA write partial will not complete its update, leaving the original data and the bad ECC unchanged. NotData is *not* set.

Note | If either the L2 Cache Error Enable nceen or CERER.l2u_socu bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.9 L2 Cache Data Uncorrectable ECC Error for Scrub (LDSU)

When an uncorrectable data ECC error is detected on a data array scrub, the error information (ldsu) is captured in the L2 Cache Error Status register and the {way[3:0],Index[8:0]} is captured in the L2 Cache Error Address register. In addition, if the L2 Cache Error Enable nceen bit is set, hardware generates an L2U error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2u_socu bit is set, and the DESR.f bit is clear, hardware records the error in the DESR by setting l2u. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

NotData is *not* set.

Note | If either the L2 Cache Error Enable nceen or CERER.l2u_socu bit is reset, the error is ignored, but the error information is still captured in the L2 Cache Error Status and L2 Cache Error Address registers.

25.9.10 L2 Cache Tag Uncorrectable ECC Error

OpenSPARC T2 provides SEC ECC on the L2 cache tags, thus multiple bit ECC errors are not detected (unless they happen to generate an illegal single-bit error syndrome). An L2 Cache scrub mechanism is provided to detect single bit errors and to correct the single bit errors before they can become double bit errors. However, the scrubber only works on accessed indices. If an index is not accessed for a long time, it is susceptible to double-bit errors. The probability of this is very low.

Note | To ensure that all lines are routinely accessed, software can provide a "software scrub" of each index.

25.9.11 L2 Cache VUAD Uncorrectable ECC Error (LVF)

On every L2 access, ECC is checked for all 32 vd bits and 32 ua bits in the set. When an uncorrectable ECC multiple bit error is detected, the error information (lvf, synd) is captured in the L2 Cache Error Status register and the PA{39:6} is captured in the L2 Cache Error Address register. Note that the Syndrome is captured for a L2 cache VUAD ECC error. In addition, this fatal error generates a *warm_reset* trap to the entire chip.

Implementation Note | For instructions that require multiple passes through the L2 pipeline, VUAD ECC is only checked on the first pass.

25.9.12 L2 Cache Directory Uncorrectable Parity Error (LRF)

During every L2 store operation, parity is checked for the directory entry. When an uncorrectable parity error is detected, the error information LRF is captured in the L2 Cache Error Status register. The L2 Cache Error Address register captures the following directory index information in the address field:

- bits{15:12} — panel
- bits{11:9} — strandid
- bits{8:7} — l1way{1:0}
- bit{6} — icachedir

If icachedir = 1, panel = PA{10, 9, 5, l1way2{2}}; else panel = PA{10, 9, 5, 4}

In addition, this fatal error generates a *warm_reset* trap to the entire chip.

25.9.13 L2 Cache Data NotData Error for Processor Access (NDSP)

NotData errors are detected by the L2 Cache, captured in the L2 NotData Error register and then reported to the requesting virtual processor. The following control bits are used:

L2 Cache Error Enable nceen bit — When set, causes the error to be sent to the requesting virtual processor; when reset, error is ignored, but still captured in the L2 NotData Error register.

Associated CERER bit — When set (along with nceen and SETER.pscce bit) causes L2$ error to be captured in I/DSFSR register, and conditions reporting the precise error trap; when disabled, L2$ error is ignored, I/DSFSR is not changed, the precise error trap is lost, and processing continues with bad data.

SETER.pscce bit — When set (along with nceen and associated CERER bit), causes L2$ error to be captured in I/DSFSR register, and conditions reporting the precise error trap to be presented to the virtual processor. When SETER.pscce is reset, the precise error trap is lost, and processing continues with bad data.

The error information stored in the L2 NotData Error register is shown in TABLE 25-27 on page 291. The errors are recorded as NDSP errors.

25.9.13.1 TTE Request for ITLB (ITL2ND)

When a NotData error is detected on a TTE access for an instruction fetch, the error information (ndsp, rw, vcid, PA{39:4}) is captured in the L2 NotData Error register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.hwtwl2 bits are set, a precise *instruction_access_MMU_error* trap is generated to the requesting virtual processor. The ISFSR will have the itl2nd encoding set. The VA of the instruction fetch is available in TPC[TL]. Software can attempt recovery.

Note | If nceen or CERER.hwtwl2 are not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.13.2 TTE Request for DTLB (DTL2ND)

When a NotData error is detected on a TTE access for a load or store, the error information (ndsp, rw, vcid, PA{39:4}) is captured in the L2 NotData Error register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.hwtwl2 bits are set, a precise *data_access_MMU_error* trap is generated to the requesting virtual processor. The DSFSR will have the dtl2nd encoding set. The VA of the data access is *not* logged in the D-SFAR. Bits 47:13 of the VA are available in the tag access register. Software must search the L2 ESRs to determine the failing physical address.

Note | If nceen or CERER.hwtwl2 are not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.13.3 Instruction Fetch (ICL2ND)

When a NotData error is detected on an instruction fetch access, the error information (ndsp, rw, vcid, PA{39:4}) is captured in the L2 NotData Error register. If the L2 Cache Error Enable nceen and SPARC CERER.icl2nd bits are set, the line is loaded into the L1 instruction cache with bad parity. In addition, if the SPARC SETER.pscce bit is set, a precise *instruction_access_error* trap is generated to the requesting virtual processor. The ISFSR will contain icl2nd, and the VA of the instruction fetch is logged in TPC[TL].

Note | If nceen, CERER.icl2nd, or SETER.pscce are not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.13.4 Load Hit (DCL2ND)

When a NotData error is detected on a load access, the error information (ndsp, rw, vcid, PA{39:4} is captured in the L2 NotData Error register. If the L2 Cache Error Enable nceen and SPARC CERER.dcl2nd bits are set, the line is loaded into the L1 data cache with bad parity. In addition, if the SPARC SETER.pscce bit is set, a precise *data_access_error* trap is generated to the requesting virtual processor. The DSFSR will contain dcl2nd. The VA of the data access is *not* logged in the D-SFAR. Software must search the L2 ESRs to determine the failing physical address.

Note | If nceen, CERER.dcl2nd, or SETER.pscce is not set, the error is not reported and no trap is generated. Hardware continues executing using the (invalid) data read from L2.

25.9.13.5 Atomic Hit (DCL2ND)

When a NotData error is detected on an atomic access, the error information (ndsp, rw, vcid, PA{39:4}) is captured in the L2 NotData Error register. In addition, if the L2 Cache Error Enable nceen, and SPARC CERER.dcl2nd and SETER.pscce bits are set, hardware records the error in the DSFSR by encoding dcl2nd. The VA of the data access is *not* logged in the D-SFAR. Software must search the L2 ESRs to determine the failing physical address. Hardware generates a precise *data_access_error* trap to the requesting virtual processor. The atomic operation will not complete its update, leaving the original data unchanged and marked with NotData.

> If nceen, CERER.dcl2nd, or SETER.pscce is not set, hardware neither records the error in the DSFSR nor takes a trap. It continues executing using the (invalid) data read from L2.

25.9.13.6 Prefetch Hit (L2ND)

When a NotData error is detected on a prefetch access, the error information (ndsp, rw, vcid, PA{39:4}) is captured in the L2 NotData Error register. In addition, if the L2 Cache Error Enable nceen and SPARC CERER.l2nd bits are set and the DESR.f bit is clear, hardware records the error in the DESR by encoding l2nd. The contents of the DESR.erroraddr field are undefined. If SETER.de is set and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the requesting virtual processor.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.13.7 Partial Store Hit (L2ND)

When a NotData error is detected on the read portion of the store, the partial store does not complete its update, leaving the original data unchanged, marked with NotData. No trap is presented, and the L2 Notdata Error register is not updated.

25.9.14 L2 Cache Data NotData Error for DMA Access (NDDM)

25.9.14.1 DMA Read (L2ND)

When a NotData error is detected on a DMA read, the error information (nddm, rw, vcid = L2_CSR_REG.errorsteer, PA{39:4}) is captured in the L2 Notdata Error register. Hardware returns the data with a UE error indicator back to the DMA requestor. In addition, if the L2 Cache Error Enable nceen bit is set, hardware generates an L2ND error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2nd bit is set, and if the DESR.f bit is clear, hardware records the error in the DESR by setting l2nd. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.9.14.2 DMA Write Partial (L2ND)

When a NotData error is detected the error information (nddm, rw, vcid = L2_CSR_REG.errorsteer, PA{39:4}) is captured in the L2 NotData Error register. If the L2 Cache Error Enable nceen bit is set, hardware generates an L2ND error to the virtual processor specified in L2_CSR_REG.errorsteer. If the SPARC CERER.l2nd bit is set, and if the DESR.f bit is clear, hardware records the error in the DESR by setting l2nd. The contents of the DESR.erroraddr field are undefined. If SETER.de is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, a disrupting *sw_recoverable_error* trap is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If SETER.de is not set, or PSTATE.ie is not set and HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

The DMA write partial will not complete its update, leaving the original data unchanged marked with NotData.

25.9.15 L2 Cache Data NotData Error for Writeback

When a NotData error is detected on an eviction, no error information is logged since the NotData is simply being moved to memory. Hardware indicates the error on the data being written to memory, where the DRAM controller will write back the data with signaling ECC.

25.9.16 L2 Software Error Scrubbing Support

Errors which are software recoverable leave the L2 cache with a correctable error, which then needs to be scrubbed to prevent repetitive traps for the same soft error. Flushing the data from the cache to memory will cause the error to be corrected. OpenSPARC T2 provides a mechanism for L2 flushing through a variant of prefetch called invalidate cache entry (PrefetchICE), described in *L2 Cache Flushing* on page 443.

25.10 L2 Error Registers

25.10.1 L2 Error Enable Register

Each L2 bank has an Error Enable register that controls the reporting of L2 errors for that bank back to the initiator of an operation (or to the virtual processor specified in L2_CSR_REG.errorsteer if no initiator exists or can be readily identified). The L2 Error Enable register, the format of which is shown in TABLE 25-20, is available at address AA 0000 0000_{16} or BA 0000 0000_{16}. Address bits 8:6 select the cache bank, and address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

TABLE 25-20 Error Enable Register – L2_ERROR_EN_REG (AA 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
63:3	—	0	RO	*Reserved*
2	debug_trig_en	0	RW	Trigger enable on L2 cache errors for debug.
1	nceen	0	RW	If set to 1, report uncorrectable errors.
0	ceen	0	RW	If set to 1, report correctable errors.

Note | Errors are always logged in the L2_ERROR_STATUS_REG and L2_ERROR_ADDRESS_REG regardless of the setting of the nceen and ceen bits in L2_ERROR_EN_REG. L2_ERROR_EN_REG only controls whether or not the error is reported back to the appropriate virtual processor (the requestor or the virtual processor specified in L2_CSR_REG.errorsteer).

25.10.2 L2 Error Status Register

Each L2 bank has an Error Status register that contains status on L2 errors for that bank. The status bits in this register are cleared by writing a 1 to the bit. The error register is not cleared on reset, so software can examine its contents after an error-induced reset. The L2 Error Status register, the format of which is shown in TABLE 25-21, is available at address AB 0000 0000_{16} or BB 0000 0000_{16}. Address bits 8:6 select the cache bank, and address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

Note | L2_ERROR_STATUS is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

TABLE 25-21 L2 Error Status Register – L2_ERROR_STATUS_REG (AB 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	POR Value	R/W	Description
63	meu	0	R/W1C	Multiple uncorrected or fatal errors (one or more uncorrected or fatal errors were not logged).
62	mec	0	R/W1C	Set by *any* error detected after vec = 1 (one or more corrected errors were not logged).
61	rw	0	RW	Specifies whether the error access was a read or write. Set to 1 for a write, 0 for a read.
60	—	0	RW	Set to 0.
59:54	vcid	0	RW	ID of virtual processor that encountered error.
53	ldac	0	R/W1C	Set to 1 if the error was an L2 cache data array access correctable error.
52	ldau	0	R/W1C	Set to 1 if the error was an L2 cache data array access uncorrectable error.
51	ldwc	0	R/W1C	Set to 1 if the error was an L2 cache data array writeback correctable error.
50	ldwu	0	R/W1C	Set to 1 if the error was an L2 cache data array writeback uncorrectable error.
49	ldrc	0	R/W1C	Set to 1 if the error was an L2 cache data array dma access correctable error.
48	ldru	0	R/W1C	Set to 1 if the error was an L2 cache data array dma access uncorrectable error.
47	ldsc	0	R/W1C	Set to 1 if the error was an L2 cache data array scrub correctable error.
46	ldsu	0	R/W1C	Set to 1 if the error was an L2 cache data array scrub uncorrectable error.

TABLE 25-21 L2 Error Status Register – L2_ERROR_STATUS_REG (AB 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	POR Value	R/W	Description
45	ltc	0	R/W1C	Set to 1 if the error was an L2 cache tag array correctable error.
44	lrf	0	R/W1C	Set to 1 if the error was an L2 cache directory uncorrectable error.
43	lvf	0	R/W1C	Set to 1 if the error was an L2 cache VUAD uncorrectable error.
42	dac	0	R/W1C	Set to 1 if the error was a DRAM access correctable error.
41	dau	0	R/W1C	Set to 1 if the error was a DRAM access uncorrectable error.
40	drc	0	R/W1C	Set to 1 if the error was a DRAM dma access correctable error.
39	dru	0	R/W1C	Set to 1 if the error was a DRAM dma access uncorrectable error.
38	dsc	0	R/W1C	Set to 1 if the error was a DRAM scrub correctable error or a recoverable DRAM link error.
37	dsu	0	R/W1C	Set to 1 if the error was a DRAM scrub uncorrectable error or an unrecoverable DRAM link error.
36	vec	0	R/W1C	Set to 1 if the register contains a valid correctable error.
35	veu	0	R/W1C	Set to 1 if the register contains a valid uncorrectable or fatal error.
34	lvc	0	R/W1C	Set to 1 if the error was a L2 cache VUAD correctable error.
33:28	—1	0	RO	*Reserved*
27:0	synd	X	RW	Parity or ECC syndrome.

The vec bit is set on all cycles where a correctable error is encountered. The veu bit is set on all cycles where an uncorrectable or fatal error is encountered.

The synd field is only valid for LVF, LVC, LDAC, LDAU, LDRU and LDRC errors. The rw bit will be set to 1 for a partial store, DMA write, and atomic load/store operation if they detect an error while performing the read part of the L2 read-modify-write operation. For DAC and DAU errors, if an L2 fill happens before the data is returned to the requestor, the DAC or DAU error will be reported to the virtual processor specified in L2_CSR_REG.errorsteer, and the rw bit will be 0. The vcid field is valid only for LDAC, LDAU, DAC, and DAU errors where the error is detected synchronous with the load or store operation. If the error is reported at the time of the L2 cache fill operation, vcid will contain 0. For all other error types, this field is not valid.

TABLE 25-22 summarizes the rw, vcid, moda, and synd fields.

TABLE 25-22 rw, vcid, moda, and synd Fields Summary

Error	rw	vcid	moda	synd
During L2 fill for DAC, DAU, DRC, DRU	X	L2_CSR_REG. errorsteer	X	X
LDAC, LDAU	1 for atomics and partial stores, else 0	VCID	0	synd_127_96{6:0}, synd_95_64{6:0}, synd_63_32{6:0}, synd_31_0{6:0}
LDWC, LDWU	X	X	X	X
LDRC, LDRU	1 for write, else 0	X	X	synd_127_96{6:0}, synd_95_64{6:0}, synd_63_32{6:0}, synd_31_0{6:0} for write, X for read
LDSC,LDSU	X	X	X	X
LTC	X	X	X	X
LRF	X	X	X	X
LVC, LVF	X	L2_CSR_REG. errorsteer	X	14'b0,synd_vd{6:0},synd_ua{6:0}
DAC, DAU	1 for atomics and stores, else 0	vcid	0	X
DRC, DRU	1 for write, else 0	X	X	X

The dsc and dsu bits are logged in L2 Error Status register to notify software to check the DRAM error registers, and are set regardless of the status of the other bits. Setting dsc and/or dsu does not cause any logging of the error address or syndrome in the L2 Error Address register. It also does not update the vec/veu/mec/meu bits.

Note | A dsc or dsu error will not be reported to the core responsible fo r handling the error until an L2 miss occurs. The L2 cannot send unsolicited packets to the cores, and the L2 miss response packet has space to report these types of errors.

Note | The syndrome is *not* logged on a L2 tag correctable error as well as for NotData errors.

With regard to the remaining ue, ce bits, if multiple errors occur in the same cycle, the meu or mec bit is set and only the highest-priority error is logged based on the following priority shown in TABLE 25-23 (errors with the same priority are mutually exclusive).

TABLE 25-23 Priority for Simultaneous FE, UE, and CE Errors

Error	Priority	Bit Set if Higher Priority Error
LVF (FE)	1	Not Applicable
LRF (FE)	2	meu
LDAU (UE)	3	meu
LDSU (UE)	3	meu
LDWU (UE)	4	meu
LDRU (UE)	5	meu
DAU (UE)	6	meu
DRU (UE)	6	meu
LVC (CE)	7	mec
LTC (CE)	8	mec
LDAC (CE)	9	mec
LDSC (CE)	9	mec
LDWC (CE)	10	mec
LDRC (CE)	11	mec
DAC (CE)	12	mec
DRC (CE)	12	mec

The syndrome, rw, moda, vcid, and address are captured for the highest-priority error in that cycle.

If FE, UE, or CE errors occur in a cycle when an error status bit is already set (indicating a previous error exists that hasn't been cleared from the error status register), the information in TABLE 25-24 applies.

TABLE 25-24 UE,CE Errors Occurring in a Cycle Where Error Status Bit Already Set

Existing Error	Error	Priority	Bit Set if Highest-Priority Error	Bit Set if Higher-Priority Error in Same Cycle
LVF/LRF	LVF	1	meu	Not Applicable
LVF/LRF	LRF	2	meu	meu
LDAU/LDSU/LDWU/LDRU/DRU/DAU	LVF	1	lvf, meu	Not Applicable
LDAU/LDSU/LDWU/LDRU/DRU/DAU	LRF	2	lrf, meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDAU	3	meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDSU	3	meu	meu

TABLE 25-24 UE,CE Errors Occurring in a Cycle Where Error Status Bit Already Set

Existing Error	Error	Priority	Bit Set if Highest-Priority Error	Bit Set if Higher-Priority Error in Same Cycle
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDWU	4	meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDRU	5	meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	DAU	6	meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	DRU	6	meu	meu
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LVC	7	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LTC	8	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDAC	9	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDSC	9	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDWC	10	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	LDRC	11	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	DRC	12	mec	mec
LVF/LRF/LDAU/LDSU/LDWU/LDRU/DRU/DAU	DAC	12	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LVF	1	lvf, mec	Not Applicable
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LRF	2	lrf, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDAU	3	ldau, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDSU	3	ldsu, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDWU	4	ldwu, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDRU	5	ldru, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	DAU	6	dau, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	DRU	6	dru, mec	meu
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LVC	7	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LTC	8	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDAC	9	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDSC	9	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDWC	10	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	LDRC	11	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	DRC	12	mec	mec
LVC/LTC/LDAC/LDSC/LDWC/LDRC/DRC/DAC	DAC	12	mec	mec

For the cases above where the "Bit Set If Highest-Priority Error" column contains a value besides mec and meu, the syndrome, rw, moda, vcid, and address for the highest-priority error in that cycle will overwrite the existing syndrome, rw, moda, vcid, and address.

Once set, error status bits are only cleared by software. Hardware will never clear a set status bit. If a software write of the error register happens on the same cycle as an error, the setting of bits by the error will be based on the register state before the write, following the rules of TABLE 25-24. The setting of fields by the error will take

precedence over the same field being updated by the write; however, fields that are not changed by the error will be updated by the write (for example, if the register has the vec and ltc bits set and software does a write to clear those bit on the same cycle as a L2 cache data uncorrectable error, the error register would end up with the veu and ldau bits set, the vec and ltc bits cleared, and the rw, moda, vcid, and synd fields would contain the values for the LDAU error). The rw, moda, vcid, and synd fields are always considered to be set by an error, even if the value they are being set to is undefined (that is, set to X in TABLE 25-22).

Note | When writing the error status register on the same cycle that another error occurs, the error status register state before the register write is applied determines which bits will be updated in the register. If the error occurring on the same cycle as the write is same or lower priority than the error currently logged in the register that is being cleared by the write, this will result in the error status register having only either the vec and mec pair of bits set (for a correctable error) or veu and meu pair of bits set (for an uncorrectable error) after both the new error and the write which clears the old error bits are applied.

Note | In case VEC and MEC bits are set and there is a write to the error status register to clear the set bits, if a UE or CE occurs in the same cycle of the write that would set MEC bit according to TABLE 25-24 on page 286, the MEC bit will not get cleared but VEC bit will get cleared. A subsequent read of the error status register would see MEC bit set without VEC bit set . Similarly , in case VEU and MEU bits are set and there is a write to the error status register to clear the set bits, if a UE occurs in the same cycle of the write that would set MEU bit according to TABLE 25-24 on page 286, the MEU bit will not get cleared but VEU will get cleared. A subsequent read of the error status register would see MEU bit set without VEU bit set.

Programming Note

To minimize the possibility of missing notification of an error, software should clear any multiple error indication as soon as possible, since OpenSPARC T2 provides no indication of the number of multiple errors represented by the multiple error bit.

An example of clearing behavior for the case where an DAC error is followed closely by an DAU error would be to first log that an DAC error was seen (which is indicated by the dac bit still being set in the Error Status register), and then to do a write to the Error Status register with bits 62, 42 and 36 set to clear the mec, vec, and dac bits. The vcid, moda, rw, and synd fields of error status would then be captured to memory or a register, the physical address for the DAU would be read from the Error Address register to memory or a register, and software could do a write to the Error Status register with bits 41 and 35 set to clear the dau and veu bits and put the error status register back in a state where it can capture full error information.

Software would then invoke the code to shoot down all the TLB entries for the page with the bad cache line, force the bad cache line from L2 to memory, and kill all processes that had access to the bad cache line. If another correctable error happened after the DAU but before the write that cleared the mec, vec, and dac bits, that error would not be logged. If another correctable error happened simultaneously to or after the write that cleared the mec, vec, and dac bits, but before or simultaneously to the write that cleared the dau and veu bits, that error would be captured by the mec and vec bits being set after the dau/veu-clearing write. If another uncorrectable error happened after the DAU, but before or simultaneously to the write that cleared the dau and veu bits, that error would be captured by the meu and veu bits being set after the dau/veu-clearing write.

25.10.3 L2 Error Address Register

Each L2 bank has an Error Address register that contains the address for the L2 error within that bank. The error register is not cleared on reset, so software can examine its contents after an error-induced reset. The L2 Error Address register is available at address AC 0000 0000_{16} or BC 0000 0000_{16}. Address bits 8:6 select the cache bank, and address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

Implementation Note

Within the L2 cache itself, PA {17:9} indicates the set, and PA {8:6} indicates the bank, if all eight banks are enabled. See *L2 Bank Enable* on page 439 for details on which bits are used to select the set and the bank if fewer than all eight L2 banks are enabled.

Programming Note | If L2_IDX_HASH_EN_STATUS.enb_hp = 1, the address stored in the L2_ERROR_ADDRESS_REG will contain the hashed PA{17:11} ← {(PA{32:28} **xor** PA{17:13}) **::** (PA{19:18} **xor** PA{12:11})}.

TABLE 25-25 shows the format of the L2 Error Address register.

TABLE 25-25 L2 Error Address Register – L2_ERROR_ADDRESS_REG (AC 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	POR Value	R/W	Description
39:4	address	X	RW	Error address.
3:0	—	0	RO	*Reserved*

Note | L2_ERROR_ADDRESS is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

TABLE 25-26 lists the bits captured for each of the FE/UE/CE error types.

TABLE 25-26 Bits Captured for Each Error Type

Error	Address Bits	Address Contents
LDAC	39:4	Physical address of quadword accessed.
LDAU	39:4	Physical address of quadword accessed.
LDWC	39:6	Physical address of cache line.
LDWU	39:6	Physical address of cache line.
LDRC (rw = 0)	39:6	Physical address of cache line.
LDRC (rw = 1)	39:4	Physical address of quadword accessed.
LDRU (rw = 0)	39:6	Physical address of cache line.
LDRU (rw = 1)	39:4	Physical address of quadword accessed.
LDSC	21:9	Cache index (21:18 way, 17:9 set).
LDSU	21:9	Cache index (21:18 way, 17:9 set).
LTC	39:4	Physical address of quadword accessed.
LRF	15:6	Directory index (15:12 panel, 11:9 CoreID, 8:7 L1way{1:0}, 6 IcacheDir). If IcacheDir = 1, panel = address{10,9,5,L1way{2}}, else panel = address{10,9,5,4}.
LVF	39:4	Physical address of quadword accessed.
LVC	39:4	Physical address of quadword accessed.

TABLE 25-26 Bits Captured for Each Error Type *(Continued)*

Error	Address Bits	Address Contents
DAC	39:6	Physical address of cache line.
DAU	39:6	Physical address of cache line.
DRC	39:6	Physical address of cache line.
DRU	39:6	Physical address of cache line.

For a given error, the unused bits in the address field are not guaranteed to be zero, and should be masked by software. This register is writable by software for register diagnostics and isn't expected to be written during normal operation. However, in the event it is written on the same cycle that an error is reported, the update from the error will take precedence over the write.

25.10.4 L2 NotData Error Register

Each L2 bank has a NotData Error register that contains the status and address for the L2 NotData error within that bank. The error register is not cleared on reset, so software can examine its contents after an error-induced reset. The L2 NotData Error register is available at address AE 0000 0000_{16} and BE 0000 0000_{16}. Address bits 8:6 select the cache bank, address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

Programming Note | If L2_IDX_HASH_EN_STATUS.enb_hp = 1, the address stored in the L2_NOTDATA_ERROR_REG will contain the hashed PA{17:11} ← {(PA{32:28} **xor** PA{17:13}) **::** (PA{19:18} **xor** PA{12:11})}.

TABLE 25-27 shows the format of the L2 NotData Error register.

TABLE 25-27 L2 NotData Error Register – L2_NOTDATA_ERROR_REG (AE 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
63:52	—	0	RO	*Reserved*
51	mend	0	R/W1C	Multiple NotData errors, one or more NotData errors were not logged.
50	rw	X	RW	Specifies whether the NotData error access was a read or write. Set to 1 for a write or atomic, 0 for a read.
49	ndsp	0	R/W1C	Set to 1 if the error was a L2 cache data array access NotData error (from SPARC core).
48	nddm	0	R/W1C	Set to 1 if the error was a L2 cache data array Dma access NotData error.
47:46	—	0	RO	*Reserved*

TABLE 25-27 L2 NotData Error Register – L2_NOTDATA_ERROR_REG (AE 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
45:40	vcid	X	RW	ID of virtual processor that encountered error for ndsp, or L2_CSR_REG.errorsteer for nddm.
39:4	address	X	RW	Error address. Physical address of quadword accessed.
3:0	—	0	RO	*Reserved*

Note | L2_NOTDATA_ERROR is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

This register is writable by software for register diagnostic reasons and isn't expected to be written during normal operation. However, in the event it is written on the same cycle that an error is reported, the update from the error will take precedence over the write.

If multiple errors occur in the same cycle, the mend bit is set and only the highest-priority error is logged based on the following priority shown in TABLE 25-28.

TABLE 25-28 Priority for Simultaneous NotData Errors

Error	Priority	Bit Set if Higher Priority Error
NDSP	1	Not Applicable
NDDM	2	mend

The rw, vcid, and address are captured for the highest-priority NotData error in that cycle.

If NotData errors occur in a cycle when an error status bit is already set (indicating a previous error exists that hasn't been cleared from the error status register), the mend bit is set.

25.10.5 L2 Error Injection Register

Each cache bank has an error injection register for use in injecting errors to test error functionality or error handling code. L2 tag, VUAD, and data array errors can be injected via the diagnostic access, so the L2 error injection register only provides for the injection of directory parity errors. Errors can be injected either single/double-shot or continuously (due to restrictions in implementation, a true single-shot mode was not possible, because in a few cases the hardware must introduce two errors back-to-back). Once the enb_hp bit set, either the first (and possibly second) subsequent operation (for sdshot = 1), or all subsequent operations (for sdshot = 0)

that cause a directory update will result in the parity of the directory entry to be inverted. When in single- or double-shot mode, after the injected error(s) are generated, the enb bit is automatically reset by the hardware to 0. The L2 Error Injection register, the format of which is shown in TABLE 25-29, is available at address AD 0000 0000_{16} or BD 0000 0000_{16}. Address bits 8:6 select the cache bank, address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

TABLE 25-29 L2 Error Injection Register – L2_ERROR_INJECT_REG (AD 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	POR Value	R/W	Description
63:2	—	X	RO	*Reserved*
1	sdshot	0	RW	Controls type of error injection. 1 = single or double shot; 0 = continuous.
0	enb_hp	0	RW	Enables directory error injection.

Note | L2_ERROR_INJECT_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.11 DRAM Error Descriptions

Each 128-bit data block in memory is protected by QEC/OED (Quad Error Correct, Octal Error Detect) ECC. This ECC code supports Extended ECC for x4 DRAM chips, where the complete failure of any aligned 4-bit block can be corrected, and any error where exactly two 4-bit blocks are in error is recognized as an uncorrectable error.

L2 cache handling of DRAM detected errors for loads is processed differently, depending on the following conditions:

- If the error is detected on the bytes of the line requested by the core (critical bytes), or in the other (non-critical) bytes of the line, and
- If the error is indicated in the L2 cache during the linefill into the L2 cache .

If the error was in the block requested for load/ifetch/prefetch (the 16-byte primary line for loads , 32-byte primary line for instruction fetches) and the error got detected before the line fill , the error is delivered to the requesting virtual processor with the data in the L2_Load_Return packet with the err field indicating correctable or uncorrectable error. The error type (DSC/DSU/DAC/DAU), rw, vcid, and moda fields are captured in the L2 Cache ESR, and the PA{39:6} is captured in the L2 Cache Error Address register. The error information (DSC/DSU/DAC/DAU/DBU/FBR/FBU) is captured in the DRAM ESR.

If the cache is filled before the data is returned to the requestor on a load/ifetch and error got delivered to the L2 cache from DRAM on that line fill, or if the error is delivered to the L2 cache from DRAM during the line fill of a TTE fetch, the error is reported to the virtual processor indicated by L2_CSR_REG.errorsteer, with the data in the Error_Indication_(L2) packet with the err field indicating correctable or uncorrectable error. The error type (DSC/DSU/DAC/DAU) field is captured in the L2 cache ESR, and the PA 39:6 is captured in the L2 Cache Error Address register. The error information (DSC/DSU/DAC/DAU/DBU/FBR/FBU) is captured in the DRAM ESR.

If the error is delivered to the L2 cache during the linefill, upto two errors could be presented: one to the strand indicated by the L2_CSR_REG.errorsteer on the line fill, and conditionally an L2 cache detected error reported on the err field of the data return packet to the requestor, when the L2 Cache is accessed for the load/ifetch/ TTE data after the line fill.

In case the line fill detected uncorrectable error from DRAM ,the subsequent L2 cache detected error can be only Notdata Error to the critical quarter-line since the Uncorrectable Error would already been reported on the line fill. In case the line fill detected correctable error from DRAM, there will not be a subsequent L2 cache detected error since the line fill would fill corrected data from DRAM.

The following errors are detected in the MCU and sent to the L2 cache, where they are returned to the requesting core/strand, using the L2_Load_Return packet/ L2_Ifetch_Return packet:

- DAC, DAU, DBU, FBR, and FBU errors related to a load/ifetch/prefetch access when the error (as indicated by the err field of the packet) is observed on critical data prior to the line fill.

The following errors are detected in the MCU and sent to the L2 cache, where they are returned to the core/strand indicated by the L2_CSR_REG.errorsteer, using the Error_Indication_(L2) packet:

- DSC and DSU errors
- DAC, DAU, DBU, FBR, and FBU errors related to a store,TTE fetch,atomic access
- DAC, DAU, DBU, FBR, and FBU errors related to a load/ifetch/prefetch access when the error (as indicated by the err field of the packet) is observed on the line fill .

The flow of DRAM errors through the L2cache is shown in FIGURE 25-2.

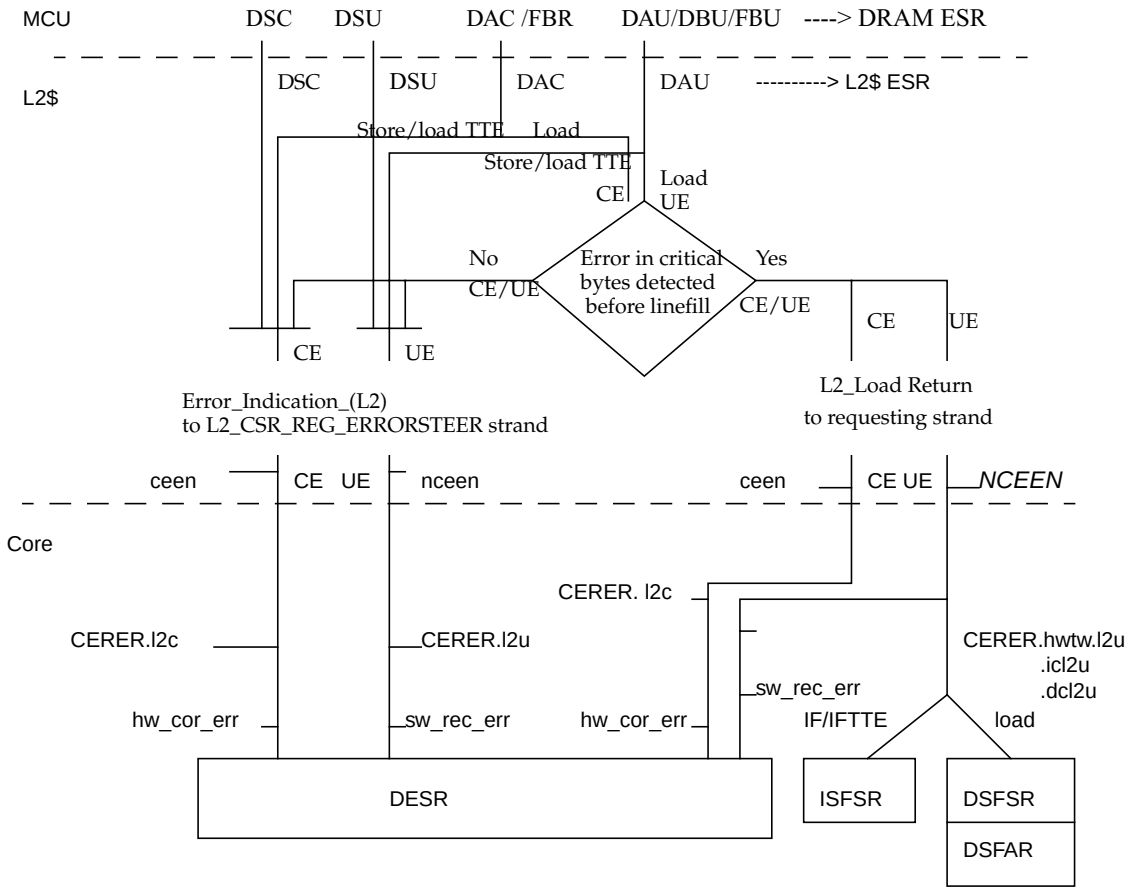


FIGURE 25-2 DRAM Error Flow From MCU to L2 Cache

Note | DRAM correctable errors (DAC, DSC, FBR) can result in two interrupts if interrupt reporting is enabled both through the L2 and through the MCU Correctable/Recoverable Error Count register described in *MCU Correctable/Recoverable Count Errors* on page 340.

25.11.1 DRAM Correctable Error for Access (DAC)

The DRAM detects correctable data ECC errors and reports them to the L2 cache as DAC errors. These errors errors are indicated by setting the dac,vec bits and the synd field in the DRAM ESR

> **Note** | If L2_CONTROL_REG.dis = 1 (the L2 cache is disabled), no logging or trap generation is performed for DRAM correctable ECC errors encountered on 32-bit and 64-bit stores.

For disrupting *sw_recoverable_error* trap conditions, if SETER.de is not set (or PSTATE.ie is clear and HPSTATE.hpriv is set), the error is still recorded in the DESR, but the trap remains pending until software sets SETER.de (and PSTATE.ie is set or HPSTATE.priv is clear).

For disrupting *hw_corrected_error* trap conditions, if SETER.dhcce is not set (or PSTATE.ie is clear and HPSTATE.hpriv is set), the error is still recorded in the DESR, but the trap remains pending until software sets SETER.dhcce (and PSTATE.ie is set or HPSTATE.priv is clear).

If either the L2 Cache Error Enable ceen or CERER.l2c_socc bit is reset, the error is ignored, but the error information is still captured in the DRAM Error Status register, the L2 Cache Error Status register, and the L2 Cache Error Address register.

These errors are logged and cause traps without regard to the ECC and FBR Error Count registers described in sections 25.12.4 and 25.12.9.

25.11.1.1 Load Miss / Instruction Fetch Miss / Prefetch Miss

The handling of the error in the L2 cache and the virtual processor depend on whether the error is reported on critical load data and whether the load occurs before the linefill.

Critical load data delivered before L2 cache linefill. When the load miss/ ifetch miss/prefetch miss request replay occurs before the line fill in the L2 cache and detects the correctable DRAM error on its critical bytes, the error information (dac,vec, rw, vcid, and moda) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

Hardware corrects the error before the data is placed into the L2 cache and returned to the virtual processor.

If the L2 Cache Error Enable ceen bit is set, an L2C error is signaled to the requesting virtual processor. The error information is delivered in the err field of an L2_Load_Return packet with the data. The requesting virtual processor indicates a *sw_recoverable_error* disrupting error condition. These errors are handled as follows:

- If the access was an Ifetch request, and CERER.icl2c is set and DESR.f is clear, hardware encodes ICL2C in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware takes a disrupting *sw_recoverable_error* trap.
- If the access was a load request, and CERER.dcl2c is set and DESR.f is clear, hardware encodes DCL2C in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware takes a disrupting *sw_recoverable_error* trap.
- In all cases, if SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Critical load data delivered after linefill. When the load miss/ifetch miss/ prefetch miss request replay occurs after the line fill in the L2 cache and correctable DRAM error is presented to the L2 cache during the line fill , the error information (dac,vec and vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable ceen bit is set, an L2C error is signaled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of an Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.dhcce bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap to the requesting virtual processor.
- If SETER.dhcce is not set, or PSTATE.ie is not set, or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.11.1.2 Atomic Miss / TTE Miss

On an atomic request or a TTE miss request, the line fill into the L2 cache with corrected data occurs first. Then the critical data is accessed from the L2 cache.

When the correctable DRAM error is presented to the L2 cache on an atomic/TTE miss request, the error information (dac,vec and vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable ceen bit is set, an L2C error is signaled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.dhcce bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap to the requesting virtual processor.
- If SETER.dhcce is not set, or PSTATE.ie is not set, or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.11.1.3 Partial Store Miss

When the correctable DRAM error is presented to the L2 cache, the error information (dac,vec and vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable ceen bit is set, an L2C error is signalled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of an Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.DHCCE bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap to the requesting virtual processor.
- If SETER.dhcce is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.11.1.4 Store Miss

When the correctable DRAM error is presented to the L2 cache, the error information (dac,vec and vcid = L2_CSR_REG.errorsteer) is also captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register. Hardware corrects the error before the data is placed into the L2 cache (the new store data will also be correct).

If the L2 Cache Error Enable ceen bit is set, an L2C error is signaled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of an Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.dhcce bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap to the requesting virtual processor.
- If SETER.dhcce is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

If the L2 Cache is disabled (L2Control register dis bit is set), no logging occurs in the L2 Cache Error Status register and L2 Cache Error Address register, and no trap is presented.

25.11.1.5 DMA Read (DRC/DAC)

When a correctable error is presented to the L2 cache, the error information (drc,vec and rw) bits are captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register. Hardware corrects the error and returns it to the DMA requestor. In addition, if the L2 Cache Error Enable ceen bit is set, hardware generates an L2C error to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *sw_recoverable_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.
- If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

L2 Error status reports DRC (DMA correctable) for this error, while DRAM reports DAC.

25.11.1.6 DMA Write Partial (DRC/DAC)

When a correctable error is presented to the L2 cache, the error information (drc,vec and rw) bits are captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register. Hardware corrects the error in the L2 cache line. In

addition, if the L2 Cache Error Enable ceen bit is set, hardware generates an L2C error to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the SPARC CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.dhcce bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap to the virtual processor specified in L2_CSR_REG.errorsteer.
- If SETER.dhcce is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

L2 Error status reports DRC (DMA correctable) for this error, while memory reports DAC.

25.11.2 DRAM Correctable ECC Error for Scrub (DSC/FBR)

The DRAM controller detects two types of correctable errors that are reported to the L2 as DSC:

- Correctable ECC error found during a scrub, the error information (dsc, synd) is captured in the DRAM Error Status register, and PA{39:4} in the DRAM Error Address register.
- Recoverable FBD link error associated with DRAM accesses. These errors are indicated by the fbr bit set in the DRAM ESR. The error information is captured in the DRAM FDB Error Syndrome register.

When a correctable scrub error is presented to the L2 cache, DSC bit is set in the L2 Cache Error Status register. Hardware corrects the error in memory by writing back the corrected data and parity (this rewrite will be unable to correct a permanently failed bit). In addition, if the L2 Cache Error Enable ceen bit is set, hardware generates an L2C error to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *hw_corrected_error* disrupting error condition. These errors are handled as follows:

- If the CERER.l2c_socc bit is set and DESR.f is clear, hardware encodes l2c in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.dhcce bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *hw_corrected_error* trap.

- If SETER.dhcce is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.dhcce and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note that the dsc bit is set in the L2 Cache ESR regardless of the status of any other bits in the register. The dsc bit is logged to notify software to check the DRAM error register.

25.11.2.1 FBD Channel Recoverable Error (FBR)

There are four FBDIMM channel recoverable error conditions that the DRAM controller detects: CRC error on data, Alert Frames from the AMBs, Alert Asserted in Status Frames from the AMBs, or Status Frame Parity Error. When one of these conditions is detected, the DRAM controller will attempt to recover from the condition. If it successfully recovers, an FBR will be logged in the MCU ESR, the error condition will be logged in the MCU Error Syndrome Register, and a DSC will be reported to the L2.

25.11.3 DRAM Uncorrectable Error for Access (DAU/DBU/FBU)

The DRAM detects three types of uncorrectable errors and reports them to the L2 cache as DAU errors:

- Uncorrectable data ECC errors, indicated by setting the dau,veu bits and the synd field in the DRAM ESR.
- Out-of-bounds errors to nonexistent DRAM addresses (for details on out-of-bounds addresses, see *Access to Nonexistent Memory* on page 375). An out-of-bounds error is signaled as a cache line with all data marked with an uncorrectable error. For each 32-bit chunk, the data is loaded into the L2 cache with signaled ECC. These errors are indicated in the DRAM ESR with the dbu bit set. Note that writeback and block stores to out-of-bounds addresses are dropped with no error indication provided. Refer to TABLE 26-3 on page 376 for the complete list of OOB address ranges for different memory configurations.
- Unrecoverable FBD link error associated with DRAM accesses. These errors are indicated by the fbu bit set in the DRAM ESR. The error information is captured in the DRAM FDB Error Syndrome register.

25.11.3.1 Load Miss/Instruction Fetch Miss

The handling of the error in the L2 cache and the virtual processor depend on whether the error is reported on critical load data and whether the load occurs before the linefill. The linefill data is loaded into the L2 cache with signaled ECC.

Critical load data delivered before L2 cache linefill. When the load/ifetch miss request replay occurs before the line fill in the L2 cache and detects the uncorrectable DRAM error on its critical bytes(the 16-byte primary line for loads/ prefetches, 32-byte primary line for instruction fetches) , the error information (dau,veu, rw, vcid, and moda) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

For each 32-bit chunk with an error, the data is loaded into the L2 cache with corrupted ECC. The L1 cache data is loaded with bad parity.

If the L2 Cache Error Enable nceen bit is set, an L2U error is signaled to the requesting virtual processor. The error information is delivered in the err field of a L2_Load_Return packet with the data. The requesting virtual processor indicates a precise error condition. These errors are handled as follows:

- If the access was an instruction fetch and the SPARC CERER.icl2u bit is set and SETER.pscce is set, hardware records icl2u in the ISFSR and takes a precise *instruction_access_error* trap. The VA of the instruction access is in TPC[TL]. If SETER.pscce is clear, hardware does not take a trap nor does it record any error in the ISFSR. Instead, it continues executing using the (corrupt) data from memory.
- If the access was a data fetch and the CERER.dcl2u bit is set and SETER.pscce is set, hardware records dcl2u in DSFSR, and takes a precise *data_access_error* trap. The VA of the data access is *not* logged in DSFSR. Software must examine the L2 ESRs to determine the failing physical address. If SETER.pscce is clear, hardware does not take a trap nor does it record any error in DSFSR or DSFAR. Instead, it continues executing using the corrupt data.

Critical load data delivered after L2 cache line fill. When the load /ifetch request replay occurs after the line fill and in the L2 cache and uncorrectable DRAM error is presented to the L2 cache during the line fill , the error information (dau,veu and vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable nceen bit is set, an L2U error is signaled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *sw_recoverable_error* disrupting error condition. These errors are handled as follows:

- If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes L2U in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.

- If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.11.3.2 Atomic Miss/TTE Miss

Since the atomic access or TTE miss service always occurs after the L2 cache linefill, the L2 cache presents the error after the linefill. The L2 cache also writes the data with signaled ECC. The error information (dau,veu and vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable nceen bit is set, an L2U error is signaled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *sw_recoverable_error* disrupting error condition. These errors are handled as follows:

- If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes l2u in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.
- If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

When the atomic accesses the L2 cache after the linefill , it will detect a NotData condition on the load and store accesses. See *Atomic Hit (DCL2ND)* on page 280.

For an atomic, if the error was in the word to be updated, the atomic operation will not complete its update, leaving the original data and the bad ECC unchanged.

When the TTE miss accesses the L2 cache after the line fill , it will detect a NotData condition . See *TTE Request for ITLB (ITL2ND)* on page 278 and *TTE Request for DTLB (DTL2ND)* on page 279

25.11.3.3 Prefetch Miss

When an uncorrectable error is presented to the L2 cache, the L2 cache presents the error after the linefill. The L2 cache also writes the data with signaled ECC. The error information (dau,veu, vcid = L2_CSR_REG.errorsteer) is captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

If the L2 Cache Error Enable nceen bit is set, an L2U error is signalled to the virtual processor specified in L2_CSR_REG.errorsteer. The error information is delivered in the err field of a Error_Indication_(L2) packet. That virtual processor indicates a *sw_recoverable_error* disrupting error condition. These errors are handled as follows:

- If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes l2u in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.
- If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Prefetch does not load data into the requesting core's data cache.

25.11.3.4 Store Miss

When an uncorrectable error is presented to the L2 cache, the error information (dau,veu, rw, vcid) is captured *i*n the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register. For each 32-bit chunk with an error, the data is loaded into the L2 cache with signaled ECC. For a partial store, the partial store will not complete its update, leaving the original data and the bad ECC unchanged. In addition, if the L2 Cache Error Enable nceen bit is set, a disrupting L2U error is generated to the virtual processor specified in L2_CSR_REG.errorsteer.

If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes L2U in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.

If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

25.11.3.5 DMA Read (DRU/DAU)

When the uncorrectable error is presented to the L2 cache, the error information (dru,veu, vcid = L2_CSR_REG.errorsteer) are captured *i*n the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register. Hardware returns the data back to the DMA requestor, with a UE error indicator for each 128-bit chunk with an error. In addition, if the L2 Cache Error Enable nceen bit is set, an L2U error is reported to the virtual processor specified in L2_CSR_REG.errorsteer.

If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes l2u in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.

If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

L2 Error status will report DRU (DMA uncorrectable) for this error, while the MCU DRAM will report DAU.

25.11.3.6 DMA Write Partial (DRU/DAU)

When the uncorrectable error is presented to the L2 cache, the error information (dru,veu and vcid = L2_CSR_REG.errorsteer) are captured in the L2 Cache Error Status register and PA{39:6} in the L2 Cache Error Address register.

These errors occur for DMA 8-byte writes where the line is read from DRAM into the L2 cache where the DMA write data is merged. For each 32-bit chunk with an error, the data is loaded into the L2 cache with signaled ECC. In addition, if the L2 Cache Error Enable nceen bit is set, an L2U error is reported to the virtual processor specified in L2_CSR_REG.errorsteer.

If the SPARC CERER.l2u_socu bit is set, and the DESR.f bit is clear, hardware encodes l2u in the DESR. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.

If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

L2 Error status will report DRU (DMA uncorrectable) for this error, while the MCU DRAM will report DAU.

25.11.3.7 FBD Channel Unrecoverable CRC Error (FBU)

When the MCU detects a CRC error on a read packet, it will retry the read transaction. If the error condition persists, the MCU will force a Fast Reset of the FBD channel and retry the read transaction. If the error persists after the Fast Reset, the error is logged as an FBU in the MCU ESR, a CRC error is logged in the MCU Error Syndrome Register, and the error is reported to the L2 as a DAU along with the corrupted data from the channel.

25.11.4 DRAM Uncorrectable ECC Error for Scrub (DSU/FBU)

When an uncorrectable ECC error is found during a scrub, the information (dsu, synd) is captured in the DRAM Error Status register, with PA{39:4} captured in the DRAM Error Address register.

When the uncorrectable ECC error for scrub is presented to the L2 cache, the error information (dsu) is captured in the L2 Cache Error Status register. In addition, if the L2 Cache Error Enable nceen bit is set, an L2U error is reported to the virtual processor specified in L2_CSR_REG.errorsteer.

If the SPARC CERER.l2u_socu bit is set and the DESR.f bit is clear, hardware encodes l2u in the desr. The contents of the DESR.erroraddr field are undefined. In addition, if the SETER.de bit is set, and PSTATE.ie is set or HPSTATE.hpriv is clear, hardware generates a disrupting *sw_recoverable_error* trap.

If SETER.de is not set or PSTATE.ie is not set or HPSTATE.hpriv is set, hardware keeps the trap request pending until software either a) resets DESR.f, clearing the trap request, or b) sets SETER.de and either sets PSTATE.ie or clears HPSTATE.hpriv, causing hardware to take the trap.

Note that the dsu bit is set in the L2 Cache ESR regardless of the status of any other bits in the register. The dsc bit is logged to notify software to check the DRAM error register.

25.11.4.1 FBD Channel Unrecoverable Status Frame Parity and Alert Frame Errors

When the MCU detects a Status Frame Parity r an Alert Frame Error, it first issues a Soft Channel Reset on the FBD channel to try to clear the condition. If the error condition persists, it forces a Fast Reset of the FBD channel. If the error conditon persists after the Fast Reset, an FBU is logged in the MCU ESR, the error condition is logged in the MCU Error Syndrome Register, and the error is reported to the L2 as a DSU.

25.11.5 DRAM Software Error Scrubbing Support

Some errors will leave the DRAM with a correctable error, which then needs to be scrubbed to prevent repetitive traps for effectively the same soft error. DRAM scrubbing can be done through the L2 cache. To scrub, load the line into the L2, dirty it without modifying it (via a CAS that matches on the compare and has the same store data as the data returned on the load), then use a PrefetchICE to flush the line back to DRAM.

25.12 DRAM Error Registers

This section describes the error control and log registers for the DRAM. Each DRAM channel has its own set of error registers.

25.12.1 DRAM Error Status Register

This register contains status on DRAM errors. The status bits in this register are cleared by writing a 1 to the bit position.

Note | DRAM_ERROR_STATUS is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

TABLE 25-30 shows the format of the DRAM Error Status register.

TABLE 25-30 DRAM Error Status Register – DRAM_ERROR_STATUS_REG ($84\ 0000\ 0280_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63	meu	0	R/W1C	Multiple uncorrected errors, one or more uncorrected errors were not logged.
62	mec	0	R/W1C	Multiple corrected errors, one or more corrected errors were not logged.
61	dac	0	R/W1C	Set to 1 if the error was a DRAM access correctable error.
60	dau	0	R/W1C	Set to 1 if the error was a DRAM access uncorrectable error.
59	dsc	0	R/W1C	Set to 1 if the error was a DRAM scrub correctable error.
58	dsu	0	R/W1C	Set to 1 if the error was a DRAM scrub uncorrectable error.
57	dbu	0	R/W1C	Set to 1 if the error was an access to a nonexistent DRAM address (address out of bounds).
56	meb	0	R/W1C	Set to 1 if there were multiple out-of-bound errors
55	fbu	0	RW1C	Set to 1 if the error was a FBDIMM channel unrecoverable error
54	fbr	0	R/W1C	Set to 1 if the error was a FBDIMM channel recoverable error
53:16	—	0	RO	*Reserved*
15:0	synd	X	RW	ECC syndrome.

If both correctable and uncorrectable errors occur in the same cycle, the mec bit is set and only the appropriate bit for the uncorrectable error is set. With the exception of DBU, the syndrome is captured for the highest-priority error in that cycle (DSU,

DAU, and FBU are priority 1, DSC, DAC, and FBR are priority 2). The syndrome remains unchanged if the error is a DBU. The address is loaded if the highest-priority error in the cycle is either a DSU or DSC.

If there are multiple errors, in different 16-byte chunks, in a single access, they are treated as multiple errors, since there is only logging state to describe a single 16-byte chunk error.

Note | MCU FBDIMM Unrecoverable error (FBU) can be injected through NCU; but error is not reported back to NCU, unlike FBR.

If errors occur in a cycle where an error status bit is already set, TABLE 25-31 applies.

TABLE 25-31 Errors When Error Status Bit Is Already Set

Existing Error	Error	Priority	Bit Set
DSU/DAU/FBU	DSU, DAU, or FBU	1	meu
DSU/DAU/FBU	DBU	1	dbu
DSU/DAU/FBU	DSC, DAC or FBR	2	mec
DBU	DSU	1	dsu
DBU	DAU	1	dau
DBU	FBU	1	fbu
DBU	DBU	1	meb
DBU	DSC	2	dsc
DBU	DAC	2	dac
DBU	FBR	2	fbr
DSC/DAC/FBR	DSU	1	dsu
DSC/DAC/FBR	DAU	1	dau
DSC/DAC/FBR	DBU	1	dbu
DSC/DAC/FBR	FBU	1	fbu
DSC/DAC/FBR	DSC, DAC, or FBR	2	mec

For the cases above where the "Bit Set" column contains a value besides meb, mec, and meu, the syndrome (for DSU/DAU/FBU/DSC/DAC/FBR) and address (for DSU/DSC) for the highest-priority error will overwrite the existing syndrome and address.

Once set, error status bits are only cleared by software. Hardware will never clear a set status bit. If a software write of the error register happens on the same cycle as an error, the setting of bits by the error will be based on the register state before the write. The setting of fields by the error will take precedence over the same field being updated by the write; however, fields that are not changed by the error will be updated by the write (for example, if the register has the dac bit set and software does a write to clear that bit on the same cycle as a DRAM scrub uncorrectable error, the error register would end up with the dac bit cleared and the mec bit set, and the rw and synd fields would contain the values for the error).

25.12.2 DRAM Error Address Register

The DRAM Error Address register contains the physical address for the DRAM scrub error. DRAM load access address for errors are expected to be logged by L2 controller.

Note | DRAM_ERROR_ADDRESS is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

TABLE 25-32 shows the format of the DRAM Error Address register.

TABLE 25-32 DRAM Error Address Register – DRAM_ERROR_ADDRESS_REG ($84\ 0000\ 0288_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:40	—	0	RO	*Reserved*
39:4	address	X	RW	Error address.
3:0	—	0	RO	*Reserved*

This register is writable by software for register diagnostic reasons and isn't expected to be written during normal operation. However, in the event it is written on the same cycle that an error is reported, the update from the error will take precedence over the write.

25.12.3 DRAM Error Injection Register

Each DRAM channel has an Error Injection register for use in injecting DRAM errors to test error functionality or error handling code. The DRAM Error Injection register only provides for the injection of bad ECC on data written to memory. To inject an ECC error on data read from memory, bad ECC must be set up via a memory write, and then the memory locations with the bad ECC accessed with a read. Errors can be injected either single-shot or continuously. Once the enb_hp bit is set, either the first subsequent operation (for sshot = 1), or all subsequent operations (for sshot = 0) that

cause a DRAM write will **xor** eccmask with the normally generated ECC when writing memory. When in single-shot mode, after the first injected error is generated, the sshot and enb_hp are automatically reset by the hardware to 0.

TABLE 25-33 shows the format of the DRAM Error Injection register.

TABLE 25-33 DRAM Error Injection Register – DRAM_ERROR_INJECT_REG ($84\ 0000\ 0290_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:32	—	0	RO	*Reserved*
31	enb_hp	0	RW	Enables error injection.
30	sshot	0	RW	Controls type of error injection. 1 = single shot; 0 = continuous.
29:16	—	0	RO	*Reserved*
15:0	eccmask	0	RW	ECC mask for error injection. The mask is **xor**ed with the computed ECC.

Note | DRAM_ERROR_INJECT_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.12.4 DRAM Error Counter Register

Each DRAM channel has an Error Counter register for use in counting DRAM correctable ECC errors and generating a trap when the counter decrements to 0. Each 16-byte chunk with an error will cause the count field to decrement by one.

When count transitions from 1 to 0, the corresponding mcu{0-3} ECC bit is set in the SOC_ERROR_STATUS_REG described in *SOC Error Status Register* on page 342, which if enabled will generate a *sw_recoverable_error* trap to the lowest enabled strand in the SOC_ERROR_STEER_REG.

TABLE 25-34 shows the format of the DRAM Error Counter register.

TABLE 25-34 DRAM Error Counter Register – DRAM_ERROR_COUNTER_REG ($84\ 0000\ 0298_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:16	—	X	RO	*Reserved*
15:0	COUNT	0	RW	Counter that decrements with each ECC correctable error unless already 0.

Note | DRAM_ERROR_COUNTER_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.12.5 DRAM Error Location Register

Each DRAM channel has an Error Location register for software to sample to locate a bad memory part. The register is set on a correctable DRAM error if a correctable error is not already logged in the MCU ESR. The bit set corresponds to the nibble that was corrected. Bits 35:32 correspond to the ECC nibbles and 31:0 to data nibbles. When in dual-channel mode, 35:18 refer to nibbles in channel 0 and 17:0 refer to nibbles in channel 1.

TABLE 25-35 shows the format of the DRAM Error Location register.

TABLE 25-35 DRAM Error Location Register – DRAM_ERROR_LOCATION_REG (84 0000 $02A0_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:36	—	0	RO	*Reserved*
35:0	location	X	RW	DRAM ECC error location.

Note | DRAM_ERROR_LOCATION_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.12.6 DRAM Error Retry Register

Each DRAM channel has an Error Retry register for software to sample to locate a bad memory part. The register is set on each correctable DRAM error.

TABLE 25-36 shows the format of the DRAM Error Retry register.

TABLE 25-36 DRAM Error Retry Register – DRAM_ERROR_RETRY_REG (84 0000 $02A8_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:37	—	0	RW	*Reserved*
36	valid	0	RW	Error Retry register is valid.
35:20	synd2	0	RO	Syndrome from second retry read.

TABLE 25-36 DRAM Error Retry Register – DRAM_ERROR_RETRY_REG ($84\ 0000\ 02A8_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
19:18	type2	0	RW	Result of second retry read.
17:2	synd1	0	RO	Syndrome from first retry read
1:0	type1	0	RW	Result of first retry read. 00 – No read. 01 – No error. 10 – Correctable error. 11 – Uncorrectable error.

Note | DRAM_ERROR_RETRY_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.12.7 DRAM FBD Error Syndrome Register

Each DRAM channel has an FBD Error Syndrome for FBD link errors. When an FBD link error is detected, the syndrome is captured in this register, and for a recoverable (unrecoverable) link error, the corresponding mcu{0-3} FBR (mcu{0-3} FBU) bit is set in the SOC_ERROR_STATUS_REG described in *SOC Error Status Register* on page 342, which if enabled will generate a *hw_corrected_error* trap to the lowest enabled strand in the ASI_CORE_ENABLE_STATUS register. Once the valid bit is set, no further FBD link errors are logged, so software will need to clear the valid bit to enable further FBD link error detection.

TABLE 25-37 shows the format of the DRAM FBD Error Syndrome register.

TABLE 25-37 DRAM FBD Error Syndrome Register – DRAM_FBD_ERROR_SYND_REG ($84\ 0000\ 0C00_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63	valid	0	RW	Valid.
62:30	—	0	RO	*Reserved*
29:18	alert1	0	RW	AMB alert bits set in Channel 1.
17:6	alert0	0	RW	AMB alert bits set in Channel 0.
5	softreset	0	RW	MCU issued a soft channel reset command to the channel.
4	fastreset	0	RW	MCU performed a fast reset of a channel due to a soft channel reset for the error not being effective.
3	sfpe	0	RW	Status frame parity error.

TABLE 25-37 DRAM FBD Error Syndrome Register – DRAM_FBD_ERROR_SYND_REG (84 0000 $0C00_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
2	aa	0	RW	Alert asserted.
1	afe	0	RW	Alert frame error.
0	c	0	RW	CRC error.

Note | DRAM_FBD_ERROR_SYND_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.12.8 DRAM FBD Injected Error Source Register

When the NCU signals the MCU to inject an error, this register determines into which error detection logic the error will be injected.

TABLE 25-38 shows the format of the DRAM FBD Injected Error Source register.

TABLE 25-38 DRAM FBD Error Syndrome Register – DRAM_FBD_INJ_ERROR_SRC_REG (84 0000 $0c08_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
62:2	—	0	RO	*Reserved*
1:0	errorsource	0	RW	Source of the error. 0 – CRC error. 1 – Alert frame error. 2 – Alert asserted. 3 – Status frame parity error.

Note | DRAM_FBD_INJ_ERROR_SRC_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

Note | Continuous injection of Alert Frame errors may cause a system to hang. The memory controller will be continually processing the Alert Frame errors and will not have time to service DRAM read requests.

Note | When injecting Alert Asserted errors, the memory controller will only report one AA error to the L2 until the MCU Error Syndrome register is cleared.

25.12.9 DRAM FBR Count Register

This register controls the sending of FBD recoverable error interrupts to the NCU. Each DRAM channel has an FBR Count register for use in counting FBD recoverable channel errors and generating a trap when the counter decrements to 0. Each FBR will cause the count field to decrement by one.

When count transitions from 1 to 0, the corresponding mcu{0-3} FBR bit is set in the SOC_ERROR_STATUS_REG described in *SOC Error Status Register* on page 342, which if enabled will generate a *sw_recoverable_error* trap to the lowest enabled strand in the SOC_ERROR_STEER_REG.

TABLE 25-39 shows the format of the DRAM FBR Count register.

TABLE 25-39 DRAM FBR Count Register – DRAM_FBR_COUNT_REG (84 0000 $0C10_{16}$) (Count 4 Step 4096)

Bit	Field	POR Value	R/W	Description
63:17	—	0	RO	*Reserved*
16	countone	0	RW	If set, the MCU will generate an interrupt to the NCU on every FBR error.
15:0	count	0	RW	Count of FBR errors, decremented on every FBR error if countone is not set. When this value decrements from 1 to 0, an FBR interrupt will be sent to the NCU.

Note | DRAM_FBR_COUNT_REG is not cleared on warm reset to allow software to examine the error information that may have required the warm reset.

25.13 Block Loads and Stores

OpenSPARC T2 supports 64-byte block load and store access to ASI_BLK_P, ASI_BLK_S, ASI_BLK_COMMIT_P, ASI_BLK_COMMIT_S, ASI_BLK_PL, ASI_BLK_SL, ASI_BLK_AIUP, ASI_BLK_AIUS, ASI_BLK_AIUPL, ASI_BLK_AIUSL.

Loads for these operations consist of four 16-byte "helper" loads, while stores are composed of eight 8-byte "helper" stores. Store buffer errors encountered on any of the helper stores will be logged the same as if it was a non-helper store and will not

affect the issuing of subsequent helper stores. For block stores, if there is an error on any of the helper stores, the entire block store will be terminated and no memory updates performed.

For the helper loads, errors (including L2C) will be accumulated on all four helpers and reported (and trapped if enabled) after the final helper completes. The error reported will be the highest-priority error (ND → UE → CE) if one or more errors are detected. If multiple errors of the highest priority are detected, the error reported will be the one associated with the earliest helper.

The block load is considered a single operation, and even if multiple uncorrectable or correctable errors (or a combination of the two) are encountered, they are treated as a single error with respect to updating the D-SFSR. The floating-point register file will be updated with load results up to the point of the earliest helper that encountered an uncorrectable error. Loading of results into the floating-point register file will be suppressed for the helper that encountered the uncorrectable error and any subsequent helpers.

Programming Note When a block load encounters an error, the destination registers of the helper operations issued before the helper encountering the error will be updated. This implies that the exception taken for the error will not be strictly precise. However, the TPC and TNPC will still be updated as if the exception were fully precise and TPC will point to the block load instruction that encountered the error.

25.14 CMP Error Summary

TABLE 25-40 summarizes CMP error handling. Column heads and error-type abbreviations are described in TABLE 25-41 on page 320.

TABLE 25-40 SPARC, L2 and DRAM Error Handling Summary *(1 of 5)*

Error Type	Error	L2 ESR ISFSR/DSFSR/ DESR/DFESR	(NotData ESR = ND)	DRAM Status	PA	Syn	Trap	Trap Type	Section
ITLB tag parity	CE	ITTP				is	p	IM	25.7.1.2
IT tag multiple hit	CE	ITTM				is	p	IM	25.7.1.1
ITLB Data Parity	CE	ITDP				is	p	IM	25.7.1.3
ITLB MRA uncorrectable	UE	ITMU				is	p	IM	25.7.1.4
ITLB L2 correctable	CE	ITL2C	LDAC		L	l	di	E	25.9.1.1
ITLB L2 uncorrectable	UE	ITL2U	LDAU		L	l	p	IM	25.9.6.1
ITLB L2 NotData	ND	ITL2ND	ND.NDSP		N		p	IM	25.9.13.1
Icache valid bit	CE	ICVP				de	di	C	25.7.3.1
Icache tag parity	CE	ICTP				de	di	C	25.7.3.2
Icache tag multiple hit	CE	ICTM				de	di	C	25.7.3.3
Icache data parity	CE	ICDP				de	di	C	25.7.3.4
Icache L2 correctable	CE	ICL2C	LDAC		L	l	di	E	25.9.1.3
Icache L2 uncorrectable	UE	ICL2U	LDAU		L	l	p	I	25.9.6.3
Icache L2 NotData	ND	ICL2ND	ND.NDSP		N		p	I	25.9.14.2
IRF correctable ECC error	CE	IRFC				ds	p	P	25.7.5
IRF uncorrectable ECC error	UE	IRFU				ds	p	P	25.7.5
FRF correctable ECC error	CE	FRFC				ds	p	P	25.7.6
FRF uncorrectable ECC error	UE	FRFU				ds	p	P	25.7.6
DTLB tag parity	CE	DTTP				ds	p	DM	25.7.2.2
DT tag multiple hit	CE	DTTM				ds	p	DM	25.7.2.1
DTLB data parity	CE	DTDP				ds	p	DM	25.7.2.3
DTLB MRA uncorrectable	UE	DTMU				ds	p	DM	25.7.2.4
DTLB L2 correctable	CE	DTL2C	LDAC		L	l	di	E	25.9.1.2
DTLB L2 uncorrectable	UE	DTL2U	LDAU		L	l	p		25.9.6.2
DTLB L2 NotData	ND	DTL2ND	ND.NDSP		N		p	DM	25.9.13.2
Dcache valid bit	CE	DCVP				de	di	C	25.7.4.1
Dcache tag parity	CE	DCTP				de	di	C	25.7.4.2

TABLE 25-40 SPARC, L2 and DRAM Error Handling Summary *(2 of 5)*

Error Type	Error	L2 ESR ISFSR/DSFSR/ DESR/DFESR	L2 ESR (NotData ESR = ND)	DRAM Status	PA	Syn	Trap	Trap Type	Section
Dcache tag multiple hit	CE	DCTM				de	di	C	25.7.4.3
Dcache Data Parity	CE	DCDP				de	di	C	25.7.4.4
Dcache L2 Correctable	CE	DCL2C	LDAC		L	l	di	E	25.9.1.4
Dcache L2 Uncorrectable	UE	DCL2U	LDAU		L	l	p	D	25.9.6.4 and 25.9.6.5
Dcache L2 NotData	ND	DCL2ND	ND.NDSP		N		p	D	25.9.13.4 and 25.9.13.5
Store Buffer Data Load hit Correctable ECC	CE	SBDLC				ds	p	P	25.7.7.1
Store Buffer Data Load hit Uncorrectable ECC	UE	SBDLU				ds	p	P	25.7.7.2
Store Buffer Data PCX read or ASI store Correctable	CE	SBDPC				de	di	C	25.7.7.4
Store Buffer Data PCX read or ASI store Uncorrectable	UE	SBDPU				de	di	E	25.7.7.5
Store Buffer Address PCX read or ASI ring store read Uncorrectable	DE	SBDIOU				df	df	S	25.7.7.6
Store Buffer Address PCX read or ASI ring store read Address Bit Parity	DE	SBAPP				df	df	S	25.7.7.7
TSA Correctable	CE	TSAC				ds	p	P	25.7.10
TSA Uncorrectable	UE	TSAU				ds	p	P	25.7.10
MRA Uncorrectable	UE	MRAU				ds	p	P	25.7.11
SCA Correctable	CE	SCAC				ds	p	P	25.7.8
SCA Uncorrectable	UE	SCAU				ds	p	P	25.7.8
Tick_Compare Correctable Precise	CE	TCCP				ds	p	P	25.7.9
Tick_Compare Correctable Disrupting	CE	TCCD				de	di	E	25.7.9
Tick_Compare Uncorrectable Precise	UE	TCUP				ds	p	P	25.7.9
Tick_Compare Uncorrectable Disrupting	UE	TCUD				de	di	E	25.7.9
IO error from Load from noncacheable address	UE	SOCU			S	s	p	D	

TABLE 25-40 SPARC, L2 and DRAM Error Handling Summary *(3 of 5)*

Error Type	Error	ISFSR/DSFSR/ DESR/DFESR	L2 ESR (NotData ND)	DRAM Status	PA	Syn	Trap	Trap Type	Section
IO error from Ifetch from noncacheable address	UE	ICL2U			S	s	p	I	
L2$ data ecc: LD_h, If_h, ATOM_h, PF_h	CE		LDAC		L	l	di	E	
L2$ data ecc: LD_h, ATOM_h	UE	LDAU	LDAU		LS	l	p	D	
L2$ data ecc: PF_h	UE		LDAU		L	l	di	E	
L2$ data ecc: If_h	UE	LDAU	LDAU		LS	l	p	I	
L2$ data NotData: LD_h, ATOM_h	ND	DCL2ND	ND.NDSP		N		p	D	
L2$ data NotData: PF_h	ND		ND.NDSP		N		di	E	
L2$ data NotData: If_h	ND	ICL2ND	ND.NDSP		N		p	I	
L2$ data ecc: PST_h	CE		LDAC		L	l	di	C	
L2$ data ecc: PST_h	UE		LDAU		L	l	di	E	
L2$ data ecc: PST_h	ND	L2ND	ND.NDSP		N	de	di	E	
L2$ data ecc: wb	CE		LDWC		L	l	di (ES)	C	
L2$ data ecc: wb	UE		LDWU		L	l	di (ES)	E	
L2$ data ecc:dma_read	CE		LDRC		L	l	di (ES)	E	
L2$ data ecc:dma_read	UE		LDRU		L	l	di (ES)	E	
L2$ data ecc:dma_read	ND	L2ND	ND.NDDM		N	de	di (ES)	E	
L2$ data ecc:dma_write_partial	CE		LDRC		L	l	di (ES)	C	
L2$ data ecc:dma_write_partial	UE		LDRU		L	l	di (ES)	E	
L2$ data ecc:dma_write_partial	ND	L2ND	ND.NDDM		N	de	di (ES)	E	
L2$ data ecc: scrub	CE		LDSC		L	l	di (ES)	C	
L2$ data ecc: scrub	UE		LDSU		L	l	di (ES)	E	
L2$ tag ecc: all refs	CE		LTC		L	—	di (ES)	C	
L2 $ dir parity: scrub	FE		LRF		L	l	F	R	
L2 $ vuad ecc: all refs	FE		LVF		L	l	F	R	
L2 $ vuad ecc: all refs	CE		LVC		L	l	di (ES)	C	
Dram ECC error: LD_m_cc, If_m_c32B, PF_m_cc	CE		DAC	DAC	L	D	di	E	
FBDIMM recoverable error: LD_m_cc, If_m_c32B, PF_m_cc	CE		DAC	FBR	L	F	di	E	

TABLE 25-40 SPARC, L2 and DRAM Error Handling Summary *(4 of 5)*

Error Type	Error	ISFSR/DSFSR/ DESR/DFESR	L2 ESR (NotData ESR = ND)	DRAM Status	PA	Syn	Trap	Trap Type	Section
Dram ECC error: LD_m_cc, ATOM_m_cc	UE	LDAU	DAU	DAU	LS	D	p	D	
Dram ECC error: PF_m_cc	UE		DAU	DAU	L	D	di	E	
Dram ECC error: If_m_c32B	UE	LDAU	DAU	DAU	LS	D	p	I	
FBDIMM unrecoverable error: LD_m_cc, ATOM_m_cc	UE	LDAU	DAU	FBU	LS	F	p	D	
FBDIMM unrecoverable error: PF_m_cc	UE		DAU	FBU	L	F	di	E	
FBDIMM unrecoverable error: If_m_c32B	UE	LDAU	DAU	FBU	LS	F	p	I	
Dram address out of bounds: LD_m, ATOM_m	UE	LDAU	DAU	DBU	LS	D	p	D	
Dram address out of bounds: PF_m	UE		DAU	DBU	L	D	di	E	
Dram address out of bounds: If_m	UE	LDAU	DAU	DBU	LS	D	p	I	
Dram ECC error: ST_m, PST_m, LD_m_ncc, If_m_nc32B, ATOM_m, PF_m_ncc	CE		DAC	DAC	L	D	di (ES)	C	
Dram ECC error: dma_read_req, dma_write_partial	CE		DRC	DAC	L	D	di (ES)	C	
FBDIMM recoverable error: Status Frame Parity Error, Alert Frames, AMB Alert Asserted, CRC	CE		DSC	FBR	L	F	di (ES)	C	
Dram ECC error: ST_m,PST_m, LD_m_ncc, If_m_nc32B, ATOM_m, PF_m_ncc	UE		DAU	DAU	L	D	di (ES)	E	
Dram ECC error: dma_read_req, dma_write_partial	UE		DRU	DAU	L	D	di (ES)	E	
FBDIMM unrecoverable error: CRC	UE		DAU	FBU	L	F	di (ES)	E	
FBDIMM unrecoverable error: Status Frame Parity Error, Alert Frame	UE		DSU	FBU	L	F	di (ES)	E	

TABLE 25-40 SPARC, L2 and DRAM Error Handling Summary *(5 of 5)*

Error Type	Error	ISFSR/DSFSR/ DESR/DFESR	L2 ESR (NotData ESR = ND)	DRAM Status	PA	Syn	Trap	Trap Type	Section
Dram address out of bounds: ST_m, PST_m	UE		DAU	DBU	L	D	di (ES)	E	
Dram address out of bounds: dma_read_req, dma_write_partial	UE		DRU (DMA read) DAU (DMA write)	DBU	L	D	di (ES)	E	
Dram Scrub error	CE		DSC	DSC	D	D	di (ES)	C	
Dram Scrub error	UE		DSU	DSU	D	D	di (ES)	E	

TABLE 25-41 describes the column heads and error-type abbreviations (in alphabetical order) of TABLE 25-40.

TABLE 25-41 Description of Column Heads and Error-Type Abbreviations in TABLE 25-40

Column or Error-Type Abbr	Meaning
Error	FE – fatal error; UE – uncorrected error; CE – corrected error, ND - NotData error, DF – deferred error
PA logging	S – D-SFAR; N – L2 NotData error; L – L2 error address; D – DRAM error address
SYN logging	is – I-SFSR; ds – D-SFSR; de – DESR; df – DFESR; l – L2 error status; D – DRAM error status; F – DRAM FBD error syndrome
Trap	p – precise to requestor; di – disrupting to requestor; di (ES) – disrupting to the virtual processor specified in L2_CSR_REG.errorsteer; f - fatal warm reset to all virtual processors
Trap Type	I – *instruction_access_error*; IM – *instruction_access_MMU_error*; D – *data_access_error,* DM – *data_access_MMU_error*; P - *internal_processor_error,*; E - *sw_recoverable_error,* C – *hw_corrected_error*; S – *store_error*; R – *warm_reset*
ATOM_h	Atomic operation hit
ATOM_m_cc	Atomic operation miss critical 16-byte chunk
ATOM_m_ncc	Atomic operation miss noncritical chunk
dma_read	DMA read any size
dma_write_partial	subline DMA write
If_h	I-fetch hit
If_m_c32B	Ifetch miss critical 32-byte chunk
If_m_nc32B	Ifetch miss noncricital 32-byte chunk
LD_h	Load hit
LD_m_cc	Load miss critical 16-byte chunk
LD_m_ncc	Load miss noncritical chunk
PF_h	Prefetch hit
PF_m_cc	Prefetch miss critical 16-byte chunk
PF_m_ncc	Prefetch miss noncritical chunk

TABLE 25-41 Description of Column Heads and Error-Type Abbreviations in TABLE 25-40 *(Continued)*

Column or Error-Type Abbr	Meaning
PST_h	Partial Store hit
PST_m	Partial Store miss
wb	writeback to memory

25.15 Boot ROM Interface (SSI)

TABLE 25-42 describes the SSI's handling of errors. The error indication on read returns is delivered regardless of the SSI_TIMEOUT.erren bit, where it is up to the virtual processor to ignore the error or receive it based on whether or not CERER.icl2u and CERER.dcl2u bit are set. Logging the error and sending an error interrupt are controlled by the erren bit. Note that returning zeros on an I-fetch timeout will tend to cause an *illegal_instruction* trap.

TABLE 25-42 SSI Error Handling

Error	TType	Severity	Core Error Info	Trap Taken	Error Returned on Transaction	NCU Error Info (if SSI_TIMEOUT. erren =1)
SSI parity error	Read	Uncorrectable	I-SFSR = icl2u or D-SFSR = dcl2u	*instruction_ access_error* or *data_access_ error*	Yes (with data)	SSI_LOG. parity
SSI parity error	Write	Uncorrectable	Not Recorded	None	No	SSI_LOG. parity
SSI timeout	Read	Uncorrectable	I-SFSR = icl2u or D-SFSR = dcl2u	*instruction_ access_error* or *data_access_ error*	Yes (with data = 0)	SSI_LOG. tout
SSI timeout	Write	Uncorrectable	Not Recorded	None	No	SSI_LOG. tout

Note | I-fetch to any other I/O space other than SSI boot ROM space would result in error packet being returned to the requesting SPC by NCU.

25.15.1 SSI Parity Error

SSI has serial parity on all requests and responses. Odd parity on any response will be treated as a parity error.

On reads, the SSI block will return the data, but marked with an error indication, which will tend to cause an NCU error at the requesting SPARC. For both reads and writes, the SSI block will issue an error interrupt via int_man{1} (if SSI_TIMEOUT.erren is set).

25.15.2 SSI Timeout

SSI only supports a single transaction outstanding at any time, and write transactions receive a positive acknowledgement to inform OpenSPARC T2 of their completion. Whenever OpenSPARC T2 issues a read or write transaction, it starts a timer to the value specified in SSI_TIMEOUT[TIMEVAL], which then decrements by 1 every SSI cycle. If the time underflows before the transaction completes, it is treated as a timeout.

On reads, the SSI block will return zeros, but marked with an error indication, which will tend to cause an NCU error at the requesting SPARC. For both reads and writes, the SSI block will issue an error interrupt via int_man{1} (if SSI_TIMEOUT.*ERREN* is set).

25.15.3 SSI Error Registers

The serial bus interface to the Boot ROM is called SSI, hence the registers dealing with errors on this interface are SSI registers.

TABLE 25-43 and TABLE 25-44 define the format of the SSI Timeout and Log registers, respectively.

TABLE 25-43 SSI Timeout Register– SSI_TIMEOUT (FF 0001 0088_{16})

Bit	Field	Initial Value	R/W	Description
63:25	—	X	RO	*Reserved*
24	erren	0	RW	Enables error logging and error interrupt generation in the SSI.
23:0	timeval	200000_{16}	RW	Number of SSI cycles before an unacknowledged request causes a timeout error.

The default value for timeout is about 40 msec.

TABLE 25-44 217 SSI Log Register – SSI_LOG (FF 0000 0018_{16})

Bit	Field	Initial Value	R/W	Description
63:2	—	X	RO	*Reserved*
1	parity	0	RW1C	Parity error detected on response.
0	tout	0	RW1C	No response before TIMEVAL.

25.16 Error Injection Summary

Most of the large arrays on OpenSPARC T2 have some error protection, and also the capability of getting an error injected. TABLE 25-45 specifies the programmatic interface for injection, plus some notes of the type of injection.

TABLE 25-45 Error Injection Summary

Error	Control	Notes
ITLB data parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
ITLB CAM parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
DTLB data parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
DTLB CAM parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
Icache data parity	ASI_ICACHE_INSTR	Optionally flip parity on ASI write.
Icache tag parity	ASI_ICACHE_TAG	Optionally flip parity on ASI write.
Dcache data parity	ASI_DCACHE_DATA	Flip parity bits under mask on ASI write.
Dcache tag parity	ASI_DCACHE_TAG	Optionally flip parity on ASI write.
Int RegFile ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
FP RegFile ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
Scratchpad array ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
Tick_compare Array ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
TSA ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
MRA parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
Store buffer CAM parity	ASI_ERROR_INJECT_REG	Continuous parity flip on update.
Store buffer data ECC	ASI_ERROR_INJECT_REG	Continuous ECC XOR on update.
L2 data ECC	L2_DIAG_DATA	Write 4B data and computed ECC.
L2 tag ECC	L2_DIAG_TAG	Write tag and computed ECC.
L2 directory parity	L2_ERROR_INJECT_REG	Single/double or continuous parity flip on update.
L2 UA ECC	L2_DIAG_UA	Write UA bits and computed ECC.
L2 VD ECC	L2_DIAG_VD	Write VD bits and computed ECC.
DRAM ECC	DRAM_ERROR_INJECT_REG	Continuous syndrome XOR on write.
SOC errors	SOC_ERROR_INJECT_REG	Continuous parity/ECC corrupt.

25.17 SOC Error Descriptions

An error in the system-on-the-chip (SOC) can occur on the major types of SOC operations. The SOC detects correctable and uncorrectable errors; it does not detect NotData errors. Any error in the SOC can be made fatal under the control of the SOC Fatal Error Enable register, see *SOC Fatal Error Enable Register* on page 351 for details.

The SOC operation types are as follows:

- Error on load to I/O space (PIO load)
- Error on store to I/O space (PIO store)
- Errors on an interrupt request
- Errors on DMA reads and writes
- Error interrupts from the MCU due to an error count registers

These errors are discussed in the following sections.

25.18 PIO Load Errors

Three classes of PIO load errors are detected in the SOC: fatal errors (FE), which cause a warm reset; uncorrectable errors (UE), which generate a *data_access_error* trap to the requesting strand or *sw_recoverable_error* to the strand specified in the SOC Error Steering Register ; or correctable errors (CE), which generate a *hw_corrected_error* trap to the requesting strand .

See TABLE 25-46 for details.

TABLE 25-46 SOC Errors for a PIO Load Request

Error	SOCESR bit	Recommended Error Type	Error Info (Core)	Error Info (NCU)	Trap
NCU uncorrectable on command or thread_id from PCX	ncupcxue{19}	FE	DESR = SOCU if SOC EIE bit (SOC Error Interrupt Enable bit) set	NCUSYN (etag, rqtyp, cpu_id, thread_id, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
SII command parity error or Ctag UE	siidmuue{1}	FE	DESR = SOCU if SOC EIE bit set	NCUSIISYN (etag, ctag, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU Ctag UE	ncuctague{22}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (ctag)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU data parity	ncudataparity{14}	FE	D-SFSR = SOCU, DESR = SOCU if SOC EIE bit set	NCUSYN (ctag)	Warm reset if configured as fatal; otherwise, *data_access_error*. and *sw_recoverable_error* if SOC EIE bit set
NCU Mondo Table Parity	ncumondotable{15}	UE	D-SFSR = SOCU, DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *data_access_error* , and *sw_recoverable_error* if SOC EIE bit set
NCU DMU data parity	ncudmuue{21}	FE	DESR = SOCU, DESR = SOCU IF SOC EIE BIT SET	NCUSYN (etag, reqtype, cpuid, strandid, pa)	Warm reset if configured as fatal; otherwise *data_access_error* , and, *sw_recoverable_error* if SOC EIE bit set
NCU FIFO output error	ncucpxue{20}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise *sw_recoverable_error* if SOC EIE bit set
NCU timeout, unmapped error or uncorrected error	—	UE	D-SFSR = SOCU	Not recorded	Warm reset if configured as fatal; otherwise, *data_access_error*.

TABLE 25-46 SOC Errors for a PIO Load Request

Error	SOCESR bit	Recommended Error Type	Error Info (Core)	Error Info (NCU)	Trap
NCU PCX Data	ncupcxdata{18}	UE	DESR = SOCU if SOC EIE bit set	NcuSyn (Etag, Rqtyp, Cpu_id, Thread_id, and PA)	Warm reset if configured as fatal; otherwise sw_recoverable_error if SOC EIE bit set
SII Ctag CE	siidmuce{3}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise *hw_corrected_error* if SOC EIE bit set.
NCU Ctag CE	ncuctagce{23}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *hw_corrected_error* if SOC EIE bit set.

25.18.1 Uncorrectable PIO Load Errors Recommended as Fatal

The following uncorrectable PIO Load errors are recommended to be set as fatal in the SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. Fatal errors associated with a PIO load cause the PIO load to be dropped and a warm reset to be initiated.NCUPCXDATANCUPCXDATA

25.18.1.1 Uncorrectable NCU FIFO Errors (NcuPcxUE)

The NCU provides DEDSEC protection on the NCU FIFO. If an uncorrectable error is detected on the command or thread ID fields, bit 19, ncupcxue, in the SOC ESR is set. The error information (etag, rqtyp, cpu_id, thread_id, and PA{39:0}) is recorded in the NCUSYN register.

No response is returned, so the requesting strand will become hung. Therefore, this error needs to be fatal.

25.18.1.2 Uncorrectable SII Ctag/Command Parity Errors (SiiDmuCtagUE)

For PIO load return requests (through the DMU) to the SII, the Ctag ECC is checked and also the command parity. If an uncorrectable Ctag ECC error or command parity error is detected, bit 1, siidmuctague, in the SOC ESR is set. The error information (etag, ctag, and PA{39:0}) is recorded in the NCUSIISYN register.

No response is returned, so the requesting strand will become hung. Therefore, this error needs to be fatal.

25.18.1.3 Uncorrectable NCU Ctag Error (NcuCtagUe)

The NCU also checks Ctag ECC from the SII. If an uncorrectable Ctag ECC error is detected, bit 22, ncuctague, in the SOC ESR is set. The error information (Ctag) is recorded in the NCUSYN register. A return packet is never generated to the requesting strand and that strand will become hung. Therefore, this error needs to be fatal.

25.18.1.4 NCU Data Parity Error (NcuDataParity)

The NCU checks data parity for returning PIO load requests from the DMU (via the SII). If a data parity error is detected, bit 14, ncudataparity, in the SOC ESR is set. The error information (Ctag) is recorded in the NCUSYN register. For a PIO load, this error does not need to be fatal, because an error packet is returned to the requesting strand. However, since the ncudataparity bit in the SOCESR is shared with interrupts from the SII, which do need to be fatal, the fatal error will also occur for interrupts when the ncuctague bit is set.

25.18.1.5 NCU FIFO Output Error (NcuCpxUe)

The NCU checks the output FIFO to the CPX. If the NCU detects an error in the output FIFO, bit 20, ncucpxue, in the SOC ESR is set.No other error information is captured.

No response is returned to the requesting strand, which will become hung. Therefore, this error needs to be fatal.

25.18.1.6 NCU Parity Error from NCU DMU PIO Req FIFO (NcuDmuUe)

If NCU detects a parity error from the NCU DMU PIO Req FIFO on a PIO load, bit 21, ncudmuue, in the SOC ESR is set. The error information (etag, rqtyp, cpu_id, thread_id and PA{39:0}) is recorded in the NCUSYN register. NCU returns data on CPX packet marked with UE. NCU squashes the PIO load request to DMU.

However since the parity error can be in cpu_id[2:0] bits , the load return might happen to the wrong CPU. Hence this error needs to be fatal.

25.18.2 Uncorrectable PIO Load Errors

The following uncorrectable PIO load errors are recommended to *not* be set as fatal in the SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For uncorrectable SOC detected errors associated with the PIO load, the NCU Load Return packet is sent to the requesting virtual processor with the err field indicating uncorrectable error.

25.18.2.1 NCU Mondo Table Error (NcuMondoTable)

A PIO load can access the Mondo Table in the NCU. If the NCU detects a data parity error, bit 15, ncumondotable, in the SOC ESR is set. No other error is information captured.

25.18.2.2 NCU PCX FIFO Data Parity Error (NcuPCXData)

A PIO load request to NCU that has parity error in the data (payload) logs ncupcxdata bit 18 in the SOC ESR, eventhough the data is ignored by NCU. The error information (etag, rqtyp, cpu_id, thread_id and PA{39:0}) is recorded in the NCUSYN register. NCU returns data on CPX packet without any error bit set.

25.18.2.3 Other Uncorrectable NCU Errors

Errors may have been detected for the PIO load request. These errors are reported as bits in the packet from the SII. The NCU checks the error bits 29:31 in the SII to NCU packet. If bit 31, timeout; bit 30, Unmapped address error (dmuae); or bit 29, uncorrected error from DMU (dmuue), is set, a UE is reported back to the thread that issue the PIO load. Bits 29:31 are logically **or**ed. No ESR is set in the SOC.

25.18.3 Correctable PIO Load Errors

The following correctable PIO load errors are recommended to *not* be set as fatal in the SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For correctable SOC detected errors associated with the PIO load, the NCU Load Return packet is sent to the requesting virtual processor with the err field indicating correctable error.

25.18.3.1 Correctable SII Ctag Error (SiiDmuCtagCE)

If a correctable Ctag error is detected, the Ctag is corrected and bit 3, siidmuctagce, in the SOC ESR is set. No other error information is provided. Data parity is not checked at this point. It will be checked in the NCU.

25.18.3.2 Correctable NCU Etag Error

If a correctable Ctag error is detected by the NCU, the Ctag is corrected and bit 2, ncuctagce, in the SOC ESR is set. No other error information is provided.

25.19 PIO Store Errors

There are two classes of PIO store errors detected in the SOC, fatal errors (FE), which cause a warm reset, and uncorrectable errors (UE), which generate a *sw_recoverable_error*.

The PIO store errors are presented in TABLE 25-47.

TABLE 25-47 SOC Errors for a PIO Store Request

Error	SOCESR bit	Recommended Error Type	Error Info (core)	Error Info (NCU)	Trap
NCU uncorrectable on command or thread_id from PCX	ncupcxue{19}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (etag, rqtyp, cpu_id, thread_id, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
NCU store data parity	ncupcxdata{18}	UE	DESR = SOCU if SOC EIE bit set	NCUSYN (etag, rqtyp, cpu_id, thread_id, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
NCU DMU Credit parity	ncudmucredit{42}	UE	DESR = SOCU if SOC EIE bit set	Not Recorded	Warm reset if configured as fatal; otherwise, sw_recoverable_error if SOC EIE bit set
NCU FIFO output error	ncucpxue{20}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
NCU DMU data parity	ncudmuue{21}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (etag, reqtype, cpuid, strandid, pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.

25.19.1 Uncorrectable PIO Store Errors Recommended as Fatal

The following uncorrectable PIO Store errors are recommended to be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. Fatal errors associated with a PIO store cause the PIO store to be dropped and a warm reset to be initiated.

25.19.1.1 Uncorrectable NCU FIFO Errors (NcuPcxUE)

The NCU provides DEDSEC protection on the NCU FIFO. If an uncorrectable error is detected on the command or thread ID fields, bit 19, ncupcxue, in the SOC ESR is set. The error information (etag, rqtyp, cpu_id, thread_id, and PA{39:0}) is recorded in the NCUSYN register.

No store ACK is returned, so the requesting strand will become hung. Therefore this error needs to be fatal.

25.19.1.2 NCU FIFO Output Error (NcuCpxUe)

The NCU checks the output FIFO to the CPX. If the NCU detects an error in the output FIFO, bit 20, ncucpxue, in the SOC ESR is set. No other error information is captured.

No store ACK is returned to the requesting strand, which will become hung. Therefore this error needs to be fatal.

25.19.1.3 NCU Parity Error from NCU DMU PIO Req FIFO (NcuDmuUe)

If the NCU detects a parity error from the NCU DMU PIO Req FIFO on a PIO Store, bit 21, ncudmuue, in the SOC ESR is set. The error information (etag, rqtyp, cpu_id, thread_id and PA{39:0}) is recorded in the NCUSYN register. NCU returns store ACK on CPX packet without any error. NCU squashes the PIO store request to DMU.

However since the parity error can be in cpu_id[2:0] bits , the store ACK might happen to the wrong CPU. Hence this error needs to be fatal.

25.19.2 Uncorrectable PIO Store Errors

The following uncorrectable PIO Store errors are *not* recommended to be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. The SOC sends an Error Indication (SOC) packet to the CPX with the err field indicating uncorrectable error.

25.19.2.1 NCU Store Data Parity Error (NcuPcxData)

The NCU checks data parity on the store data. If a data parity error is detected, bit 18, ncupcxdata, in the SOC ESR is set. No other error information is provided.

25.19.2.2 NCU DMU Credit Parity (NcuDmuCredit)

If NCU detects a parity error on the token returned from DMU to NCU after completion of a PIO Store Request , bit 42, ncudmucredit, in the SOC ESR is set . No other error information is provided.

25.20 Interrupt Errors

Interrupts can be sourced from the DMU. There are three classes of interrupt errors detected in the SOC: fatal errors (FE), which cause a warm reset; uncorrectable errors (UE), which generate a *sw_recoverable_error* trap to the strand specified in the SOC Error Steering register; or correctable errors (CE), which generate a *hw_corrected_error* trap to the strand specified in the SOC Error Steering register. The SOC Error Steering register is described in *SOC Error Steering Register* on page 350. Interrupt errors are presented in TABLE 25-48.

TABLE 25-48 SOC Errors for Interrupts

Error	SOCESR bit	Recommended Error Type	Error Info (core)	Error Info (NCU)	Trap
SII command parity error or Ctag UE	siidmuue{1}	FE	DESR = SOCU if SOC EIE bit set	NCUSIISYN (etag, ctag, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU Ctag UE	ncuctague{22}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (ctag)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
DMU mondo ACK credit parity	dmuncucredit {9}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU data parity	ncudataparity14}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (ctag)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU mondo FIFO parity	ncumondofifo {16}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU interrupt table parity	ncuinttable{17}	FE	DESR = SOCU if SOC EIE bit set	NCUSYN (etag, reqtype, cpuid, strandid)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set

TABLE 25-48 SOC Errors for Interrupts

Error	SOCESR bit	Recommended Error Type	Error Info (core)	Error Info (NCU)	Trap
NCU FIFO output error	ncucpxue{20}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set
NCU mondo table parity	NCU Mondo Table{15}	UE	D-SFSR= SOCU, and DESR= SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *data_access_error*. and *sw_recoverable_error* if SOC EIE bit set
SII Ctag CE	siidmuce{3}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *hw_corrected_error* if SOC EIE bit set
NCU Ctag CE	ncuctagce{23}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *hw_corrected_error* if SOC EIE bit set

25.20.1 Uncorrectable Interrupt Errors Recommended as Fatal

The following uncorrectable Interrupt errors are recommended to be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. Fatal errors associated with an interrupt cause the interrupt to be dropped, and a warm reset to be initiated.

25.20.1.1 Uncorrectable SII Ctag/Command Parity Errors (SiiDmuCtagUE)

For interrupt requests (through the DMU) to the SII, the Ctag ECC is checked and also the command parity. If an uncorrectable Ctag ECC error or command parity error is detected, bit 1, siidmuctague, in the SOC ESR is set. The error information (etag, ctag and pa{39:0}) is recorded in the NCUSIISYN register.

For interrupts, this error condition does not have to be a fatal error, since the thread_id is not contained in the Ctag for interrupts. However, since the siidmuctague bit in the SOC ESR is shared with PIO loads, which do need to be fatal, the fatal error will also occur for interrupts when the siidmuctague bit is set.

25.20.1.2 Uncorrectable NCU Ctag Error (NcuCtagUe)

The NCU also checks Ctag ECC from the SII. If an uncorrectable Ctag ECC error is detected, bit 22, ncuctague, in the SOC ESR is set. The error information (etag, ctag) is recorded in the NCUSYN register.

For interrupts, this error condition does not have to be a fatal error, since the thread_id is not contained in the Ctag for interrupts. However, since the ncuctague bit in the SOC ESR is shared with PIO loads, which do need to be fatal, the fatal error will also occur for interrupts when the ncuctague bit is set.

25.20.1.3 DMU Mondo Ack Credit Parity(DmuNcuCredit)

DMU checks parity of Mondo Ack Credit from NCU. If a parity error is detected, bit 9, dmuncucredit , in the SOC ESR is set. There is no other information captured.

DMU Interrupt control unit keeps waiting for Ack forever, no future interrupt requests can be issued. So this needs to be a fatal error.

25.20.1.4 NCU Data Parity Error (NcuDataParity)

The NCU checks data parity for the interrupt requests from the DMU (via the SII). If a data parity error is detected, bit 14, ncudataparity, in the SOC ESR is set. The error information (Etag, Ctag) is recorded in the NCUSYN register.

The interrupt is lost, so this needs to be a fatal error.

25.20.1.5 NCU FIFO Output Error (NcuCpxUe)

The NCU checks the output FIFO to the CPX. If the NCU detects an error in the output FIFO, bit 20, ncucpxue, in the SOC ESR is set. There is no other error information captured.

The interrupt is lost, so this needs to be a fatal error.

25.20.1.6 NCU Mondo FIFO Parity Error (NcuMondoFifo)

The NCU checks the data parity of the Mondo FIFO during interrupt processing. If a data parity error is detected during the read of the NCU Mondo FIFO, bit 16, ncumondofifo, in the SOC ESR is set. No other error information is captured.

NCU does not send data return CPX packet for load, so thread hangs. So this needs to be a fatal error.

25.20.1.7 NCU Interrupt Table Parity Error (NcuIntTable)

The NCU checks the data parity of the NCU Interrupt table during the processing of a non-Mondo interrupt. If a data parity error is detected during the read of the NCU Interrupt table, bit 17, ncuinttable, in the SOC ESR is set. The error information (etag, rqtyp, cpu_id, thread_id) is recorded in the NCUSYN register.

The interrupt is lost, so this needs to be a fatal error.

25.20.2 Uncorrectable Interrupt Errors

All uncorrectable Interrupt errors other than ncumondotable are recommended to be set as fatal. For uncorrectable SOC detected errors associated with an interrupt, the SOC sends the error Indication (SOC) packet to the CPX with the err field indicating uncorrectable error. The packet is sent to the target thread of the interrupt.

25.20.2.1 NCU Mondo Table Parity Error (NcuMondoTable)

NCU checks for data parity error while reading the Mondo table for a PIO read request.If the NCU detects an error, bit 15, ncumondotable, in the SOC ESR is set. There is no other error information captured. PIO load returns UE on the CPX packet to the requesting core.

25.20.3 Correctable Interrupt Errors

The following correctable Interrupt errors are recommended *not* to be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For correctable SOC detected errors associated with an interrupt, the SOC sends the Error Indication (SOC) packet to the CPX with the err field indicating correctable error. The packet is sent to the target thread of the interrupt

25.20.3.1 Correctable SII Ctag Error (SiiDmuCtagCE)

If a correctable Ctag error is detected, the Ctag is corrected and bit 3, siidmuctagce, in the SOCESR is set. No other error information is provided. Data parity is not checked at this point. It will be checked in the NCU.

25.20.3.2 Correctable NCU Ctag Error (NCUCtagCE)

If a correctable Ctag error is detected by the NCU, the Ctag is corrected and bit 23, ncuctagce, in the SOC ESR is set. No other error information is provided.

25.21 DMA Reads and Writes

I/O DMA requests to the L2 cache can be made from the DMU. There are three classes of DMA errors detected in the SOC: fatal errors (FE), which cause a warm reset; uncorrectable errors (UE), which generate a *sw_recoverable_error* trap to the strand specified in the SOC Error Steering register; or correctable errors (CE), which generate a *hw_corrected_error* trap to the strand specified in the SOC Error Steering register. The SOC Error Steering register is described in *SOC Error Steering Register* on page 350.

For DMA writes, errors can be detected in the SII section of the SOC and the L2 cache. See the L2 cache section for details on DMA errors.

For DMA reads, errors can be detected in the SII, SIO, and DMU sections of the SOC, and the L2 cache.

For DMA errors, the transaction continues to completion for correctable and uncorrectable errors. The e and ue bits in the I/O block packet indicate uncorrectable errors. For fatal errors, the transaction continues until the warm reset is effective.

TABLE 25-49 SOC Errors for DMA Read and Write Request

Error	SOC ESR bit	Recommended Error Type	Error Info (Core)	Error Info (NCU)	Trap
SII command parity error or Ctag UE from DMU	siidmuue{1}	FE	DESR = SOCU if SOC EIE bit set	NCUSIISYN (etag, ctag, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
SIO Ctag UE	sioctague{25}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set .
DMU Ctag UE	dmuctague{11}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
DMU Credit parity from SII	dmusiicredit{12}	FE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
SII address parity from DMU	siidmuaparity{7}	FE	DESR = SOCU if SOC EIE bit set	NCUSIISYN (etag, ctag, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
SII data parity from DMU	siidmudparity{5}	UE	DESR = SOCU if SOC EIE bit set	NCUSIISYN (etag, ctag, and pa)	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.

TABLE 25-49 SOC Errors for DMA Read and Write Request *(Continued)*

Error	SOC ESR bit	Recommended Error Type	Error Info (Core)	Error Info (NCU)	Trap
DMU Data parity from SIO	dmudataparity{13}	UE	DESR = SOCU if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *sw_recoverable_error* if SOC EIE bit set.
SIO Ctag CE	sioctagce{26}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, hw_corrected_error if SOC EIE bit set.
SII Ctag CE from DMU	siidmuce{3}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *hw_corrected_error* if SOC EIE bit set.
DMU Ctag CE	dmuctagce{10}	CE	DESR = SOCC if SOC EIE bit set	Not recorded	Warm reset if configured as fatal; otherwise, *hw_corrected_error* if SOC EIE bit set.

25.21.1 Uncorrectable DMA Errors Recommended as Fatal

The following uncorrectable DMA errors are recommended to be set as fatal in the SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For fatal errors, the DMA transaction attempts to continue (with the write being quashed) until the warm reset occurs.

25.21.1.1 Uncorrectable SII Ctag ECC Error or Command Parity Error (SiiDmuCtagUe, SiiNiuCtagUe)

If an uncorrectable Ctag ECC error or command parity error is detected by the SII on a DMA access from the DMU, a fatal error is presented. The DMA access attempts to complete by setting the e bit in the packet to the L2 cache; however, the DMA write is quashed. If the DMA request is sourced from the DMU, the bit 1, siidmuctague, in the SOC ESR is set. The error information (etag, ctag and pa[39:0]) is recorded in the NCUSIISYN register.

This error does not have to be fatal. However, the siidmuctague bit is shared with the PIO load access, which must be fatal. The siiniuctagce bit is made fatal for consistency.

25.21.1.2 Uncorrectable SIO Ctag ECC Error (SioCtagUe)

The L2 cache provides a response to the SIO for DMA reads, DMA 8-byte writes and write invalidates. If an uncorrectable Ctag ECC error is detected by the SIO on a DMA access, bit 25, sioctague, in the SOC ESR is set. No other error information is provided in the SOC. This error is recommended as fatal because it would cause indeterministic machine behavior.

25.21.1.3 Uncorrectable DMU Ctag ECC Error (DmuCtagUe)

If an uncorrectable Ctag ECC error is detected by the DMU on a DMA read access from the SIO, bit 11, dmuctague, in the SOC ESR is set. No other error information is provided in the SOC. This error is recommended as fatal because it would cause indeterministic machine behavior.

25.21.1.4 DMU Credit Parity error (DmuSiiCredit)

If the DMU detects a parity error in the credit field during a DMA write acknowledge from the SII, bit 12, dmusiicredit, in the SOC ESR is set. No other error information is provided in the SOC.This error is recommended as fatal because it would cause indeterministic machine behavior.

25.21.1.5 SII Address Parity Error (SiiDmuAParity, SiiNiuAParity)

If an address parity error is detected by the SII on a DMA write from the DMU, the write is squashed. For a DMA read, the e bit is set in the packet sent to the L2 cache. If the DMA request is sourced from the DMU, bit 7, siidmuaparity, in the SOC ESR is set. The error information (etag, ctag, and PA{39:0}) is recorded in the NCUSIISYN register. This error is recommended as fatal because it would cause indeterministic machine behavior.

25.21.2 Uncorrectable DMA Errors

The following uncorrectable DMA errors are recommended to *not* be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For uncorrectable SOC detected errors associated with a DMA read or write, the transaction continues with the DMA write being squashed in the L2 cache; the DMA read continues with the e bit set in the I/O Block header for address parity errors, and the ue bit set for data parity errors.

The SOC sends the Error Indication (SOC) packet to the CPX with the err field indicating uncorrectable error. The packet is sent to the target thread of the interrupt.

25.21.2.1 SII Data Parity Error (SiiDmuDParity, SiiNiuDParity)

If a data parity error is detected by the SII on a 64 byte DMA write fromr the DMU, data is corrupted (poisoned) by flipping the two least significant bits of the Data ECC field. If the data is sourced from the DMU, bit 5, siidmudparity, in the SOC ESR is set. The error information (etag, ctag, and PA{39:0}) is recorded in the NCUSIISYN register.

However if there is a data parity error on a partial DMA write (less than or equal to 8 bytes) from the DMU, SII cannot detect the error and poison the data going to L2. As a result, silent data corruption happens in L2 and eventually memory.

25.21.2.2 DMU Data Parity Error (DmuDataParity)

On a DMA read access, if a data parity error is detected by the DMU on the data from the SIO, bit 13, dmudataparity, in the SOC ESR is set. No other error information is provided in the SOC.

25.21.3 Correctable DMA Errors

The following correctable DMA errors are recommended to *not* be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. For correctable SOC detected errors associated with a DMA access, the SOC sends the Error Indication (SOC) packet to the CPX with the err field indicating correctable error. The packet is sent to the thread indicated in the Error Steering register field of the NCU Strand Enable Status register.

For correctable errors, the DMA read or write operation completes.

25.21.3.1 Correctable SII Ctag ECC Error (SiiDmuCtagCe, SiiNiuCtagCe)

If a correctable Ctag ECC error is detected by the SII on a DMA access from the DMU, the access completes. If the DMA request is sourced from the DMU, bit 3, siidmuctagce, in the SOC ESR is set. No other error information is provided in the SOC.

25.21.3.2 Correctable SIO Ctag ECC Error (SioCtagCe)

If a correctable Ctag ECC error is detected by the SIO on a DMA access from the DMU, the access completes, and bit 26, sioctagce, in the SOC ESR is set. No other error information is provided in the SOC.

25.21.3.3 Correctable DMU Ctag ECC Error (DmuCtagCe)

If a correctable Ctag ECC error is detected by the DMU on a DMA read access, the access completes, and bit 10, dmuctagce, in the SOC ESR is set. No other error information is provided in the SOC.

25.22 MCU Correctable/Recoverable Count Errors

The MCU correctable/recoverable count errors are recommended to *not* be set as fatal in SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. There are four MCU FBdimms (MCU0 –MCU3. Each MCU FBdimm has an associated MCU Syndrome register, MCU ECC Correctable Error Count register, MCU Recoverable Error Count register, and associated error bits in the SOC ESR.

When the count registers transition from 1 to 0, an error signal is sent to the SOC. The SOC sets the corresponding bit in the SOC ESR, and sends the Error Indication (SOC) packet to the CPX with the err field indicating correctable error. The packet is sent to the thread indicated by the SOC Error Steering register.

TABLE 25-50 SOC Errors for MCU Error Count Interrupts

Error	SOCESR Bit	Recommended Error Type	Error Info (Core)	Error Info (MCU)	Trap
MCU0 correctable ECC error count reg = 0	mcu0ecc{32}	CE	DESR = SOCC	DRAM ESR0 (dsc, dac)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.
MCU1 correctable ECC error count reg = 0	mcu1ecc{35}	CE	DESR = SOCC	DRAM ESR1 (dsc, dac)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.
MCU2 correctable ECC error count reg = 0	mcu2ecc{38}	CE	DESR = SOCC	DRAM ESR2 (dsc, dac)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.
MCU3 correctable ECC error count reg = 0	mcu3ecc{41}	CE	DESR = SOCC	DRAM ESR3 (dsc, dac)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.
MCU0 recoverable error count reg = 0	mcu0fbr{31}	CE	DESR = SOCC	DRAM FDB SYN REG0 (alert, soft reset, fast reset, error source)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.

TABLE 25-50 SOC Errors for MCU Error Count Interrupts *(Continued)*

Error	SOCESR Bit	Recommended Error Type	Error Info (Core)	Error Info (MCU)	Trap
MCU1 recoverable error count reg = 0	mcu1fbr{34}	CE	DESR = SOCC	DRAM FDB SYN REG1 (alert, soft reset, fast reset, error source)	Warm reset PF if configured as fatal; otherwise, *hw_corrected_error*.
MCU2 recoverable error count reg = 0	mcu2fbr{37}	CE	DESR = SOCC	DRAM FDB SYN REG2 (alert, soft reset, fast reset, error source)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.
MCU3 recoverable error count reg = 0	mcu3fbr{40}	CE	DESR = SOCC	DRAM FDB SYN REG3 (alert, soft reset, fast reset, error source)	Warm reset if configured as fatal; otherwise, *hw_corrected_error*.

25.22.1 MCU ECC Correctable Errors (Mcu0ECC, Mcu1ECC, Mcu2ECC, Mcu3ECC)

Each MCU maintains a DRAM_ERROR_COUNT_REGISTER for counting correctable ECC errors and generating an error signal when the count decrements to zero.

When the count transitions from 1 to 0, the corresponding error bit in the SOC ESR is set: bit 32, mcu0ecc, for MCU0; bit 35, mcu1ecc, for MCU1; bit 38, mcu2ecc, for MCU2; and bit 41, mcu3ecc, for MCU3. Additional error information (DSC, DAC) is provided in the DRAM Error Status register in the corresponding MCU.

25.22.2 MCU Recoverable Errors (Mcu0Fbr, Mcu1Fbr, Mcu2Fbr, Mcu3Fbr)

Each MCU maintains a DRAM_FBR_COUNT_REG for counting recoverable link errors and generating an error signal when the count decrements to zero.

When the count transitions from 1 to 0 or if the count_one field in MCU Count register or MCU fbrcnt are set to 1, the corresponding error bit in the SOC ESR is set: bit 31, mcu0fbr, for MCU0; bit 34, mcu1fbr, for MCU1; bit 37, mcu2fb, for MCU2; and bit 40, mcu3fbr, for MCU3.

Additional error information (alert, soft reset, fast reset, error source) is captured in the corresponding DRAM FDB Syndrome register.

25.23 SOC Error Registers

This section describes the error control and log registers for the SOC. The SOC error registers are located in the NCU.

Error injection for the SOC error sources are provided by the SOC Error Injection register. The SOC provides a double-buffered error register for *sw_recoverable_error* or *hw_corrected_error* handling. All errors that are enabled by the SOC Error Log Enable register are captured in the SOC Error Status register. In addition, when a *sw_recoverable_error* trap is generated, the contents of the SOC Error Status register are copied to the SOC Pending Error Status register, and the SOC Error Status register is cleared. Each error source can generate no trap, a *sw_recoverable_error* trap, a *hw_corrected_error* trap, or a warm reset of the OpenSPARC T2 chip under control of the SOC Error Interrupt Enable and SOC Fatal Error Enable registers.

25.23.1 SOC Error Status Register

The Error Status register contains status on SOC errors. The status bits in this register are cleared by writing a 0 to the bit position. This register is not changed on a warm reset to allow inspection of the error status by software following the warm reset.

Note | All SOC errors may be set to fatal using the SOC Fatal Error Enable register described in *SOC Fatal Error Enable Register* on page 351. The error type column lists the error type assuming the error has not been set to be fatal.

TABLE 25-51 shows the format of the SOC Error Status register.

TABLE 25-51 SOC Error Status Register – SOC_ERROR_STATUS_REG ($80\ 0000\ 3000_{16}$) *(1 of 4)*

Bit	Field	Error Type	Initial Value	R/W	Error Information (core)	Error Information (SOC)	Description
63	v	—	0	RW	—	—	Multiple uncorrected errors, one or more uncorrected errors were not logged.
62:43	SPARE4	—	0_{16}	RW	—	—	*Reserved* for errors to be assigned in future versions of chip if required
42	ncudmucredit	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if an uncorrectable parity error is detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	CE	0	RW	DESR = SOCC	DRAM ESR3 (dsc, dac)	Set to 1 if MCU 3 detected a correctable DRAM ECC error with its DRAM Error Count register reaching zero.
40	mcu3fbr	CE	0	RW	DESR = SOCC	DRAM FBD SYN REG3 (alert, soft reset, fast reset, error source)	Set to 1 if MCU 3 detected a FBDIMM recoverable error with its DRAM Recoverable Link Error Count register reaching zero.
39	SPARE3	—	0	RW	—	—	*Reserved* for errors to be assigned in future versions of chip if required
38	mcu2ecc	CE	0	RW	DESR = SOCC	DRAM ESR2 (dsc, dac)	Set to 1 if MCU 3 detected a correctable DRAM ECC error with its DRAM Error Count register reaching zero.
37	mcu2fbr	CE	0	RW	DESR = SOCC	DRAM FDB SYN REG2 (alert, soft reset, fast reset, error source)	Set to 1 if MCU 3 detected a FBDIMM recoverable error with its DRAM Recoverable Link Error Count register reaching zero.
36	SPARE2	—	0	RW	—	—	*Reserved* for errors to be assigned in future versions of chip if required
35	mcu1ecc	CE	0	RW	DESR = SOCC	DRAM ESR1 (dsc, dac)	Set to 1 if MCU 3 detected a correctable DRAM ECC error with its DRAM Error Count register reaching zero.
34	mcu1fbr	CE	0	RW	DESR = SOCC	DRAM FDB SYN REG1 (alert, soft reset, fast reset, error source)	Set to 1 if MCU 3 detected a FBDIMM recoverable error with its DRAM Recoverable Link Error Count register reaching zero.

TABLE 25-51 SOC Error Status Register – SOC_ERROR_STATUS_REG ($80\ 0000\ 3000_{16}$) *(2 of 4)*

Bit	Field	Error Type	Initial Value	R/W	Error Information (core)	Error Information (SOC)	Description
33	SPARE1	—	0	RW	—	—	*Reserved* for errors to be assigned in future versions of chip if required
32	mcu0ecc	CE	0	RW	DESR = SOCC	DRAM ESR0 (dsc, dac)	Set to 1 if MCU 0 detected a correctable DRAM ECC error with the DRAM Error Count register reaching zero.
31	mcu0fbr	CE	0	RW	DESR = SOCC	DRAM FDB SYN REG0 (alert, soft reset, fast reset, error source)	Set to 1 if MCU 0 detected a FBDIMM recoverable error with the DRAM Recoverable Link Error Count register reaching zero.
30	SPARE0	—	0	RW	—	—	*Reserved* for errors to be assigned in future versions of chip if required
29			0				
28			0				
27			0				
26	sioctagce	CE	0	RW	DESR = SOCC	Not recorded	Set to 1 if the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the SIO detected a CTAG uncorrected error from the old FIFO.
24	spare		0	RW	—	—	Hardware does not log any error for this or can inject any error for this, but asserts interrupt on software write.
23	ncuctagce	CE	0	RW	DESR = SOCC	Not recorded	Set to 1 if the NCU detected a CTAG corrected error on an interrupt write or a PIO read return from the SII.
22	ncuctague	UE	0	RW	DESR = SOCU	NCUSYN (ctag)	Set to 1 if the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return from the SII.
21	ncudmuue	UE	0	RW	D-SFSR = SOCU, DESR = SOCU	NCUSYN (etag, rqtyp, cpu_id, thread_id, and PA)	Set to 1 if the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the NCU detected an error in the output FIFO to the crossbar.

TABLE 25-51 SOC Error Status Register – SOC_ERROR_STATUS_REG ($80\ 0000\ 3000_{16}$) *(3 of 4)*

Bit	Field	Error Type	Initial Value	R/W	Error Information (core)	Error Information (SOC)	Description
19	ncupcxue	UE	0	RW	DESR = SOCU	NCUSYN (etag, rqtyp, cpu_id, thread_id, and PA)	Set to 1 if the NCU detected a command or Thread_id parity error in PIO/CSR commands from the processors (PCX FIFO).
18	ncupcxdata	UE	0	RW	DESR = SOCU	NCUSYN (etag, rqtyp, cpu_id, thread_id, and PA)	Set to 1 if the NCU detected a data parity error in PIO/CSR data from the processors (PCX FIFO).
17	ncuinttable	UE	0	RW	DESR = SOCU	NCUSYN (etag, rqtyp, cpu_id, thread_id, and pa)	Set to 1 if the NCU detected an error while reading the interrupt table for a non-mondo interrupt request.
16	ncumondofifo	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the NCU detected an error while reading the mondo FIFO for an interrupt request.
15	ncumondotable	UE	0	RW	D-SFSR = SOCU, DESR = SOCU	Not recorded	Set to 1 if the NCU detected an error while reading the mondo table for a PIO read request.
14	ncudataparity	UE	0	RW	D-SFSR = SOCU, DESR = SOCU	NCUSYN (ctag)	Set to 1 if the NCU detected a data parity error for interrupt write or PIO read return data through the SII from the DMU.
13	dmudataparity	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the DMU detected a data parity error in a DMA read return from the SIO.
12	dmusiicredit	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the DMU detected an uncorrected CTAG error in the DMA read return from the SIO.
10	dmuctagce	CE	0	RW	DESR = SOCC	Not recorded	Set to 1 if the DMU detected a corrected CTAG error in the DMA read return from the SIO.
9	dmuncucredit	UE	0	RW	DESR = SOCU	Not recorded	Set to 1 if the DMU detected a parity error in the mondo acknowledge credit from NCU.
8	dmuinternal	UE	0	RW	Not recorded	Not recorded	Set to 1 if the DMU detected an internal error.

TABLE 25-51 SOC Error Status Register – SOC_ERROR_STATUS_REG ($80\ 0000\ 3000_{16}$) *(4 of 4)*

Bit	Field	Error Type	Initial Value	R/W	Error Information (core)	Error Information (SOC)	Description
7	siidmuaparity	UE	0	RW	DESR = SOCU	NCUSIISYN (etag, ctag, and pa)	Set to 1 if the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6			0				
5	siidmudparity	UE	0	RW	DESR = SOCU	NCUSIISYN (etag, ctag, and pa)	Set to 1 if the SII detected a parity error on data for DMA write transactions from DMU FIFO.
4			0				
3	siidmuctagce	CE	0	RW	DESR = SOCC	Not recorded	Set to 1 if the SII detected a corrected ECC CTAG error on a transaction from the DMU FIFO.
2			0				
1	siidmuctague	UE	0	RW	DESR = SOCU	NCUSIISYN (etag, ctag, and pa)	Set to 1 if the SII detected an uncorrectable ECC CTAG error or Command Parity error on a transaction from the DMU FIFO.
0			0				

Once set, error status bits are only cleared by software. Hardware will never clear a set status bit. If a software write of the error register happens on the same cycle as an error, the setting of bits by the error will be based on the register state before the write. The setting of fields by the error will take precedence over the same field being update by the write; however, fields that are not changed by the error will be updated by the write (for example, if the register has the ncucpxue bit set and software does a write to clear that bit and the v bit on the same cycle as a NCUCTAGCE error, the error register would end up with the ncucpxue bit cleared and the ncuctagce and v bits set).

25.23.2 SOC Error Log Enable Register

This register enables the logging of SOC errors.

TABLE 25-52 shows the format of the SOC Error Log Enable register.

TABLE 25-52 SOC Error Log Enable Register – SOC_ERROR_LOG_ENABLE_REG ($80\ 0000\ 3008_{16}$) *(1 of 2)*

Bit	Field	Initial Value	R/W	Description
63:43	SPARE4	0_{16}	RW	*Reserved* (for errors to be assigned in future versions of chip if required).
42	ncudmucredit	1_{16}	RW	Set to 1 to log an uncorrectable parity error is detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	1_{16}	RW	Set to 1 to log MCU 3 exceeded data CE threshold.
40	mcu3fbr	1_{16}	RW	Set to 1 to log MCU 3 generated a FBDIMM recoverable error.
39	SPARE3	1_{16}	RW	*Reserved* (for errors to be assigned in future versions of chip if required).
38	mcu2ecc	1_{16}	RW	Set to 1 to log MCU 2 exceeded data CE threshold.
37	mcu2fbr	1_{16}	RW	Set to 1 to log MCU 2 generated a FBDIMM recoverable error.
36	SPARE2	1_{16}	RW	*Reserved* (for errors to be assigned in future versions of chip if required).
35	mcu1ecc	1_{16}	RW	Set to 1 to log MCU 1 exceeded data CE threshold.
34	mcu1fbr	1_{16}	RW	Set to 1 to log MCU 1 generated a FBDIMM recoverable error.
33	SPARE1	1_{16}	RW	*Reserved* (for errors to be assigned in future versions of chip if required).
32	mcu0ecc	1_{16}	RW	Set to 1 to log MCU 0 exceeded data CE threshold.
31	mcu0fbr	1_{16}	RW	Set to 1 to log MCU 0 generated a FBDIMM recoverable error.
30	SPARE0	1_{16}	RW	Spare 0 (for errors to be assigned in future versions of chip if required).
29		1_{16}		
28		1_{16}		Set to 1 to log the NIU detected a CTAG uncorrected error in the DMA read return from the SIO.
27		1_{16}		
26	sioctagce	1_{16}	RW	Set to 1 to log the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	1_{16}	RW	Set to 1 to log the SIO detected a CTAG uncorrected error from the old FIFO.
24	SPARE	1_{16}	RW	Hardware does not log any error for this or can inject any error for this, but asserts interrupt on software write.
23	ncuctagce	1_{16}	RW	Set to 1 to log the NCU detected a CTAG corrected error on an interrupt write or a PIO read return.
22	ncuctague	1_{16}	RW	Set to 1 to log the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return.
21	ncudmuue	1_{16}	RW	Set to 1 to log the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	1_{16}	RW	Set to 1 to log the NCU detected an error in the Output FIFO to the crossbar.
19	ncupcxue	1_{16}	RW	Set to 1 to log the NCU detected an error in PIO/CSR commands from the processors.
18	ncupcxdata	1_{16}	RW	Set to 1 to log the NCU detected an error in PIO/CSR data from the processors.
17	ncuinttable	1_{16}	RW	Set to 1 to log the NCU detected an error while reading the interrupt table.

TABLE 25-52 SOC Error Log Enable Register – SOC_ERROR_LOG_ENABLE_REG ($80\ 0000\ 3008_{16}$) *(2 of 2)*

Bit	Field	Initial Value	R/W	Description
16	ncumondofifo	1_{16}	RW	Set to 1 to log the NCU detected an error while reading the mondo FIFO.
15	ncumondotable	1_{16}	RW	Set to 1 to log the NCU detected an error while reading the mondo table.
14	ncudataparity	1_{16}	RW	Set to 1 to log the NCU detected an parity error for interrupt write or PIO read return data.
13	dmudataparity	1_{16}	RW	Set to 1 to log the DMU detected a parity error in a DMA read return from the SIO.
12	dmusiicredit	1_{16}	RW	Set to 1 to log the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	1_{16}	RW	Set to 1 to log the DMU detected an uncorrected error in the DMA read return from the SIO.
10	dmuctagce	1_{16}	RW	Set to 1 to log the DMU detected a corrected error in the DMA read return from the SIO.
9	dmuncucredit	1_{16}	RW	Set to 1 to log the DMU detected a parity error in the Mondo acknowledge credit from NCU.
8	dmuinternal	1_{16}	RW	Set to 1 to log the DMU detected an internal error.
7	siidmuaparity	1_{16}	RW	Set to 1 to log the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6		1_{16}		
5	siidmudparity	1_{16}	RW	Set to 1 to log the SII detected a parity error on data for DMA transactions from DMU FIFO.
4		1_{16}		
3	siidmuctagce	1_{16}	RW	Set to 1 to log the SII detected a corrected error on a transaction from the DMU FIFO.
2		1_{16}		
1	siidmuctague	1_{16}	RW	Set to 1 to log the SII detected an uncorrected error on a transaction from the DMU FIFO.
0		1_{16}		

25.23.3 SOC Error Interrupt Enable Register

This register controls which errors will generate a Error Indication (SOC) error packet to the CPX. If the eie bit is set, Error Indication (SOC) error packet will always be sent to the CPX irrespective of whether the error caused the transaction to be terminated or not. The Correctable SOC errors will set an error code of 01_2 in the err field of the packet while uncorrectable SOC errors set an error code of 10_2 in the err field.

TABLE 25-53 shows the format of the SOC Error Interrupt Enable register.

TABLE 25-53 SOC Error Interrupt Enable Register – SOC_ERROR_INTERRUPT_ENABLE_REG ($80\ 0000\ 3010_{16}$)

Bit	Field	Initial Value	R/W	Description
62:43	SPARE4	0	RW	Set to 1 to interrupt errors for whatever error bits would get assigned to bits 62:43 in future versions of chip.
42	ncudmucredit	0	RW	Set to 1 to interrupt on an uncorrectable parity error is detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	0	RW	Set to 1 to interrupt on MCU 3 exceeded data CE threshold.
40	mcu3fbr	0	RW	Set to 1 to interrupt on MCU 3 generated a FBDIMM recoverable error.
39	spare3	0	RW	Set to 1 to interrupt error for whatever error bit would get assigned to bits 39 in future versions of chip.
38	mcu2ecc	0	RW	Set to 1 to interrupt on MCU 2 exceeded data CE threshold.
37	mcu2fbr	0	RW	Set to 1 to interrupt on MCU 2 generated a FBDIMM recoverable error.
36	SPARE2	0	RW	Set to 1 to interrupt error for whatever error bit would get assigned to bits 36 in future versions of chip.
35	mcu1ecc	0	RW	Set to 1 to interrupt on MCU 1 exceeded data CE threshold.
34	mcu1fbr	0	RW	Set to 1 to interrupt on MCU 1 generated a FBDIMM recoverable error.
33	SPARE1	0	RW	Set to 1 to interrupt error for whatever error bit would get assigned to bits 33 in future versions of chip
32	mcu0ecc	0	RW	Set to 1 to interrupt on MCU 0 exceeded data CE threshold.
31	mcu0fbr	0	RW	Set to 1 to interrupt on MCU 0 generated a FBDIMM recoverable error.
30	SPARE0	0	RW	Set to 1 to interrupt error for whatever error bit would get assigned to bits 30 in future versions of chip
29		0		
28		0		
27		0		
26	sioctagce	0	RW	Set to 1 to interrupt on the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	0	RW	Set to 1 to interrupt on the SIO detected a CTAG uncorrected error from the old FIFO.
24	spare	0	RW	HW does not log any error for this or can inject any error for this, but asserts interrupt on SW write.
23	ncuctagce	0	RW	Set to 1 to interrupt on the NCU detected a CTAG corrected error on an interrupt write or a PIO read return.
22	ncuctague	0	RW	Set to 1 to interrupt on the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return.
21	ncudmuue	0	RW	Set to 1 to interrupt on the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	0	RW	Set to 1 to interrupt on the NCU detected an error in the output FIFO to the crossbar.
19	ncupcxue	0	RW	Set to 1 to interrupt on the NCU detected an error in PIO/CSR commands from the processors.

TABLE 25-53 SOC Error Interrupt Enable Register – SOC_ERROR_INTERRUPT_ENABLE_REG ($80\ 0000\ 3010_{16}$) *(Continued)*

Bit	Field	Initial Value	R/W	Description
18	ncupcxdata	0	RW	Set to 1 to interrupt on the NCU detected an error in PIO/CSR data from the processors.
17	ncuinttable	0	RW	Set to 1 to interrupt on the NCU detected an error while reading the interrupt table.
16	ncumondofifo	0	RW	Set to 1 to interrupt on the NCU detected an error while reading the mondo FIFO.
15	ncumondotable	0	RW	Set to 1 to interrupt on the NCU detected an error while reading the mondo table.
14	ncudataparity	0	RW	Set to 1 to interrupt on the NCU detected an parity error for interrupt write or PIO read return data.
13	dmudataparity	0	RW	Set to 1 to interrupt on the DMU detected a parity error in a DMA read return from the SIO.
12	dmusiicredit	0	RW	Set to 1 to interrupt on the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	0	RW	Set to 1 to interrupt on the DMU detected an uncorrected error in the DMA read return from the SIO.
10	dmuctagce	0	RW	Set to 1 to interrupt on the DMU detected a corrected error in the DMA read return from the SIO.
9	dmuncucredit	0	RW	Set to 1 to interrupt on the DMU detected a parity error in the Mondo acknowledge credit from NCU.
8	dmuinternal	0	RW	Set to 1 to interrupt on the DMU detected an internal error.
7	siidmuaparity	0	RW	Set to 1 to interrupt on the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6		0		
5	siidmudparity	0	RW	Set to 1 to interrupt on the SII detected a parity error on data for DMA transactions from DMU FIFO.
4		0		
3	siidmuctagce	0	RW	Set to 1 to interrupt on the SII detected a corrected error on a transaction from the DMU FIFO.
2		0		
1	siidmuctague	0	RW	Set to 1 to interrupt on the SII detected an uncorrected error on a transaction from the DMU FIFO.
0		0		

25.23.4 SOC Error Steering Register

This register controls which virtual processor will be sent SOC error interrupts.

TABLE 25-53 shows the format of the SOC Error Steering register.

TABLE 25-54 SOC Error Steering Register – SOC_ERROR_STEER_REG ($90\ 0104\ 1000_{16}$)

Bit	Field	Initial Value	R/W	Description
62:6	—	0	RO	*Reserved*
5:0	vcid	0	RW	ID of virtual processor that will be target of SOC error interrupts.

Notes

Software should program the Error Steering register the same in NCU and L2 so that multiple errors for the same error reported by both L2 and NCU go to same thread.

For errors reported multiple times (through PIO load return precise trap on crossbar and then SOC error packet and disruptive trap) the vcid can be different for the traps.

25.23.5 SOC Fatal Error Enable Register

This register controls which errors will generate a fatal error, resetting OpenSPARC T2.

TABLE 25-55 shows the format of the SOC Fatal Error Enable register.

TABLE 25-55 SOC Fatal Error Enable Register – SOC_FATAL_ERROR_ENABLE_REG ($80\ 0000\ 3020_{16}$) *(1 of 3)*

Bit	Field	Initial Value	R/W	Description
62:43	SPARE4	0	RW	Set to 1 to make fatal for whatever error bit would get assigned to bits 62:43 in future versions of chip
42	ncudmucredit	0	RW	Set to 1 to make fatal an uncorrectable parity error detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	0	RW	Set to 1 to make fatal MCU 3 exceeded data CE threshold.
40	mcu3fbr	0	RW	Set to 1 to make fatal MCU 3 generate an FBDIMM recoverable error.
39	SPARE3	0	RW	Set to 1 to make fatal whatever error bit would get assigned to bits 39 in future versions of chip
38	mcu2ecc	0	RW	Set to 1 to make fatal MCU 2 exceeded data CE threshold.
37	mcu2fbr	0	RW	Set to 1 to make fatal MCU 2 generated a FBDIMM recoverable error.
36	SPARE2	0	RW	Set to 1 to make fatal for whatever error bit would get assigned to bits 36 in future versions of chip
35	mcu1ecc	0	RW	Set to 1 to make fatal MCU 1 exceeded data CE threshold.
34	mcu1fbr	0	RW	Set to 1 to make fatal MCU 1 generated a FBDIMM recoverable error.
33	SPARE1	0	RW	Set to 1 to make fatal for whatever error bit would get assigned to bits 33 in future versions of chip
32	mcu0ecc	0	RW	Set to 1 to make fatal MCU 0 exceeded data CE threshold.

TABLE 25-55 SOC Fatal Error Enable Register – SOC_FATAL_ERROR_ENABLE_REG ($80\ 0000\ 3020_{16}$) *(2 of 3)*

Bit	Field	Initial Value	R/W	Description
31	mcu0fbr	0	RW	Set to 1 to make fatal MCU 0 generated a FBDIMM recoverable error.
30	SPARE0	0	RW	Set to 1 to make fatal for whatever error bit would get assigned to bits 30 in future versions of chip.
29		0		
28		0		
27		0		
26	sioctagce	0	RW	Set to 1 to make fatal the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	0	RW	Set to 1 to make fatal the SIO detected a CTAG uncorrected error from the old FIFO.
24	spare	0	RW	HW does not log any error for this or can inject any error for this, but asserts interrupt on SW write.
23	ncuctagce	0	RW	Set to 1 to make fatal the NCU detected a CTAG corrected error on an interrupt write or a PIO read return.
22	ncuctague	0	RW	Set to 1 to make fatal the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return.
21	ncudmuue	0	RW	Set to 1 to make fatal the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	0	RW	Set to 1 to make fatal the NCU detected an error in the output FIFO to the crossbar.
19	ncupcxue	0	RW	Set to 1 to make fatal the NCU detected an error in PIO/CSR commands from the processors.
18	ncupcxdata	0	RW	Set to 1 to make fatal the NCU detected an error in PIO/CSR data from the processors.
17	ncuinttable	0	RW	Set to 1 to make fatal the NCU detected an error while reading the interrupt table.
16	ncumondofifo	0	RW	Set to 1 to make fatal the NCU detected an error while reading the mondo FIFO.
15	ncumondotable	0	RW	Set to 1 to make fatal the NCU detected an error while reading the mondo table.
14	ncudataparity	0	RW	Set to 1 to make fatal the NCU detected an parity error for interrupt write or PIO read return data.
13	dmudataparity	0	RW	Set to 1 to make fatal the DMU detected a parity error in a DMA read return from the SIO.
12	dmusiicredit	0	RW	Set to 1 to make fatal the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	0	RW	Set to 1 to make fatal the DMU detected an uncorrected error in the DMA read return from the SIO.
10	dmuctagce	0	RW	Set to 1 to make fatal the DMU detected a corrected error in the DMA read return from the SIO.

TABLE 25-55 SOC Fatal Error Enable Register – SOC_FATAL_ERROR_ENABLE_REG ($80\ 0000\ 3020_{16}$) *(3 of 3)*

Bit	Field	Initial Value	R/W	Description
9	dmuncucredit	0	RW	Set to 1 to make fatal the DMU detected a parity error in the Mondo acknowledge credit from NCU.
8	dmuinternal	0	RW	Set to 1 to make fatal the DMU detected an internal error.
7	siidmuaparity	0	RW	Set to 1 to make fatal the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6		0		
5	siidmudparity	0	RW	Set to 1 to make fatal the SII detected a parity error on data for DMA transactions from DMU FIFO.
4		0		
3	siidmuctagce	0	RW	Set to 1 to make fatal the SII detected a corrected error on a transaction from the DMU FIFO.
2		0		
1	siidmuctague	0	RW	Set to 1 to make fatal the SII detected an uncorrected error on a transaction from the DMU FIFO.
0		0		

25.23.6 SOC Pending Error Status Register

The Pending Error Status register contains the state of the SOC_ERROR_STATUS_REG when the disrupting trap was generated as a result of an SOC error logged that had its corresponding bit set in SOC_ERROR_INTERRUPT_ENABLE_REG. The valid bit of this register prevents further disrupting traps from being generated by the SOC. This register is not changed on a warm reset to allow inspection of the error status by software following the warm reset.

TABLE 25-56 shows the format of the SOC Pending Error Status register.

TABLE 25-56 SOC Pending Error Status Register – SOC_PENDING_ERROR_STATUS_REG ($80\ 0000\ 3028_{16}$) *(1 of 3)*

Bit	Field	Initial Value	R/W	Description
63	v	0	RW	Valid bit, prevents generation of further SOC *sw_recoverable_error* traps.
62:43	SPARE4	0	RW	*Reserved* for future versions of the chip.
42	ncudmucredit	0	RW	Set to 1 to if an uncorrectable parity error is detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	0	RW	Set to 1 if MCU 3 exceeded data CE threshold.
40	mcu3fbr	0	RW	Set to 1 if MCU 3 generated a FBDIMM recoverable error.
39	SPARE3	0	RW	*Reserved* for future versions of the chip.

TABLE 25-56 SOC Pending Error Status Register – SOC_PENDING_ERROR_STATUS_REG ($80\ 0000\ 3028_{16}$) *(2 of 3)*

Bit	Field	Initial Value	R/W	Description
38	mcu2ecc	0	RW	Set to 1 if MCU 2 exceeded data CE threshold.
37	mcu2fbr	0	RW	Set to 1 if MCU 2 generated a FBDIMM recoverable error.
36	SPARE2	0	RW	*Reserved* for future versions of the chip.
35	mcu1ecc	0	RW	Set to 1 if MCU 1 exceeded data CE threshold.
34	mcu1fbr	0	RW	Set to 1 if MCU 1 generated a FBDIMM recoverable error.
33	spare1	0	RW	*Reserved* for future versions of the chip.
32	mcu0ecc	0	RW	Set to 1 if MCU 0 exceeded data CE threshold.
31	mcu0fbr	0	RW	Set to 1 if MCU 0 generated a FBDIMM recoverable error.
30	SPARE0	0	RW	*Reserved* for future versions of the chip.
29		0		
28		0		
27		0		
26	sioctagce	0	RW	Set to 1 if the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	0	RW	Set to 1 if the SIO detected a CTAG uncorrected error from the old FIFO.
24	spare	0	RW	Hardware does not log any error for this or can inject any error for this, but asserts interrupt on SW write.
23	ncuctagce	0	RW	Set to 1 if the NCU detected a CTAG corrected error on an interrupt write or a PIO read return.
22	ncuctague	0	RW	Set to 1 if the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return.
21	ncudmuue	0	RW	Set to 1 if the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	0	RW	Set to 1 if the NCU detected an error in the output FIFO to the crossbar.
19	ncupcxue	0	RW	Set to 1 if the NCU detected an error in PIO/CSR commands from the processors.
18	ncupcxdata	0	RW	Set to 1 if the NCU detected an error in PIO/CSR data from the processors.
17	ncuinttable	0	RW	Set to 1 if the NCU detected an error while reading the interrupt table.
16	ncumondofifo	0	RW	Set to 1 if the NCU detected an error while reading the mondo FIFO.
15	ncumondotable	0	RW	Set to 1 if the NCU detected an error while reading the mondo table.
14	ncudataparity	0	RW	Set to 1 if the NCU detected an parity error for interrupt write or PIO read return data.
13	dmudataparity	0	RW	Set to 1 if the DMU detected a parity error in a DMA read return from the SIO.
12	dmusiicredit	0	RW	Set to 1 if the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	0	RW	Set to 1 if the DMU detected an uncorrected error in the DMA read return from the SIO.

TABLE 25-56 SOC Pending Error Status Register – SOC_PENDING_ERROR_STATUS_REG ($80\ 0000\ 3028_{16}$) *(3 of 3)*

Bit	Field	Initial Value	R/W	Description
10	dmuctagce	0	RW	Set to 1 if the DMU detected a corrected error in the DMA read return from the SIO.
9	dmuncucredit	0	RW	Set to 1 if the DMU detected a parity error in the Mondo acknowledge credit from NCU.
8	dmuinternal	0	RW	Set to 1 if the DMU detected an internal error.
7	siidmuaparity	0	RW	Set to 1 if the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6		0		
5	siidmudparity	0	RW	Set to 1 if the SII detected a parity error on data for DMA transactions from DMU FIFO.
4		0		
3	siidmuctagce	0	RW	Set to 1 if the SII detected a corrected error on a transaction from the DMU FIFO.
2		0		
1	siidmuctague	0	RW	Set to 1 if the SII detected an uncorrected error on a transaction from the DMU FIFO.
0		0		

25.23.7 SOC Error Injection Register

This register controls the injection of errors. Continuous errors are generated for any error types with their bit set in the register.

TABLE 25-57 shows the format of the SOC Error Injection register.

TABLE 25-57 SOC Error Injection Register – SOC_ERROR_INJECTION_REG ($80\ 0000\ 3018_{16}$) *(1 of 3)*

Bit	Field	Initial Value	R/W	Description
63:43	SPARE4	0	RW	*Reserved* for future versions of the chip.
42	ncudmucredit	0	RW	Set to 1 to enable error injection for an uncorrectable parity error is detected on the credit token bus to NCU for DMU PIO write credits.
41	mcu3ecc	0	RW	Set to 1 to enable error injection on MCU 3 exceeded data CE threshold.
40	mcu3fbr	0	RW	Set to 1 to enable error injection on MCU 3 generated a FBDIMM recoverable error.
39	mcu3fbu	0	RW	Set to 1 to enable error injection on MCU 3 generated a FBDIMM unrecoverable error.
38	mcu2ecc	0	RW	Set to 1 to enable error injection on MCU 2 exceeded data CE threshold.
37	mcu2fbr	0	RW	Set to 1 to enable error injection on MCU 2 generated a FBDIMM recoverable error.

TABLE 25-57 SOC Error Injection Register – SOC_ERROR_INJECTION_REG ($80\ 0000\ 3018_{16}$) *(2 of 3)*

Bit	Field	Initial Value	R/W	Description
36	mcu2fbu	0	RW	Set to 1 to enable error injection on MCU 2 generated a FBDIMM unrecoverable error.
35	mcu1ecc	0	RW	Set to 1 to enable error injection on MCU 1 exceeded data CE threshold.
34	mcu1fbr	0	RW	Set to 1 to enable error injection on MCU 1 generated a FBDIMM recoverable error.
33	mcu1fbu	0	RW	Set to 1 to enable error injection on MCU 1 generated a FBDIMM unrecoverable error.
32	mcu0ecc	0	RW	Set to 1 to enable error injection on MCU 0 exceeded data CE threshold.
31	mcu0fbr	0	RW	Set to 1 to enable error injection on MCU 0 generated a FBDIMM recoverable error.
30	mcu0fbu	0	RW	Set to 1 to enable error injection on MCU 0 generated a FBDIMM unrecoverable error.
29		0	RW	
28		0		
27		0		
26	sioctagce	0	RW	Set to 1 to enable error injection on the SIO detected a CTAG corrected error from the old FIFO.
25	sioctague	0	RW	Set to 1 to enable error injection on the SIO detected a CTAG uncorrected error from the old FIFO.
24	SPARE	0	RW	Hardware does not log any error for this or can inject any error for this, but asserts interrupt on software write.
23	ncuctagce	0	RW	Set to 1 to enable error injection on the NCU detected a CTAG corrected error on an interrupt write or a PIO read return.
22	ncuctague	0	RW	Set to 1 to enable error injection on the NCU detected a CTAG uncorrected error on an interrupt write or a PIO read return.
21	ncudmuue	0	RW	Set to 1 to enable error injection on the NCU detected a parity error in the NCU DMU PIO Req FIFO.
20	ncucpxue	0	RW	Set to 1 to enable error injection on the NCU detected an error in the output FIFO to the crossbar.
19	ncupcxue	0	RW	Set to 1 to enable error injection on the NCU detected an error in PIO/ CSR commands from the processors.
18	ncupcxdata	0	RW	Set to 1 to enable error injection on the NCU detected an error in PIO/ CSR data from the processors.
17	ncuinttable	0	RW	Set to 1 to enable error injection on the NCU detected an error while reading the interrupt table.
16	ncumondofifo	0	RW	Set to 1 to enable error injection on the NCU detected an error while reading the mondo FIFO.
15	ncumondotable	0	RW	Set to 1 to enable error injection on the NCU detected an error while reading the mondo table.
14	ncudataparity	0	RW	Set to 1 to enable error injection on the NCU detected an parity error for interrupt write or PIO read return data.

TABLE 25-57 SOC Error Injection Register – SOC_ERROR_INJECTION_REG ($80\ 0000\ 3018_{16}$) *(3 of 3)*

Bit	Field	Initial Value	R/W	Description
13	dmudataparity	0	RW	Set to 1 to enable error injection on the DMU detected a parity error in a DMA read return from the SIO.
12	dmusiicredit	0	RW	Set to 1 to enable error injection on the DMU detected a parity error in the DMA write acknowledge credit from the SII.
11	dmuctague	0	RW	Set to 1 to enable error injection on the DMU detected an uncorrected error in the DMA read return from the SIO.
10	dmuctagce	0	RW	Set to 1 to enable error injection on the DMU detected a corrected error in the DMA read return from the SIO.
9	dmuncucredit	0	RW	Set to 1 to enable error injection on the DMU detected a parity error in the Mondo acknowledge credit from NCU.
8	dmuinternal	0	RW	Set to 1 to enable error injection on the DMU detected an internal error.
7	siidmuaparity	0	RW	Set to 1 to enable error injection on the SII detected a parity error on address field for DMA transactions from DMU FIFO.
6		0		
5	siidmudparity	0	RW	Set to 1 to enable error injection on the SII detected a parity error on data for DMA transactions from DMU FIFO.
4		0		
3	siidmuctagce	0	RW	Set to 1 to enable error injection on the SII detected a corrected error on a transaction from the DMU FIFO.
2		0		
1	siidmuctague	0	RW	Set to 1 to enable error injection on the SII detected an uncorrected error on a transaction from the DMU FIFO.
0		0		

25.23.8 SOC SII Error Syndrome Register

This register logs the SII Error Syndrome for the errors listed in the etag field. This register is not changed on a warm reset to allow inspection of the syndrome by software following the warm reset.

TABLE 25-58 shows the format of the SOC SII Error Syndrome register.

TABLE 25-58 SOC SII Error Syndrome Register – SOC_SII_ERROR_SYNDROME_REG ($80\ 0000\ 3030_{16}$)

Bit	Field	Initial Value	R/W	Description
63	v	0	RW	Valid bit, set to 1 when the syndrome has been logged.
62:59	—	0	RO	*Reserved*

TABLE 25-58 SOC SII Error Syndrome Register – SOC_SII_ERROR_SYNDROME_REG ($80\ 0000\ 3030_{16}$)

Bit	Field	Initial Value	R/W	Description
58:56	etag	0	RW	Error tag: 7 – SIIDMUAPARITY;5 – SIIDMUDPARITY;1– SIIDMUCTAGUE
55:40	ctag	0	RW	ctag field from the header.
39:0	pa	0	RW	Physical address.

25.23.9 SOC NCU Error Syndrome Register

This register logs the NCU Error Syndrome for the NCUCTAGUE, NCUDMUUE, NCUPCXUE, NCUPCXDATA, NCUINTTABLE, and NCUDATAPARITY errors. This register is not changed on a warm reset to allow inspection of the syndrome by software following the warm reset.

TABLE 25-59 shows the format of the SOC NCU Error Syndrome register.

TABLE 25-59 SOC NCU Error Syndrome Register – SOC_NCU_ERROR_SYNDROME_REG ($80\text{-}0000\text{-}3038_{16}$)

Bit	Field	Initial Value	R/W	Description
63	v	0	RW	Valid bit, set to 1 when the syndrome has been logged.
62	g	0	RW	Valid bit for CTAG field. If g = 0, none of the other field valid bits (r, c, s, or p) will be set and bits 15:0 indicate the Ctag. If v = 1, the other field valid bits can be nonzero and bits 39:0 indicate the PA.
61	r	0	RW	Valid bit for reqtype field.
60	c	0	RW	Valid bit for coreid field.
59	s	0	RW	Valid bit for strandid field.
58	p	0	RW	Valid bit for pa field.
57:56	—	0	RO	*Reserved*
55:51	etag	0	RW	Error tag: 42 – NCUDMUCredit; 41 – MCU3ECC; 40 – MCU3FBR; 38 – MCU2ECC; 37 – MCU2FBR; 35 – MCU1ECC; 34 – MCU1FBR; 32 – MCU0ECC; 31 – MCU0FBR; 26 – SIOCTAGCE; 25 – SIOCTAGUE; 24 – TESTMODE; 23 – NCUCTAGCE; 22 – NCUCTAGUE; 21 – NCUDMUUE; 20 – NCUCPXUE; 19 – NCUPCXUE; 18 – NCUPCXDATA; 17 – NCUINTTABLE 16 – NCUMONDOFIFO; 15 – NCUMONDOTABLE 14 – NCUDATAPARITY; 13 – DMUDATAPARITY; 12 – DMUSIICREDIT 11 – DMUCTAGUE; 10 – DMUCTAGCE; 9 – DMUNCUCREDIT 8 – DMUINTERNAL; 3 – SIIDMUCTAGCE
50:46	reqtype	0	RW	Request type

TABLE 25-59 SOC NCU Error Syndrome Register – SOC_NCU_ERROR_SYNDROME_REG ($80\text{-}0000\text{-}3038_{16}$)

Bit	Field	Initial Value	R/W	Description
45:43	coreid	0	RW	Physical strand ID.
42:40	strandid	0	RW	Strand ID on physical core.
39:0	pa_ctag	0	RW	Physical address{39:0} if p is set. If g bit is set, contains ctag in 15:0.

CHAPTER 26

Memory Controller

26.1 Overview

OpenSPARC T2 interfaces to external registered DDR2 fully buffered DIMMs (FBDs) through unidirectional high-speed links. OpenSPARC T2 II interfaces directly to external registered DDR2 DIMMs. There are four memory branches on OpenSPARC T2. Each memory branch services 64-byte read and write requests from two L2 cache banks of the on-chip L2 Cache unit.

The features of the OpenSPARC T2 memory controller are as follows:

- Uses 10-bit southbound and 14-bit northbound FBD channel protocols running at 12 times the SDRAM cycle rate.
- Supports 256-Mbit DRAM components for x4 data width; supports 512-Mbit, 1-Gbit, and 2-Gbit DRAM components for x4 and x8 data widths.
- Maximum memory of 128 Gbytes per branch using sixteen 8-Gbyte DDR2 FBDs
- Supports up to 16 ranks of DDR2 DIMMs per branch (8 pairs of double-sided FBDs)
- Supports registered DDR2 DIMMs of clock frequency up to 400 MHz
- Supports 128 bits of write data and 16 bits ECC per SDRAM cycle and 256 bits of read data and 32 bits ECC per SDRAM cycle.
- Supports DDR2 SDRAM burst length of 4 when using both FBD channels in a branch, burst length of 8 when using a single channel per branch.
- ECC generation, check, correction, and Extended ECC.
- Programmable DDR2 SDRAM power throttle control
- System peak memory bandwidth (4 branches): 50 Gbytes/s for reads, 25 Gbytes/s for writes.
-

Note | OpenSPARC T2 does not support the FBD Hot Plug feature.

26.2 Memory Terminology

A few of the more common memory and DRAM terms are described here.

bank Most DDR SDRAM chips are broken up into four or eight logical banks internally to enable full pipelining of memory operations.

channel Port connecting processor chip to DIMM.

DIMM Dual Inline Memory Module. Industry-standard SDRAM module package. A stick of memory.

DRAM chip Single chip inside the DIMM. We differentiate the type by how many bits it outputs and its capacity. (x4 means 4-bit output, x8 means 8-bit output, x16, x32 etc., and 256-Mbit or 512-Mbit capacity). Most common ones are the x4, x8 outputs.

rank A data group that can be accessed from a DIMM. Each DIMM has two chip selects. When a DIMM has two ranks, each chip select accesses DRAMs on one side of the DIMM independently. When a DIMM has one rank, both chip selects must be asserted at the same time to access all DRAMs on the DIMM. For x4 SDRAMs, single-rank DIMMs have 18 devices and double-rank DIMMs have 36 devices.

RAS/CAS RAS stands for "row address strobe." When this signal is asserted, a particular bank is enabled. It is also often referred to as "active" command. CAS stands for "column address strobe." When this signal is asserted, the column address and Read/Write signals are transmitted.

refresh DRAM requires what is often referred to as "refresh" cycle. Every row in the DRAM requires a refresh access every 15.6µS/7.8µS.

single-channel mode A low-power configuration with one DIMM per memory channel. Only one FBD channel is used, and the memory burst length is 8.

26.3 Fully Buffered DIMM (FBD) Terminology

A few of the more common FBD terms are described here.

advanced memory buffer (AMB) Buffers memory traffic between the host and the SDRAMs. Requests are sent by the host to the AMB across a high-speed link, and the AMB drives the requests to the SDRAMs using the DDR2 protocol.

bit lane A differential pair of signals in one direction.

cyclic redundancy code (CRC) An error detection code sent with data across the FBD link to protect the data from errors. When a CRC error is detected, the faulty frame must be retransmitted.

DDR branch A minimum aggregation of DDR channels that operate in lock-step to support error correction. A rank spans a branch. In OpenSPARC T2, a branch consists of one or two DDR channels.

DDR channel A channel that consists of a data channel with 72 bits of data and an addr/cntrl channel.

DDR data channel A data channel that consists of 72 bits of data divided into 18 data groups.

FBD Fully buffered DIMM.

frame Groups of bits containing commands or data sent across the link over 12 cycles.

Linear Feedback Shift register (LFSR) A shift register where the data input to the last register is a function of the outputs of other registers.

link High-speed parallel differential point-to-point interface.

northbound (NB) The direction of signals running from the farthest DIMM toward the host.

slot Socket for a DIMM.

southbound (SB) The direction of signals running from the host controller toward the DIMMs.

training sequence (TS) A sequence of bits sent per bit lane from the host to the FBDs to initialize the channel operation.

unit interval (UI) Average time interval between voltage transitions of a signal. Approximately 200 ps for DIMMs running at 800 MHz.

26.4 DRAM Branch Configuration

Each DRAM branch can have a different memory size and a different kind of DIMM (for example, a different number of ranks or different CAS latency). Software should not use address space larger than four times the lowest memory capacity in a channel because the cache lines are interleaved across channels.

The MCU employs the following design requirements:

- x4 and x8 DRAM parts are supported. Extended ECC is not supported for x8 DRAM parts.
- DIMM capacity, configuration, and timing parameters cannot be different within a memory branch.
- DRAM banks are always closed after read or write command by issuing an autoprecharge command.
- Burst length is 4 (bl = 4) when using a two channels per DDR branch. Burst length is 8 (bl = 8) when using a single channel per branch.
- There is a fixed 1 dead cycle for switching commands from one rank on a DIMM to the other rank on the same DIMM.
- Reads, writes, and refreshes across DDR branches have no relationship to each other. They are all independent.

There are four independent DDR branches per CPU chip, each controlled by a separate MCU. Each branch can be configured with one or two channels and supports up to 16 ranks of DIMMs as shown in FIGURE 26-1. Each channel can be populated with up to eight single- or dual-rank FBDs. When a branch is configured with two channels, the two FBDs that share the same AMB ID are accessed in lock-step. Data is returned 144 bits per frame for 8 frames in single channel mode and 288 bits per frame for 4 frames in dual channel mode. In either mode, the total data transfer size is 512 bits, or 64 bytes, the cache line size for the L2 cache.

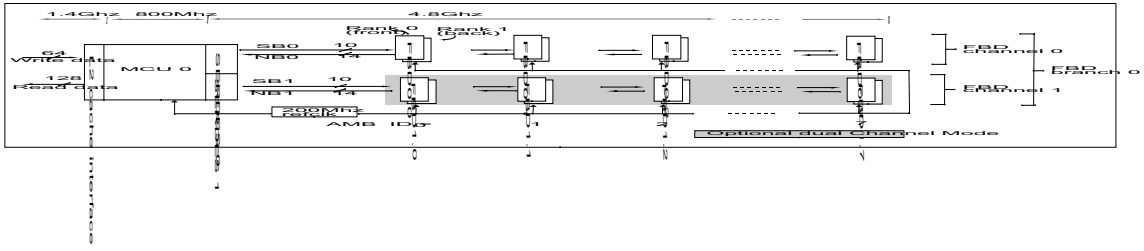


FIGURE 26-1 DDR Branch Configuration

Each FBD contains four or eight internal banks that can be controlled independently. These internal banks are controlled inside the SDRAM chips themselves. Accesses can overlap between different internal banks. In a normal configuration, every read and write operation to SDRAM will generate a burst length of 4 with 16 bytes of data transferred every half memory clock cycle. In single-channel mode, reads and writes will have a burst length of 8 with 8 bytes of data transferred every half memory cycle.

TABLE 26-1 shows the memory organizations supported by OpenSPARC T2. Table 26-2 shows how the MCU should be programmed for the supported DIMM configurations.

TABLE 26-1 OpenSPARC T2 Memory Configurations

DIMM	Base Device	Part	Ranks	# of Devices	Min. Memory per Branch	Max. Memory per Branch
512 MB	256 Mb	x4	1	18	512 MB	8 GB
1 GB	256 Mb	x4	2	36	1 GB	16 GB
1 GB	512 Mb	x4	1	18	1 GB	16 GB
2 GB	512 Mb	x4	2	36	2 GB	32 GB
2 GB	1 Gb	x4	1	18	2 GB	32 GB
4 GB	1 Gb	x4	2	36	4 GB	64 GB
4 GB	2 Gb	x4	1	18	4 GB	64 GB
8 GB	2 Gb	x4	2	36	8 GB	128 GB
512 MB	512 Mb	x8	1	9	512 MB	8 GB
1 GB	512 Mb	x8	2	18	1 GB	16 GB
1 GB	1 Gb	x8	1	9	1 GB	16 GB
2 GB	1 Gb	x8	2	18	2 GB	32 GB
2 GB	2 Gb	x8	1	9	2 GB	32 GB
4 GB	2 Gb	x8	2	18	4 GB	64 GB

TABLE 26-2 MCU programming for supported DIMMs

DIMM	Base Device	Ranks	8 bank mode	RAS Address Width	CAS Addr Width	Stacked
512 MB	256 Mb (64 Mb x4)	1	0	D_{16}	B_{16}	0
1 GB	256 Mb (64 Mb x4)	2	0	D_{16}	B_{16}	1
1 GB	512 Mb (128 Mb x4)	1	0	E_{16}	B_{16}	0
2 GB	512 Mb (128 Mb x4)	2	0	E_{16}	B_{16}	1
2 GB	1 Gb (256 Mb x4)	1	1	E_{16}	B_{16}	0
4 GB	1 Gb (256 Mb x4)	2	1	E_{16}	B_{16}	1
4 GB	2 Gb (512 Mb x4)	1	1	F_{16}	B_{16}	0
8 GB	2 Gb (512 Mb x4)	2	1	F_{16}	B_{16}	1
512 MB	512 Mb (64 Mb x8)	1	0	D_{16}	A_{16}	0

DIMM	Base Device	Ranks	8 bank mode	RAS Address Width	CAS Addr Width	Stacked
1 GB	512 Mb (64 Mb x8)	2	0	D_{16}	A_{16}	1
1 GB	1 Gb (128 Mb x8)	1	0	E_{16}	A_{16}	0
2 GB	1 Gb (128 Mb x8)	2	0	E_{16}	A_{16}	1
2 GB	2 Gb (256 Mb x8)	1	1	E_{16}	A_{16}	0
4 GB	2 Gb (256 Mb x8)	2	1	E_{16}	A_{16}	1

26.5 FBD Channel Configuration

The FBD specification supports two southbound channel configurations and five northbound channel configurations. OpenSPARC T2 will support both southbound configurations—the 10-bit and 10-bit failover modes—and two of the northbound configurations, the 14-bit and 14-bit failover modes. These modes support data packets of 64-bit data and 8-bit ECC. The 10-bit southbound mode provides 22 bits of CRC while the 10-bit failover mode has 10 bits of CRC. The 14-bit northbound mode provides 24 bits of CRC on read data (12 bits per 72-bit data packet), and the 14-bit failover mode provides 12 bits of CRC (6 bits per 72-bit data packet).

During channel initialization, software will determine if a channel can be fully utilized (10-bit southbound or 14-bit northbound mode) or if a failover mode must be used in which one of the bit lanes is muxed out.

26.5.1 FBD Channel Initialization

There are two ways to initialize the FBD channels. The first way uses a hardware state machine for initialization. This initialization triggered by writing to the Channel Reset register. In addition to a hardware state machine, the FBD channels can also be initialized through a software interface. This allows more flexibility in the initialization over the dedicated hardware state machine. Software must perform the following sequence of events to initialize an FBD channel:

1. Because the SerDes PLLs are enabled before the TXBCLKIN is stable, the transmit FIFO which tracks the phase drift between TXBCLKIN and the SerDes internal clock may be too far off center when the MCU begins transmitting data. In order to re-center this FIFO, software must toggle the TX_ENFTP bit in the SERDES_CONFIG_BUS_REG from 0 to 1 and back to 0.

2. Drive Electrical Idle on the SB channel's TX outputs by setting the Channel State register to disable. Channels must remain in `Disable` state for at least tDisable (51 frames) before transitioning to `Calibrate` state.

3. To transition to `Calibrate` state, set Channel State register to calibrate for longer than twice tClkTrain time (42 frames). Once the AMBs are in the `Calibrate` state, they must remain in this state for at least tCalibrate time (480K frames).

4. Drive Electrical Idle on SB channel to transition AMBs to `Disable` state. Remain in `Disable` state for at least tDisable time (51 frames).

5. Set the Channel State register to training to begin driving TS0 patterns on the SB channel to transition the AMBs to the `Training` state. The TS0 patterns are sent to the last AMB until TS0 patterns are received on the northbound channel with the AMB ID from the last AMB. Software will use the Training State Loopback registers to determine how many correct TS0 patterns have been received on the northbound channel. This training requires approximately 275 frames with eight DIMMs per channel. After several correct TS0 patterns have been received on 13 of 14 of the bit lanes, initialization can proceed to step 5.

6. Set the Channel State register to testing to begin driving TS1 patterns on the SB channel to transition the AMBs to the `Testing` state. The IBIST engine within the MCU will take over after the TS1 header has been sent, and it will signal the MCU upon its completion so the MCU can send the trailer and begin the next training sequence. After several TS1 patterns with the AMB ID of the last AMB have been received correctly, and software/IBIST has determined that at least 9 southbound and 13 northbound bit lanes are working, initialization can proceed to step 6.

7. Set the Channel State register to polling to begin driving TS2 patterns on the SB channel to transition the AMBs to the `Polling` state. Continue sending TS2 patterns to the last AMB until correct TS2 patterns are received on the NB channel. This determines the read round-trip delay for the channel. TS2 patterns can be sent to intermediate AMBs to determine which channel protocols they support and to check that they can properly merge their data into the NB data stream. AMBs that are not able to merge their data into the NB data stream correctly will assert their data_merge_error status bit. Once initialization reaches the L0 state, software can check these bits by using AMB Configuration register read commands to determine how to adjust the COMMAND_TO_DATA_INCR registers in the AMBs to increase the channel latency.

8. Set the Channel State register to config to begin driving TS3 patterns on the SB channel to transition the AMBs to the `Config` state. The TS3 patterns program the configuration of the SB and NB channels (always 10 SB and 14 SB for OpenSPARC T2) and which channel bits are muxed out if using a failover mode. TS3 patterns are issued until the patterns are correctly received on the NB channel.

9. Set the Channel State register to l0 to transition AMBs to `L0` state. After four consecutive NOPs have been sent on the SB channel, the channel is ready to accept channel and DRAM commands.

26.5.2 Interconnect BIST (IBIST)

Interconnect BIST provides a mechanism for system level testing of the FBD channel connections. The memory controller has IBIST transmit and receive engines, each with a pattern generater. The transmit engine sends patterns to an AMB on the southbound channel. The AMB loops the patterns back to the receive engine which checks the patterns for errors.

IBIST is optionally run during the TS1 stage of the FBD channel initialization sequence. The following sequence is used to run IBIST.

1. Set AMBID in FBD_CHANNEL_STATE_REG to the target AMB.
2. Set channel to be test in TS1_SB_NB_MAPPING_REG.
3. Set SBTS0CNT field in SBFIBINIT_REG and TRAINING_STATE_MIN_TIME_REG such that SBTS0CNT * 12 is greater than TRAINING_STATE_MIN_TIME_REG + 240.
4. Start RX engine by setting start bit in NBFIBPORTCTL_REG.
5. Start TX engine by setting start bit in SBFIBPORTCTL_REG.

After starting the TX engine, the memory controller will sequence the initialization state machine to the TS0 state and then to the TS1 state, buring which IBIST will be run. After IBIST completes, the memory controller will continue sequencing to the TS2 and TS3 states and complete in L0 state. Error status for the IBIST run will be logged in the NBFIBPORTCTL_REG.

26.6 AMB Initialization

1. 1.5V, 1.8V, and 3.3V power supplies come up:
   - RESET# asserted low while power supplies are coming up.
   - CKEs are low upon 1.8V power-up.
2. BIOS queries SPD on all the FBDs on the channel to determine operating conditions:
   - Channel frequency, compatible DIMMs, DRAM, and AMB parameters
3. Clocks up and stable at required frequency.
   - Reference clocks should be stable for at least 1 ms before RESET# deasserted.
   - DRAM clocks (CLK/CLK) may be toggling at this time.
4. RESET# deasserted high.

- CKEs to DRAMs remain low.

5. No in-band or SMBus transactions for at least 2 ms after RESET# deasserted.
6. AMB parameters critical for robust link initialization are programmed via SMBus.
   - Architected link registers:
     i. LINKPARNXT: link frequency. **Note:** some AMBs may use this write to trigger PLL init. After writing to LINKPARNXT, 200 µs is required prior to any in-band activity. Note: It is generally satisfied by additional SMBus activity.
     ii. FBDSBCFGNXT: SB transmitter drive strength, de-emphasis setting, and pass-through mode
     iii. FBDNBCFGNXT: NB transmitter drive strength, de-emphasis setting, and pass-through mode
   - Personality bytes from SPD needed for link inititalization
     i. PERSBYTE{5:0}NXT: Remaining Personality bytes are not required for link init and may be loaded over the high speed FBD configuration register accesses
   - These next register values must be transferred to the matching current registers before the FBD link leaves the `Disable` state
     i. Updates may be done right after the NXT register is updated when link is in electrical idle. Updates must be complete before the beginning of training.
7. FBD link is initialized including `Calibration` state.
   - Refer to *FBD Channel Configuration*" for initialization sequence.
8. Remaining AMB configuration is loaded over high speed FBD channel
   - CMD2DATA, remaining Personality bytes, other SPD parameters, DRAM parameters (MTR, DRT, DRC, etc.), Errors enabled, etc.
   - These AMB registers are loaded through the MCU Configuration register Access Address and Data registers
9. FBD Link goes through fast reset (no calibration) to establish the desired configuration.
   - DRAM clocks should be stable at this time (that is, after link train).
10. DRAM interface can now be established
    a. MRS/EMRS set up and DRAM initialization sequence using DCALCSR and DCALADDR.
       i. Refer to *Memory Initialization* for DRAM initialization sequence

ii. The DCALCSR and DCALADDR registers are accessed through the MCU Configuration register Access Address and Data registers. The address associated with the DRAM command is placed in the DCALADDR register. The command is programmed in the DCALCSR with bit 31 set to initiate the command. Software then polls this register until bit 31 is reset, indicating that the command has completed.

b. DRAM interface calibrated using DCALCSR.

c. Optionally, Membist functionality can be used to test the DRAMs.

d. DRAMs can be initialized with MemBist.

e. AMB autorefresh engine is enabled at this time.

11. Refresh must now be transferred to the host.
    - Option 1:
        i. Use fast reset on the link with DRAMs in self-refresh.
        ii. Clear DSREFTC.dissrexit to enable fast self-refresh exit when link is re-established
        iii. Put the link in disable state which automatically puts the DRAMs in self-refresh.
        iv. Start the refresh engine on the host.
        v. Bring up the link again.
        vi. Host starts sending refresh commands as soon as `L0` state is reached.
    - Option 2:
        i. Write control register to disable autorefresh engine followed by
        ii. Clear DAREFTC.arefen to turn off autorefresh.
        iii. Host then immediately takes over sending refresh commands.
12. Host now has complete control of the FBD Channel.

26.7 Memory Initialization

The power-up sequence for the SDRAMs is same as the JEDEC specification (JC 42.3). Software controls the sequence through the DDR Calibration Control and Status register and the DDR Calibration Address register within the AMB. These are accessed with AMB Configuration register reads and writes using the MCU's Configuration Register Access Address and Data registers.

26.7.1 Power On

Apply power, and maintain CKE below 0.2 * VDDQ and ODT at a low state. All other inputs may be undefined. Once RESETn is asserted to the AMB, it will drive CKE low.

26.7.2 Clocks Stable

Clocks to the DIMMs start as soon as power is enabled to the AMB. The clocks should be stable for at least 1 ms before RESETn is deasserted.

26.7.3 Assert CKE

Software has to write to the DRAM Controller Mode register within the AMB to enable CKEs to the DRAMs.

26.7.4 Software Configuration

Software needs to set all the DRAM configuration registers within the AMB to the desired value.

26.7.5 Pause for 200 µs

DDR SDRAM initialization requires a 200 µs wait after clock has been stable. The memory controller counts a fixed number of cycles to pause for this time.

26.7.6 Pause 400 ns

Controller waits for 400 ns before issuing precharge all command.

26.7.7 `precharge_all` Command

Controller issue a `precharge_all` command for all banks of the device and for all DIMMs present. This is done with a single command to all the DIMMs.

26.7.8 Issue EMRS(2) Write Command

Even though these registers contain no useful information, it is required that a write to this register be performed. The default data of 0 is written by controller.

26.7.9 Issue EMRS(3) Write Command

Even though these registers contain no useful information, it is required that a write to this register be performed. The default data of 0 is written by controller.

26.7.10 Issue EMRS(1) write command to enable DLL

Memory controller issues the extended mode register set command for DLL enable.

26.7.11 Reset DLL

Controller issues mode register set command for DLL reset.

26.7.12 `precharge_all` command

`precharge_all` command is issued for all banks of the devices and for all DIMMs present by the controller.

26.7.13 Two Auto Refresh Cycles

JEDEC specifies to issue two or more autorefresh commands, and the memory controller issues two autorefresh commands.

26.7.14 Set Mode Register to Configure the Device

Mode register command to initialize device operation is issued by the memory controller.

26.7.15 200 Cycles After DLL Reset, Set OCD Default Command

Controller counts the number of cycles from the DLL reset to this command and if it meets 200 cycle count, OCD default command is issued to the DIMMs.

26.7.16 Perform OCD Calibration

Software must read the ocd_sense_pull[up | down] CSRs within the AMBs and adjust each DRAM device accordingly. The encoded commands for the adjustments are written into the AMBs' write data FIFOs. When the OCD adjust command is issued, the data from the FIFOs is driven to the DRAMs.

The following pseudocode can be used to adjust the impedance of the SDRAM lines.

```
foreach (DRAM rank) {
  foreach (pullup, pulldown) {
    reset impedance strength;
    sense OCD;
    until (all DRAM devices set) {
       adjust each device accordingly;
       sense OCD;
    }
  }
}
```

26.7.17 Set OCD Exit Command

An OCD exit command is required to get out of the OCD calibration mode and is issued by the controller.

Note | To guarantee ODT off, vref must be valid and a low level must be applied to the ODT pin.

26.7.18 Initialization Complete

After the above step is performed, the DDR2 SDRAM initialization is finished, normal memory accesses are now allowed.

26.8 RAS Feature Overview

26.8.1 ECC and Extended ECC

The data sent to the DRAMs is protected by SEC-DED error correction. Galois field multiplication techniques are used to generate 16 bits of ECC for each 128 bits of data.

Extended ECC (failover) is a feature of the ECC bits, where they contain enough information to correct any nibble within a 128-bit word. If a single x4 SDRAM component fails, the SDRAM, which would normally hold parity information for double error detection, will be reused for normal data or single error correction information. Thus, when in failover mode, single-bit errors can still be detected and corrected, but multiple errors can no longer be detected. The DRAM Fail-Over Status register and the DRAM Fail-Over Mask register control the operation of the failover mode. Extended ECC is not supported in single channel mode.

Appendix A of the OpenSPARC T2 Programmer's Reference Manual discusses the ECC and Extended ECC algorithms for the OpenSPARC T2 MCU.

26.8.2 Memory Scrubbing

Memory scrubbing refers to the regeneration of ECC for data in memory and the correction of single-bit errors and detection of double-bit errors. When scrubbing is enabled through the DRAM Scrub Enable register described in *DRAM Scrub Enable Register* on page 380, at the end of the time interval defined by the DRAM Scrub Frequency register described in *DRAM Scrub Frequency Register* on page 378, a memory scrub request is issued to the DIMMs. The scrubbing requests have priority over L2 cache requests. First, a scrubbing read request is issued to the DIMMs, and L2 requests to the same bank as the scrub request are blocked. When the scrubbing read data returns, the error detection and correction logic is used on the data. ECC is regenerated and compared with the ECC data read from memory. If an error is detected, subsequent L2 cache transactions are halted and a single-bit or double-bit error is flagged in the DRAM Error Status register as well as being signaled to the L2 cache; then the MCU generates additional requests to the SDRAMs to collect more information on the error, as detailed in the following section. After the scrubbing transaction completes, the L2 cache requests are able to proceed.

Once a scrubbing request is sent, the time interval counter is reset and begins counting down again, and the scrub address is incremented to the next memory location.

Implementation Note There can only be one outstanding scrub command to the DIMMs. So, having a very low number in scrub frequency register does not issue a lot of scrubs.

26.8.3 ECC Error Handling

When an error occurs on a scrub read or an L2 cache read request, the MCU will flag the error to the L2 and log its error in the MCU ESR. Then it will try to determine if the error is a hard error or a transient error. When the error occurs, subsequent L2 requests will be blocked. The MCU will perform the first retry read and log its ECC status in the Retry Status register. If the first retry read does not have an uncorrectable error, the corrected read data is written back to the SDRAM, and a second retry read will be issued and its status will also be logged. Only the status from the original read will be sent to the L2 cache and logged in the ESR. One of the virtual processors must perform a register read to check the status of the subsequent reads. Full details on error handling can be found in *L2 Cache Error Descriptions* on page 261 and *L2 Error Registers* on page 282.

26.8.4 Data Poisoning

Data poisoning involves marking known corrupt data in memory with bad ECC so that any later access will get an ECC error. MCU memory poisoning is performed by flipping ECC check bits 15, 9, 5, and 0. This will generate a failing syndrome of 8221_{16} which, when encountered on a read, will most likely indicate poisoned data.

26.9 Access to Nonexistent Memory

Load accesses from nonexistent memory will take a *data_access_error* trap. Instruction fetches from nonexistent memory will take an *instruction_access_error* trap. Store accesses to nonexistent memory will be silently discarded by the system.

Please refer to TABLE 26-3 for out-of-bound address ranges for different memory configurations.

TABLE 26-3 Out-of-Bound Address Ranges for Different Memory Configurations

DIMMs per Channel	DIMM Capacity	Ranks	Dual Channel Memory	Dual Channel Out-of-Bound	Single Channel Memory	Single Channel Out-of-Bound
1	256 Mb × 4	1	1 GB	PA{39:32}	512 MB	PA{39:31}
2	256 Mb × 4	1	2 GB	PA{39:33}	1 GB	PA{39:32}
4	256 Mb × 4	1	4 GB	PA{39:34}	2 GB	PA{39:33}
8	256 Mb × 4	1	8 GB	PA{39:35}	4 GB	PA{39:34}
1	256 Mb × 4	2	2 GB	PA{39:33}	1 GB	PA{39:32}
2	256 Mb × 4	2	4 GB	PA{39:34}	2 GB	PA{39:33}
4	256 Mb × 4	2	8 GB	PA{39:35}	4 GB	PA{39:34}
8	256 Mb × 4	2	16 GB	PA{39:36}	8 GB	PA{39:35}
1	512 Mb × 4	1	2 GB	PA{39:33}	1 GB	PA{39:32}
2	512 Mb × 4	1	4 GB	PA{39:34}	2 GB	PA{39:33}
4	512 Mb × 4	1	8 GB	PA{39:35}	4 GB	PA{39:34}
8	512 Mb × 4	1	16 GB	PA{39:36}	8 GB	PA{39:35}
1	512 Mb × 4	2	4 GB	PA{39:34}	2 GB	PA{39:33}
2	512 Mb × 4	2	8 GB	PA{39:35}	4 GB	PA{39:34}
4	512 Mb × 4	2	16 GB	PA{39:36}	8 GB	PA{39:35}
8	512 Mb × 4	2	32 GB	PA{39:37}	16 GB	PA{39:36}
1	1 Gb × 4	1	4 GB	PA{39:34}	2 GB	PA{39:33}
2	1 Gb × 4	1	8 GB	PA{39:35}	4 GB	PA{39:34}
4	1 Gb × 4	1	16 GB	PA{39:36}	8 GB	PA{39:35}
8	1 Gb × 4	1	32 GB	PA{39:37}	16 GB	PA{39:36}
1	1 Gb × 4	2	8 GB	PA{39:35}	4 GB	PA{39:34}
2	1 Gb × 4	2	16 GB	PA{39:36}	8 GB	PA{39:35}
4	1 Gb × 4	2	32 GB	PA{39:37}	16 GB	PA{39:36}
8	1 Gb × 4	2	64 GB	PA{39:38}	32 GB	PA{39:37}
1	2 Gb × 4	1	8 GB	PA{39:35}	4 GB	PA{39:34}
2	2 Gb × 4	1	16 GB	PA{39:36}	8 GB	PA{39:35}
4	2 Gb × 4	1	32 GB	PA{39:37}	16 GB	PA{39:36}
8	2 Gb × 4	1	64 GB	PA{39:38}	32 GB	PA{39:37}
1	2 Gb × 4	2	16 GB	PA{39:36}	8 GB	PA{39:35}
2	2 Gb × 4	2	32 GB	PA{39:37}	16 GB	PA{39:36}
4	2 Gb × 4	2	64 GB	PA{39:38}	32 GB	PA{39:37}
8	2 Gb × 4	2	128 GB	PA{39:39}	64 GB	PA{39:38}

26.10 Power Management

The power used by the SDRAMs will be a significant portion of the system power usage. Some high-performance systems may be able to handle the maximum power consumption rates, but low-cost systems may need to limit their power usage due to cooling issues, etc. The power throttling scheme of OpenSPARC T2 limits the number of SDRAM memory access transactions during a specified time period. This is done by counting the number of banks that are opened (that is, activate cycles) during this time. Since all of the write and read transactions of OpenSPARC T2 use auto-precharge, the number of banks opened is equivalent to the number of write and read transactions. If the number of transactions during this time period exceeds a preprogrammed limit, no more memory transactions are dispatched until the time period expires. More details on memory power management can be found in *Memory Access Throttle Control* on page 411.

26.11 DRAM Control and Status Registers

This section describes the control registers and diagnostic access for the DRAM. Each DRAM branch has its own set of control, status, and error registers. Note that each DRAM branch requires that all DIMMs on that branch be of exactly the same kind. There is no requirement on the kind of DIMMs used by different DRAM branches. It is better if the size of physical memory on each branch is equal, but it would still work if the software programs the registers to the lowest value of the branches.

Note: When operating in Partial-Bank mode, the CSRs in the disabled memory controllers will not be accessible because the clocks to these controllers will be turned off to reduce power.

26.11.1 DRAM CAS Address Width Register

TABLE 26-4 shows the format of the DRAM CAS Address Width register.

TABLE 26-4 DRAM CAS Address Width Register – DRAM_CAS_ADDR_WIDTH_REG ($84\ 0000\ 0000_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	width	B_{16}	RW	Defines the number of bits of CAS address width. Any value other than A_{16} or B_{16} will be ignored. Reset only on POR.

26.11.2 DRAM RAS Address Width Register

This register indicates the number of RAS Address bits for a x4 DRAM part of a given density. When x8 parts are being used, this register should be programmed for the x4 part of the same density. The hardware will make the address width correction based on the DRAM CAS Address Width register.

TABLE 26-5 shows the format of the DRAM RAS Address Width register.

TABLE 26-5 DRAM RAS Address Width Register – DRAM_RAS_ADDR_WIDTH_REG ($84\ 0000\ 0008_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	width	F_{16}	RW	Defines the number of bits of RAS address width. Legal values are D_{16} to F_{16}. Values outside this range will be treated as F_{16}. Reset only on POR.

26.11.3 DRAM CAS Latency Register

TABLE 26-6 shows the format of the DRAM RAS Address Width register.

TABLE 26-6 DRAM CAS Address Latency Register – DRAM_CAS_LAT_REG ($84\ 0000\ 0010_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:3	—	0	RO	*Reserved*
2:0	lat	3_{16}	RW	Defines the CAS latency. 2_{16} = CL of 2; 3_{16} = CL of 3; and so on. Reset only on POR.

26.11.4 DRAM Scrub Frequency Register

TABLE 26-7 shows the format of the DRAM Scrub Frequency register.

TABLE 26-7 DRAM Scrub Frequency Register – DRAM_SCRUB_FREQ_REG ($84\ 0000\ 0018_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:12	—	0	RO	*Reserved*
11:0	freq	FFF_{16}	RW	Defines how often scrubbing should be performed in terms of DRAM clocks. Reset only on POR. A smaller number causes more frequent scrubbing.

26.11.5 DRAM Refresh Frequency Register

TABLE 26-8 shows the format of the DRAM Refresh Frequency register.

TABLE 26-8 DRAM Refresh Frequency Register – DRAM_REFRESH_FREQ_REG ($84\ 0000\ 0020_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:13	—	0	RO	*Reserved*
12:0	freq	820_{16} (for 266 MHz DIMMs)	RW	Defines how often refresh should be performed in terms of DRAM clocks. It is 2080 (820_{16}) cycles at 266 MHz clock, 2600 ($A28_{16}$) for 333 MHz, and 3120 ($C30_{16}$) for 400 MHz. These values are approximately 7.8us. When this register value is written, the refresh counter value is reset. A smaller value generates more frequent refresh requests. This register is only reset on POR.

26.11.6 DRAM Refresh Counter Register

This register is the free-running counter that triggers refresh when it equals DRAM_REFRESH_FREQ_REG. Refreshes are not issued to the DRAMs if the DRAM_DIMM_INIT_REG bit 0 is set to 1.

TABLE 26-9 shows the format of the DRAM Refresh Counter register.

TABLE 26-9 DRAM Refresh Counter Register – DRAM_REFRESH_COUNTER_REG ($84\ 0000\ 0038_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:13	—	0	RO	*Reserved*
12:0	count	0	RW	Free running counter to issue refresh command when it equals Refresh frequency register. This register is reset to 0 when the DRAM Refresh Frequency register is written or when the count equals or exceeds the DRAM Refresh Frequency value. Reset on POR, WMR, or DBR.

26.11.7 DRAM Scrub Enable Register

This register controls whether scrub should happen in background to DRAM.

TABLE 26-10 shows the format of the DRAM Scrub Enable register.

TABLE 26-10 DRAM Scrub Enable Register – DRAM_SCRUB_ENABLE_REG (84 0000 0040_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	enab	0	RW	If 1, scrub is enabled. Reset only on POR.

26.11.8 DRAM RAS to RAS Different Bank Delay Register

TABLE 26-11 shows the format of the DRAM RAS to RAS Different Bank Delay register.

TABLE 26-11 DRAM RAS to RAS Different Bank Delay Register – DRAM_TRRD_REG (84 0000 0080_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	2_{16}	RW	Trrd delay. Reset only on POR.

26.11.9 DRAM RAS to RAS Same Bank Delay Register

TABLE 26-12 shows the format of the DRAM RAS to RAS Same Bank Delay register.

TABLE 26-12 DRAM RAS to RAS Sam Bank Delay Register – DRAM_TRC_REG (84 0000 $0088)_{16}$ (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:5	—	0	RO	*Reserved*
4:0	delay	C_{16}	RW	Trc delay. This is by default equals to tRAS + tRP. Reset only on POR.

26.11.10 DRAM RAS to CAS Delay Register

TABLE 26-13 shows the format of the DRAM RAS to CAS Delay register.

TABLE 26-13 DRAM RAS to CAS Delay Register – DRAM_TRCD_REG (84 0000 0090_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	3_{16}	RW	Trcd delay. Reset only on POR.

26.11.11 DRAM Write to Read CAS Delay Register

TABLE 26-14 shows the format of the DRAM Write to Read CAS Delay register.

TABLE 26-14 DRAM Write to Read CAS Delay Register – DRAM_TWTR_REG (84 0000 0098_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	0_{16}	RW	Twtr delay. Actual delay is this register value + CL – 1 + BL/2 + iWTR. Reset only on POR. DRAM controller supports a total of these values not to exceed F_{16}.

26.11.12 DRAM Read to Write CAS Delay Register

TABLE 26-15 shows the format of the DRAM Read to Write CAS Delay register.

TABLE 26-15 DRAM Read to Write CAS Delay Register – DRAM_TRTW_REG (84 0000 $00A0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	0_{16}	RW	Trtw delay. Actual delay is this register value + BL/2 + 2 tCK. Reset only on POR. DRAM controller supports a total of these values not to exceed $F._{16}$

26.11.13 DRAM Internal Read to Precharge Delay Register

TABLE 26-16 shows the format of the DRAM internal Read to Precharge Delay register.

TABLE 26-16 DRAM internal Read to Precharge Delay Register – DRAM_TRTP_REG (84 0000 00A8$_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:3	—	0	RO	*Reserved*
2:0	delay	2_{16}	RW	Trtp delay. The minimum value for this register is 2_{16}. If software tries to program a value less than 2_{16}, the register will be loaded with the value 2_{16}. Reset only on POR.

26.11.14 DRAM Active to Precharge Delay Register

TABLE 26-17 shows the format of the DRAM Active to Precharge Delay register.

TABLE 26-17 DRAM Active to Precharge Delay Register – DRAM_TRAS_REG (84 0000 00B0$_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	9_{16}	RW	Tras delay. Reset only on POR.

26.11.15 DRAM Precharge Command Period Register

TABLE 26-18 shows the format of the DRAM Precharge Command Period register.

TABLE 26-18 DRAM Precharge Command Period Register – DRAM_TRP_REG (84 0000 00B8$_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	3_{16}	RW	Trp delay. Reset only on POR.

26.11.16 DRAM Write Recovery Period Register

TABLE 26-19 shows the format of the DRAM Write Recovery Period register.

TABLE 26-19 DRAM Write Recovery Period Register – DRAM_TWR_REG (84 0000 $00C0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3:0	delay	3_{16}	RW	Twr delay. Reset only on POR.

26.11.17 DRAM Autorefresh to Active Period Register

TABLE 26-20 shows the format of the DRAM Autorefresh to Active Period register.

TABLE 26-20 DRAM Autorefresh to Active Period Register – DRAM_TRFC_REG (84 0000 $00C8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:7	—	0	RO	*Reserved*
6:0	delay	27_{16}	RW	Trfc delay (195 ns @ 200MHz). Reset only on POR.

26.11.18 DRAM Mode Register Set Command Period Register

TABLE 26-21 shows the format of the DRAM Mode Register Set Command Period register.

TABLE 26-21 DRAM Mode Register Set Command Period Register – DRAM_TMRD_REG (84 0000 $00D0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0	RO	*Reserved*
1:0	delay	2_{16}	RW	Tmrd delay. Reset only on POR.

26.11.19 DRAM Four-Activate Window Register

TABLE 26-22 shows the format of the DRAM Four-Activate Window Register.

TABLE 26-22 DRAM Four-Activate Window Register – DRAM_FAWIN_REG (84 0000 $00D8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:5	—	0	RO	*Reserved*
4:0	mode	A_{16}	RW	Tfaw. Number of cycles in which four activate commands may be sent to a DIMM. Reset only on POR.

26.11.20 DRAM Internal Write to Read Command Delay Register

TABLE 26-23 shows the format of the DRAM Internal Write to Read Command Delay register.

TABLE 26-23 DRAM Internal Write to Read Command Delay Register – DRAM_TIWTR_REG (84 0000 $00E0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0	RO	*Reserved*
1:0	delay	2_{16}	RW	Tiwtr delay. Reset only on POR.

26.11.21 DRAM Precharge Wait Register During Power Up

This register is no longer used by hardware.

This register controls the wait time before a precharge can be issued during the power up sequence.

TABLE 26-24 shows the format of the DRAM Precharge Wait Register During Power Up.

TABLE 26-24 DRAM Precharge Wait Register – DRAM_PRECHARGE_WAIT_REG (84 0000 $00E8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	bunch	55_{16}	RW	Equivalent to 80 dram cycles, which is 400 ns @ 200 MHz.

26.11.22 DRAM DIMM Stacked Register

TABLE 26-25 shows the format of the DRAM DIMM Stacked register.

TABLE 26-25 DRAM DIMM Stacked Register – DRAM_DIMM_STACK_REG (84 0000 0108_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	stack	0_{16}	RW	Set to 1 if DIMMs are stacked, 0 otherwise. Reset only on POR.

26.11.23 DRAM Extended Mode (2) Register

TABLE 26-26 shows the format of the DRAM Extended Mode (2) register.

TABLE 26-26 DRAM Extended Mode (2) Register – DRAM_EXT_WR_MODE2_REG (84 0000 0110_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:15	—	0	RO	*Reserved*
14:0	mreg	0_{16}	RW	Default value of Extended mode register (2) defined by JEDEC. Reset only on POR.

26.11.24 DRAM Extended Mode (1) Register

TABLE 26-27 shows the format of the DRAM Extended Mode (1) register.

TABLE 26-27 DRAM Extended Mode (1) Register – DRAM_EXT_WR_MODE1_REG (84 0000 0118_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:15	—	0	RO	*Reserved*
14:0	mreg	18_{16}	RW	Extended mode register (1) defined by JEDEC. Reset only on POR.

26.11.25 DRAM Extended Mode (3) Register

TABLE 26-28 shows the format of the DRAM Extended Mode (3) register.

TABLE 26-28 DRAM Extended Mode (3) Register – DRAM_EXT_WR_MODE3_REG ($84\ 0000\ 0120_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:15	—	0	RO	*Reserved*
14:0	mreg	0_{16}	RW	Extended mode register (3) defined by JEDEC. Reset only on POR.

26.11.26 DRAM 8 Bank Mode Register

This register indicates whether a x4 DRAM part of a given density has 4 or 8 internal banks. When x8 parts are being used, this register should be programmed for the x4 part of the same density. The hardware will make the 8 bank mode correction based on the DRAM CAS Address Width register.

Note: When the CAS Address Width register is programmed to A_{16}, the 8 bank mode register will always be read as a 1; however, the hardware will use the value that was written to this register.

TABLE 26-29 shows the format of the DRAM 8 Bank Mode register.

TABLE 26-29 DRAM 8 Bank Mode Register – DRAM_8_BANK_MODE_REG ($84\ 0000\ 0128_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	disable	1_{16}	RW	If 1, the dram devices have 8 banks instead of 4 banks. Reset only on POR.

26.11.27 DRAM Branch Disabled Register

TABLE 26-30 shows the format of the DRAM Branch Disabled register.

TABLE 26-30 DRAM Branch Disabled Register – DRAM_BRANCH_DISABLED_REG ($84\ 0000\ 0138_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	disable	0_{16}	RW	Set to 1 if branch is not present or is disabled. Reset only on POR.

26.11.28 DRAM Select Low Order Address Bits Register

TABLE 26-31 shows the format of the DRAM Select Low Order Address Bits register.

TABLE 26-31 DRAM Select Low Order Address Bits Register – DRAM_SEL_LO_ADDR_BITS_REG ($84\ 0000\ 0140_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	bunch	0	RW	If 1, then bits after the bank are used for stack and rank instead of the top most bits of PA. Reset only on POR.

26.11.29 DRAM Single Channel Mode Register

TABLE 26-32 shows the format of the DRAM Single Channel Mode register.

TABLE 26-32 DRAM Single Channel Mode register – DRAM_SNGL_CHNL_MODE_REG ($84\ 0000\ 0148_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	mode	0	RW	If 1, enables single channel mode for FBD. Burst length becomes 8. Reset only on POR.

26.11.30 DRAM DIMM Initialization Register

TABLE 26-33 shows the format of the DRAM DIMM Initialization register.

TABLE 26-33 DRAM DIMM Initialization Register – DRAM_DIMM_INIT_REG ($84\ 0000\ 01A0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	X	RO	*Reserved*
1	cke	0	RW	CKE enable to the controller. Set to 1 to assert CKE high to the DIMMS.
0	init	1_{16}	RW	Software must clear this register upon completing the initialization of the MCU and AMB registers and the DRAM initialization sequence.

26.11.31 DRAM Mode Reg Write Status Register

This register is no longer used by hardware.

TABLE 26-34 shows the format of the DRAM Mode Reg Write Status register.

TABLE 26-34 DRAM Mode Reg Write Status Register – DRAM_MODE_WRITE_STATUS_REG ($84\ 0000\ 0208_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	done	0_{16}	RO	1 if mode reg write done, 0 otherwise. Also its reset to 0 if register $1A0_{16}$ is written to. Reset only on POR.

26.11.32 DRAM Initialization Status Register

TABLE 26-35 shows the format of the DRAM Initialization Status register.

TABLE 26-35 DRAM HW Initialization Status Register – DRAM_INIT_STATUS_REG ($84\ 0000\ 0210_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	done	0_{16}	RO	1 if DRAM initialization sequence is done, 0 otherwise. Also its reset to 0 if register $1A0_{16}$ is written to. Reset only on POR.

26.11.33 DRAM DIMMs Present Register

TABLE 26-36 shows the format of the DRAM DIMMs Present register.

TABLE 26-36 DRAM DIMMs Present Register – DRAM_DIMMS_PRESENT_REG ($84\ 0000\ 0218_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	X	RO	*Reserved*
3:0	num	1_{16}	RW	Number of DIMMs per channel. Reset only on POR.

26.11.34 DRAM Failover Status Register

Each DRAM branch has a failover status register which can be set by software to move a nibble of data from a bad chip to replace one of the ECC chips.

TABLE 26-37 shows the format of the DRAM Fail-Over Status register.

TABLE 26-37 DRAM Fail-Over Status Register – DRAM_FAILOVER_STATUS_REG ($84\ 0000\ 0220_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	en	0_{16}	RW	Set if in fail-over mode. Reset only on POR.

26.11.35 DRAM Failover Mask Register

Each DRAM branch has a failover status mask register which can be set by software to specify which nibble to ignore the data from.

TABLE 26-38 shows the format of the DRAM Fail-Over Mask register.

TABLE 26-38 DRAM Fail-Over Mask Register – DRAM_FAILOVER_MASK_REG ($84\ 0000\ 0228_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:35	—	0	RO	*Reserved*
34:0	mask	0_{16}	RW	Mask register to shift data. Reset only on POR.

26.11.36 Power Down Mode Register

TABLE 26-39 shows the format of the DRAM Power Down Mode register.

TABLE 26-39 Power Down Mode Register ($84\ 0000\ 0238_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	enb_hp	0_{16}	RW	Enables the use of DRAM power down mode for power saving.

Note The Power Down Mode register should not be changed during normal system operation. Doing so may cause transactions to be dropped or memory contents to be updated incorrectly.

26.11.37 FBD Channel State Register

TABLE 26-40 shows the format of the FBD Channel State register.

TABLE 26-40 FBD Channel State Register – FBD_CHANNEL_STATE_REG ($84\ 0000\ 0800_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	0_{16}	RO	*Reserved*
7	mdisable	0_{16}	RW	Disable AMB data merging for TS2 patterns.
6:3	ambid	0_{16}	RW	Target AMB for training sequences. Reset only on POR.
2:0	state	0_{16}	RW	State in initialization sequence: 0_{16} = Disable; 1_{16} = Calibrate; 2_{16} = Training; 3_{16} = Testing; 4_{16} = Polling; 5_{16} = Config, 6_{16} = L0.

26.11.38 FBD Fast Reset Flag Register

TABLE 26-41 shows the format of the FBD Fast Reset Flag register.

TABLE 26-41 FBD Fast Reset Flag Register – FBD_FAST_RESET_FLAG_REG ($84\ 0000\ 0808_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0_{16}	RO	*Reserved*
3	sync_ier	0_{16}	RW	Enable use of ier bit in Sync command. ier will be issued in last sync frame before a channel reset.
2:1	sync_r	0_{16}	RW	Indicates which status register will be received from the AMBs.
0	fastreset	0_{16}	RW	Causes MCU to use the Fast Reset sequence for FBDIMM channel initialization. Reset only on POR.

26.11.39 FBD Channel Reset Register

Writing to bit 0 of this register causes the MCU to sequence through the AMB initialization states. If the FBD Fast Reset Flag is set, the Calibration stage will not be included in the initialization.

TABLE 26-42 shows the format of the FBD Channel Reset register.

TABLE 26-42 FBD Channel Reset Register – FBD_CHNL_RESET_REG ($84\ 0000\ 0810_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1	err	0_{16}	RW	Set to 1 if an error occurred during FBD channel initialization.
0	fbdinit	0_{16}	RW	Causes MCU to initialize FBD channels. Reset to 0 when initialization is complete.

26.11.40 TS1 Southbound to Northbound Mapping Register

TABLE 26-43 shows the format of the TS1 Southbound to Northbound Mapping register.

TABLE 26-43 TS1 Southbound to Northbound Mapping Register – TS1_SB_NB_MAPPING_REG ($84\ 0000\ 0818_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:4	—	0_{16}	RO	*Reserved*
3	ibrx_chnl	0_{16}	RW	Indicates which channel for IBIST to check. Reset only on POR.
2:0	mapping	0_{16}	RW	Determines how targeted AMB maps data from SB bit lanes to NB bit lanes. Reset only on POR

26.11.41 TS1 Test Parameter Register

TABLE 26-44 shows the format of the TS1 Test Parameter register.

TABLE 26-44 TS1 Test Parameter Register – TS1_TEST_PARAMETER_REG ($84\ 0000\ 0820_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:24	—	0_{16}	RO	*Reserved*
23:0	param	0_{16}	RW	AMB test parameters for TS1 sequence. Reset only on POR.

26.11.42 TS3 Failover Configuration Register

TABLE 26-45 shows the format of the TS3 Failover Configuration register.

TABLE 26-45 TS3 Failover Configuration Register – TS3_FAILOVER_CONFIG_REG ($84\ 0000\ 0828_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15:12	sbconfig1	F_{16}	RW	Indicates which southbound lanes for channel 1 will be used. Reset only on POR.
11:8	nbconfig1	F_{16}	RW	Indicates which northbound lanes for channel 1 will be used. Reset only on POR.
7:4	sbconfig0	F_{16}	RW	Indicates which southbound lanes for channel 0 will be used. Reset only on POR.
3:0	nbconfig0	F_{16}	RW	Indicates which northbound lanes for channel 0 will be used. Reset only on POR.

26.11.43 Electrical Idle Detected Register

TABLE 26-46 shows the format of the Electrical Idle Detected register.

TABLE 26-46 Electrical Idle Detected Register – ELECTRICAL_IDLE_DETECTED_REG ($84\ 0000\ 0830_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:28	—	0_{16}	RO	*Reserved*
27:14	electidle1	$3FF_{16}$	RO	Electrical Idle detected from bit lanes in channel 1
13:0	electidle0	$3FF_{16}$	RO	Electrical Idle detected from bit lanes in channel 0

26.11.44 Disable State Period Register

TABLE 26-47 shows the format of the Disable State Period register.

TABLE 26-47 Disable State Period Register – DISABLE_STATE_PERIOD_REG ($84\ 0000\ 0838_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:10	—	0_{16}	RO	*Reserved*
9:0	count	FF_{16}	RW	Counter value for the Disable state. Once Disable state is entered, a counter will count to this value. Once the count is reached, the Disable Done bit will be set. Reset only on POR.

26.11.45 Disable State Period Done Register

TABLE 26-48 shows the format of the Disable State Period Done register.

TABLE 26-48 Disable State Period Done Register – DISABLE_STATE_PERIOD_DONE_REG ($84\ 0000\ 0840_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	0_{16}	RO	*Reserved*
0	done	0_{16}	RW	If set, indicates that the counter for the Disable state period has completed counting.

26.11.46 Calibrate State Period Register

TABLE 26-49 shows the format of the Calibrate State Period register.

TABLE 26-49 Calibrate State Period Register – CALIBRATE_STATE_PERIOD_REG $(84\ 0000\ 0848)_{16}$ (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:20	—	0_{16}	RO	*Reserved*
19:0	count	0_{16}	RW	Counter value for the Calibrate state. Once Calibrate state is entered, a counter will count to this value. Once the count is reached, the calibrate_done bit will be set. Reset only on POR.

26.11.47 Calibrate State Period Done Register

TABLE 26-50 shows the format of the Calibrate State Period Done register.

TABLE 26-50 Calibrate State Period Done Register – CALIBRATE_STATE_PERIOD_DONE_REG $(84\ 0000\ 0850_{16})$ (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:1	—	0_{16}	RO	*Reserved*
0	done	0_{16}	RW	If set, indicates that the counter for the Calibrate state period has completed counting.

26.11.48 Training State Minimum Time Register

TABLE 26-51 shows the format of the Training State Minimum Time register.

TABLE 26-51 Training State Minimum Time Register – TRAINING_STATE_MIN_TIME_REG $(84\ 0000\ 0858_{16})$ (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15:0	count	FF_{16}	RW	Minimum number of frames for Training state before starting to check for Done. Reset only on POR.

26.11.49 Training State Done Register

TABLE 26-52 shows the format of the Training State Done register.

TABLE 26-52 Training State Done Register – TRAINING_STATE_DONE_REG ($84\ 0000\ 0860_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1	timeout	0_{16}	RW	Set if MCU does not recognize 16 `Training` state sequences from FBDs before `Training` state timeout period has elapsed.
0	done	0_{16}	RW	Set when MCU has recognized 16 training state sequences from FBDs after minimum `Training` state period has elapsed.

26.11.50 Training State Timeout Register

TABLE 26-53 shows the format of the Recalibration Duration register.

TABLE 26-53 Training State Timeout Register – TRAINING_STATE_TIMEOUT_REG ($84\ 0000\ 0868_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15:0	count	$FFFF_{16}$	RW	Timeout period for `Training` state. Reset only on POR.

26.11.51 Testing State Done Register

TABLE 26-54 shows the format of the Testing State Done register.

TABLE 26-54 Testing State Done Register – TESTING_STATE_DONE_REG ($84\ 0000\ 0870_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1	timeout	0_{16}	RW	Set if MCU does not recognize 16 `Testing` state sequences from FBDs before testing state timeout period has elapsed.
0	done	0_{16}	RW	Set when MCU has recognized 16 `Testing` state sequences from FBDs.

26.11.52 Testing State Timeout Register

TABLE 26-55 shows the format of the Testing State Timeout register.

TABLE 26-55 Testing State Timeout Register – TESTING_STATE_TIMEOUT_REG (84 0000 0878_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	0_{16}	RO	*Reserved*
7:0	count	FF_{16}	RW	Timeout period for `Testing` state. Reset only on POR.

26.11.53 Polling State Done Register

TABLE 26-56 shows the format of the Polling State Done register.

TABLE 26-56 Polling State Done Register – POLLING_STATE_DONE_REG (84 0000 0880_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1	timeout	0_{16}	RW	Set if MCU does not recognize 16 `Polling` state sequences from FBDs before Polling State Timeout period has elapsed.
0	done	0_{16}	RW	Set when MCU has recognized 16 `Polling` state sequences from FBDs.

26.11.54 Polling State Timeout Register

TABLE 26-57 shows the format of the Polling State Timeout register.

TABLE 26-57 Polling State Timeout Register – POLLING_STATE_TIMEOUT_REG (84 0000 0888_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	0_{16}	RO	*Reserved*
7:0	count	FF_{16}	RW	Timeout period for `Polling` state. Reset only on POR.

26.11.55 Config State Done Register

TABLE 26-58 shows the format of the Config State Done register.

TABLE 26-58 Config State Done Register – CONFIG_STATE_DONE_REG ($84\ 0000\ 0890_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1	timeout	0_{16}	RW	Set if MCU does not recognize 16 `Config` state sequences from FBDs before Config State Timeout period has elapsed.
0	done	0_{16}	RW	Set when MCU has recognized 16 `Config` state sequences from FBDs.

26.11.56 Config State Timeout Register

TABLE 26-59 shows the format of the Config State Timeout register.

TABLE 26-59 Config State Timeout Register – CONFIG_STATE_TIMEOUT_REG ($84\ 0000\ 0898_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:8	—	0	RO	*Reserved*
7:0	count	FF_{16}	RW	Timeout period for `Config` state. Reset only on POR.

26.11.57 DRAM Per-Rank CKE Register

Writing this register or the `cke` bit of the DIMM Initialization register, $84\ 0000\ 01A0_{16}$, will cause the MCU to send a CKE command to the FBDIMMs. Each bit corresponds to a rank in a fully populated FBDIMM branch. The enable bits for the CKE command are qualified by the number of DIMMS in the branch, whether the DIMMs are stacked and whether the DIMM Initialization register `cke` bit is set.

Memory traffic must be disabled by setting bit 0 of the DIMM Initialization register when using the Per-Rank CKE register because timing checks are not performed between CKE commands and DRAM transactions.

TABLE 26-60 shows the format of the DRAM Per-Rank CKE register.

TABLE 26-60 PER_RANK_CKE_REG (84 0000 08A0$_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15	d7r1	1_{16}	RW	CKE enable for DIMM 7, Rank 1. Reset only on POR.
14	d7r0	1_{16}	RW	CKE enable for DIMM 7, Rank 0. Reset only on POR.
13	d6r1	1_{16}	RW	CKE enable for DIMM 6, Rank 1. Reset only on POR.
12	d6r0	1_{16}	RW	CKE enable for DIMM 6, Rank 0. Reset only on POR.
11	d5r1	1_{16}	RW	CKE enable for DIMM 5, Rank 1. Reset only on POR.
10	d5r0	1_{16}	RW	CKE enable for DIMM 5, Rank 0. Reset only on POR.
9	d4r1	1_{16}	RW	CKE enable for DIMM 4, Rank 1. Reset only on POR.
8	d4r0	1_{16}	RW	CKE enable for DIMM 4, Rank 0. Reset only on POR.
7	d3r1	1_{16}	RW	CKE enable for DIMM 3, Rank 1. Reset only on POR.
6	d3r0	1_{16}	RW	CKE enable for DIMM 3, Rank 0. Reset only on POR.
5	d2r1	1_{16}	RW	CKE enable for DIMM 2, Rank 1. Reset only on POR.
4	d2r0	1_{16}	RW	CKE enable for DIMM 2, Rank 0. Reset only on POR.
3	d1r1	1_{16}	RW	CKE enable for DIMM 1, Rank 1. Reset only on POR.
2	d1r0	1_{16}	RW	CKE enable for DIMM 1, Rank 0. Reset only on POR.
1	d0r1	1_{16}	RW	CKE enable for DIMM 0, Rank 1. Reset only on POR.
0	d0r0	1_{16}	RW	CKE enable for DIMM 0, Rank 0. Reset only on POR.

26.11.58 L0s Duration Register

Note The *enb_hp* bit of this register should always be set to 0. N2 does not fully support the AMB L0s state, and enabling this feature will cause unrecoverable channel errors.

TABLE 26-61 shows the format of the L0s Duration register.

TABLE 26-61 L0s Duration Register – L0S_DURATION_REG (84 0000 08A8$_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:7	—	0_{16}	RO	*Reserved*
6	enb_hp	0_{16}	RW	Enables use of L0s state.
5:0	count	$2A_{16}$	RW	Determines the number of frames that the branch will be in L0s state. Legal values are 20_{16} to $2A_{16}$. Values below 20_{16} will be treated as 20_{16}, and values above $2A_{16}$ will be treated as $2A_{16}$. Reset only on POR.

26.11.59 Channel Sync Frame Frequency Register

TABLE 26-62 shows the format of the Channel Sync Frame Frequency register.

TABLE 26-62 Channel Sync Frame Frequency Register – CHANNEL_SYNC_FRAME_FREQ_REG ($84\ 0000\ 08B0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:6	—	0_{16}	RO	*Reserved*
5:0	freq	$2A_{16}$	RW	Frequency at which sync frames are sent on channels. Reset only on POR.

26.11.60 Channel Read Latency Register

TABLE 26-63 shows the format of the Channel Read Latency register.

TABLE 26-63 Channel Read Latency Register – CHANNEL_READ_LAT_REG ($84\ 0000\ 08B8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15:8	latency1	FF_{16}	RW	Read latency for channel 1. Determined during `Polling` state.
7:0	latency0	FF_{16}	RW	Read latency for channel 0. Determined during `Polling` state.

26.11.61 Channel Capability Register

TABLE 26-64 shows the format of the Channel Capability register.

TABLE 26-64 Channel Capability Register – CHANNEL_CAPABILITY_REG ($84\ 0000\ 08C0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:10	—	0_{16}	RO	*Reserved*
9:5	capabil1	0_{16}	RO	Channel capabilities for selected AMB in channel 1. Determined during `Polling` state.
4:0	capabil0	0_{16}	RO	Channel capabilities for selected AMB in channel 0. Determined during `Polling` state.

26.11.62 Loopback Mode Control Register

TABLE 26-65 shows the format of the Loopback Mode Control register.

TABLE 26-65 Loopback Mode Control Register – LOOPBACK_MODE_CNTL_REG (84 0000 $08C8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:2	—	0_{16}	RO	*Reserved*
1:0	mode	0_{16}	RW	Branch Loopback Mode. 0 = Loopback Mode disabled; 10 = Place low-order NB data on SB bus; 11 = Place high-order NB data on SB bus

26.11.63 SerDes Configuration Bus Register

TABLE 26-66 shows the format of the SerDes Configuration Bus register.

TABLE 26-66 SerDes Configuration Bus Register – SERDES_CONFIG_BUS_REG (84 0000 $08D0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:30	—	0_{16}	RO	*Reserved*
29:28	rxtx_rate	0_{16}	RW	Receiver/transmitter operating rate. Reset only on POR.
27	tx_cm	0_{16}	RW	Transmitter common mode. Reset only on POR.
26:24	tx_swing	1_{16}	RW	Transmitter output swing. Reset only on POR.
23:20	tx_de	0_{16}	RW	Transmitter de-emphasis. Reset only on POR.
19	tx_enftp	0_{16}	RW	Transmitter enable fixed TXBCLKIN phase. Reset only on POR.
18:16	rx_term	0_{16}	RW	Receiver termination. Reset only on POR.
15	—	0_{16}	RO	*Reserved*
14:12	rx_cdr	0_{16}	RW	Receiver Clock/Data Recovery algorithm. Reset only on POR.
11:8	rx_eq	0_{16}	RW	Receiver adaptive equalizer configuration. Reset only on POR.
7:6	—	0_{16}	RO	*Reserved*
5:2	pll_mpy	6_{16}	RW	PLL multiplier. reset only on POR.
1:0	pll_lb	0_{16}	RW	PLL loopback bandwidth. Reset only on POR.

26.11.64 SerDes Transmitter and Receiver Differential Pair Inversion Register

TABLE 26-67 shows the format of the SerDes Transmitter and Receiver Differential Pair Inversion register.

TABLE 26-67 SerDes Transmitter and Receiver Differential Pair Inversion Register – SERDES_INVPAIR_REG ($84\ 0000\ 08D8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:48	—	0_{16}	RO	*Reserved*
47:42	tx1_invpair	0_{16}	RW	Invert Channel 1 TXPi/TXNi if bit is 1. Reset only on POR.
41:28	tx0_invpair	0_{16}	RW	Invert Channel 0 TXPi/TXNi if bit is 1. Reset only on POR.
27:14	rx1_invpair	0_{16}	RW	Invert Channel 1 RXPi/RXNi if bit is 1. Reset only on POR.
13:0	rx0_invpair	0_{16}	RW	Invert Channel 0 RXPi/RXNi if bit is 1. Reset only on POR.

26.11.65 SerDes Test Configuration Bus Register

TABLE 26-68 shows the format of the SerDes Test Configuration Bus register.

TABLE 26-68 SerDes Test Configuration Bus Register – SERDES_TEST_CONFIG_BUS_REG ($84\ 0000\ 08E0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:32	—	0_{16}	RO	*Reserved*
31	fsr1_tx_entest	0_{16}	RW	Enable testing of FSR1 transmit ports. Reset only on POR.
30	fsr0_tx_entest	0_{16}	RW	Enable testing of FSR0 transmit ports. Reset only on POR.
29	fsr1_rx_entest	0_{16}	RW	Enable testing of FSR1 receive ports. Reset only on POR.
28	fsr0_rx_entest	0_{16}	RW	Enable testing of FSR0 receive ports. Reset only on POR.
27	fsr1_invpatt	0_{16}	RW	FSR1 invert polarity. Reset only on POR.
26:25	fsr1_rate	0_{16}	RW	FSR1 operating rate. Reset only on POR.
24	fsr1_enbspls	0_{16}	RW	FSR1 receiver pulse boundary scan. Reset only on POR.
23	fsr1_enbsrx	0_{16}	RW	FSR1 receiver boundary scan. Reset only on POR.
22	fsr1_enbstx	0_{16}	RW	FSR1 transmitter boundary scan. Reset only on POR.
21:20	fsr1_loopback	0_{16}	RW	FSR1 loopback. Reset only on POR.
19:18	fsr1_clkbyp	0_{16}	RW	FSR1 clock bypass. Reset only on POR.
17	fsr1_enrxpatt	0_{16}	RW	FSR1 enable RX patterns. Reset only on POR.
16	fsr1_entxpatt	0_{16}	RW	FSR1 enable TX patterns. Reset only on POR.
15:14	fsr1_testpatt	3_{16}	RW	FSR1 test patterns. Reset only on POR.
13	fsr0_invpatt	0_{16}	RW	FSR0 invert polarity. Reset only on POR.
12:11	fsr0_rate	0_{16}	RW	FSR0 operating rate. Reset only on POR.
10	fsr0_enbspls	0_{16}	RW	FSR0 receiver pulse boundary scan. Reset only on POR.
9	fsr0_enbsrx	0_{16}	RW	FSR0 receiver boundary scan. Reset only on POR.
8	fsr0_enbstx	0_{16}	RW	FSR0 transmitter boundary scan. Reset only on POR.
7:6	fsr0_loopback	0_{16}	RW	FSR0 loopback. Reset only on POR.

TABLE 26-68 SerDes Test Configuration Bus Register – SERDES_TEST_CONFIG_BUS_REG ($84\ 0000\ 08E0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
5:4	fsr0_clkbyp	0_{16}	RW	FSR0 clock bypass. Reset only on POR.
3	fsr0_enrxpatt	0_{16}	RW	FSR0 enable RX patterns. Reset only on POR.
2	fsr0_entxpatt	0_{16}	RW	FSR0 enable TX patterns. Reset only on POR.
1:0	fsr0_testpatt	3_{16}	RW	FSR0 test patterns. Reset only on POR.

26.11.66 SerDes PLL Status Register

TABLE 26-69 shows the format of the SerDes PLL Status register.

TABLE 26-69 SerDes PLL Status Register – SERDES_PLL_STATUS_REG ($84\ 0000\ 08e8_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:6	—	0_{16}	RO	*Reserved*
5:3	fsr1_stspll	0_{16}	RO	PLL lock status for FSR1 macros
2:0	fsr0_stspll	0_{16}	RO	PLL lock status for FSR0 macros

26.11.67 SerDes Test Status Register

TABLE 26-70 shows the format of the SerDes Test Status register.

TABLE 26-70 SerDes Test Status Register – SERDES_TEST_STATUS_REG ($84\ 0000\ 08F0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:48	—	0_{16}	RO	*Reserved*
47:38	fsr1_tx_testfail	0_{16}	RO	Test status for FSR1 transmit ports
37:28	fsr0_tx_testfail	0_{16}	RO	Test status for FSR0 transmit ports
27:14	fsr1_rx_testfail	0_{16}	RO	Test status for FSR1 receive ports
13:0	fsr0_rx_testfail	0_{16}	RO	Test status for FSR1 receive ports

26.11.68 Configuration Register Access Address Register

TABLE 26-71 shows the format of the Configuration Register Access Address register.

TABLE 26-71 Configuration Register Access Address Register – CONFIG_REG_ACCESS_ADDR_REG ($84\ 0000\ 0900_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0_{16}	RO	*Reserved*
15	channel	0_{16}	RW	Channel of Configuration register access.
14:11	amb	0_{16}	RW	AMB ID of Configuration register access.
10:2	data	0_{16}	RW	Address for Configuration register read or write. Bits 10:8 are function and bits 7:2 are offset within the function
1:0	—	0_{16}	RO	*Reserved*

26.11.69 Configuration Register Access Data Register

TABLE 26-72 shows the format of the Configuration Register Access Data register.

TABLE 26-72 Configuration Register Access Data Register – CONFIG_REG_ACCESS_DATA_REG ($84\ 0000\ 0908_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:32	—	0_{16}	RO	*Reserved*
31:0	data	0_{16}	RW	Data for Configuration register read or write. Writing to this register generates a Configuration register write on the FBD Channel. Reading from this register generates a Configuration register read on the FBD Channel.

26.11.70 AMB IBIST SBFIBPORTCTL and SBFIBPGCTL Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-73 shows the format of the AMB IBIST SBFIBPORTCTL and SBFIBPGCTL registers.

TABLE 26-73 AMB IBIST SBFIBPORTCTL and SBFIBPGCTL Register – SBFIBPORTCTL_SBFIBPGCTL_REG ($84\ 0000\ 0E80_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:56	—	0_{16}	RO	*Reserved*
55	sbfibportctl_ sbnbmap	0_{16}	RW	Loopback mapping bit. Reset only on POR.
54:38	—	0_{16}	RO	*Reserved*

TABLE 26-73 AMB IBIST SBFIBPORTCTL and SBFIBPGCTL Register – SBFIBPORTCTL_SBFIBPGCTL_REG (84 0000 $0E80_{16}$) (Count 4 Step 4096) *(Continued)*

Bit	Field	Initial Value	R/W	Description
37	sbfibportctl_ autoinvswpen	0_{16}	RW	Auto Inversion sweep enable. Reset only on POR.
36	—	0_{16}	RO	*Reserved*
35	sbfibportctl_ loopcon	0_{16}	RW	Loop forever. Reset only on POR.
34	sbfibportctl_ complete	0_{16}	RW1C	IBIST done flag. Reset only on POR.
33	sbfibportctl_ mstrmd	1_{16}	RO	Master mode enable. Reset only on POR.
32	sbfibportctl_ ibistr	0_{16}	RW	IBIST start. Setting this bit initiates a channel reset in which IBIST is run during Testing stage. Reset only on POR.
31:26	sbfibpgctl_ ovrloopcnt	F_{16}	RW	Overall Loop Count. Reset only on POR.
25:21	sbfibpgctl_ cnstgencnt	0_{16}	RW	Constant generator loop counter. Reset only on POR.
20	sbfibpgctl_ cnstgenset	0_{16}	RW	Constant generator setting. Reset only on POR.
19:13	sbfibpgctl_ modloopcnt	F_{16}	RW	Modulo-N loop counter. Reset only on POR.
12:10	sbfibpgctl_ modperiod	1_{16}	RW	Period of the modulo-*n* counter. Reset only on POR.
9:3	sbfibpgctl_ pattloopcnt	F_{16}	RW	Pattern buffer loop counter. Reset only on POR.
2:0	sbfibpgctl_ ptgenord	0_{16}	RW	Pattern generator order. Reset only on POR.

26.11.71 AMB IBIST SBFIBPATTBUF1 and SBFIBTXMSK Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-75 shows the format of the AMB IBIST SBFIBPATTBUF1 and SBFIBTXMSK registers.

TABLE 26-74 AMB IBIST SBFIBPATTBUF1 and SBFIBTXMSK Register – SBFIBPATTBUF1_SBFIBTXMSK_REG ($84\ 0000\ 0E88_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:56	—	0_{16}	RO	*Reserved*
55:32	sbfibpattbuf1	$2CCFD_{16}$	RW	IBIST pattern buffer. Reset only on POR.
31:10	—	0_{16}	RO	*Reserved*
9:0	sbfibtxmsk	$3FF_{16}$	RW	Selects which channels to enable for testing. Reset only on POR.

26.11.72 AMB IBIST SBFIBTXSHFT Register

This 32-bit register is documented in the FBDIMM AMB Specification.

TABLE 26-75 shows the format of the AMB IBIST SBFIBTXSHFT register.

TABLE 26-75 AMB IBIST SBFIBTXSHFT Register – SBFIBTXSHFT_REG ($84\ 0000\ 0E90_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:10	—	0_{16}	RO	*Reserved*
9:0	sbfibtxshft	$3FF_{16}$	RW	Transmitter Inversion Shift register. Reset only on POR.

26.11.73 AMB IBIST SBFIBPATTBUF2 and SBFIBPATT2EN Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-76 shows the format of the AMB IBIST SBFIBPATTBUF2 and SBFIBPATT2EN registers.

TABLE 26-76 AMB IBIST SBFIBPATTBUF2 and SBFIBPATT2EN Register – SBFIBPATTBUF2_SBFIBPATT2EN_REG ($84\ 0000\ 0EA0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:56	—	0_{16}	RO	*Reserved*
55:32	sbfibpattbuf2	$FD3302_{16}$	RW	IBIST Pattern Buffer 2. Reset only on POR.
31:10	—	0_{16}	RO	*Reserved*
9:0	sbfibpatt2en	0_{16}	RW	IBIST Pattern Buffer 2 enable. Reset only on POR.

26.11.74 AMB IBIST SBFIBINIT and SBIBISTMISC Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-77 shows the format of the AMB IBIST SBFIBINIT and SBIBISTMISC registers.

TABLE 26-77 AMB IBIST SBFIBINIT and SBIBISTMISC Register – SBFIBINIT_SBIBISTMISC_REG ($84\ 0000\ 0EB0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63	—	0_{16}	RO	*Reserved*
62:53	sbts0cnt	$C8_{16}$	RW	Southbound TS0 count. Reset only on POR.
52:45	sbts1cnt	1_{16}	RW	Southbound TS1 count. Reset only on POR.
44:35	sbdiscnt	100_{16}	RW	Southbound disable state count. Reset only on POR.
34	sbcaliben	1_{16}	RW	Southbound calibration enable. Reset only on POR.
33	—	0_{16}	RO	*Reserved*
32	sbfibiniten	0_{16}	RW	IBIST initialization enable. This bit performs no function in the MCU. The IBIST Start bit of the SBFIBPORTCTL register must be written to initiate IBIST. Bit Reset only on POR.
31:24	—	0_{16}	RO	*Reserved*
23:20	ambid	0_{16}	RW	Value of AMB ID field during TS0. Reset only on POR.
19:0	sbibistcalperiod	$61A80_{16}$	RW	Number of cycles to drive 1 during Calibration. Reset only on POR.

26.11.75 AMB IBIST NBFIBPORTCTL and NBFIBPGCTL Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification. For NBFIBPORTCTL, since the MCU will always be a slave on the northbound port, Bits 0, 1, 22, and 23 will not be used by the MCU and will be read only. Bit 1 will be 0_2 and bits 23:22 will be 0_{16}.

TABLE 26-78 shows the format of the AMB IBIST NBFIBPORTCTL and NBFIBPGCTL registers.

TABLE 26-78 AMB IBIST NBFIBPORTCTL and NBFIBPGCTL Register – NBFIBPORTCTL_NBFIBPGCTL_REG ($84\ 0000\ 0EC0_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:54	—	0_{16}	RO	*Reserved*
53:44	nbfibportctl_ errcnt	0_{16}	RW1C	Error counter. Reset only on POR.
43:40	nbfibportctl_ errlnnum	0_{16}	RO	Error lane number. Reset only on POR.
39:38	nbfibportctl_ errstat	0_{16}	RW1C	Port error status. Reset only on POR.
37	nbfibportctl_ autoinvswpen	0_{16}	RW	Auto inversion sweep enable. Reset only on POR.
36	nbfibportctl_ stoponerr	0_{16}	RW	Stop IBIST on error. Reset only on POR.
35	nbfibportctl_ loopcon	0_{16}	RW	Loop forever. Reset only on POR.
34	nbfibportctl_ complete	0_{16}	RW1C	IBIST done flag. Reset only on POR.
33	nbfibportctl_ mstrmd	0_{16}	RO	Master mode enable. Reset only on POR.
32	nbfibportctl_ ibistr	0_{16}	RW	IBIST start. Reset only on POR.
31:26	nbfibpgctl_ ovrloopcnt	F_{16}	RW	Overall loop count. Reset only on POR.
25:21	nbfibpgctl_ cnstgencnt	0_{16}	RW	Constant generator loop counter. Reset only on POR.
20	nbfibpgctl_ cnstgenset	0_{16}	RW	Constant generator setting. Reset only on POR.
19:13	nbfibpgct_ modloopcnt	F_{16}	RW	Modulo-*n* loop counter. Reset only on POR.
12:10	nbfibpgctl_ modperiod	1_{16}	RW	Period of modulo-*n* counter. Reset only on POR.
9:3	nbfibpgctl_ pattloopcnt	F_{16}	RW	Pattern buffer loop counter. Reset only on POR.
2:0	nbfibpgctl_ ptgenord	0_{16}	RW	Pattern generation order. Reset only on POR.

26.11.76 AMB IBIST NBFIBPATTBUF1 Register

This 32-bit register is documented in the FBDIMM AMB Specification.

TABLE 26-79 shows the format of the AMB IBIST NBFIBPATTBUF1 register.

TABLE 26-79 AMB IBIST NBFIBPATTBUF1 Register – NBFIBPATTBUF1_REG (84 0000 $0EC8_{16}$) (Count 4 Step 4096)

Field	Bit	Initial Value	R/W	Description
—	63:56	0_{16}	RO	*Reserved*
nbfibpattbuf1	55:32	$2CCFD_{16}$	RW	IBIST pattern buffer. Reset only on POR.
—	31:0	0_{16}	RO	*Reserved*

26.11.77 AMB IBIST NBFIBRXMSK Register

This 32-bit register is documented in the FBDIMM AMB Specification.

TABLE 26-80 shows the format of the AMB IBIST NBFIBRXMSK register.

TABLE 26-80 AMB IBIST NBFIBRXMSK Register – NBFIBRXMSK_REG (84 0000 $0ED0_{16}$) (Count 4 Step 4096)

Field	Bit	Initial Value	R/W	Description
—	64:46	0_{16}	RO	*Reserved*
nbfibrxmsk	45:32	$3FFF_{16}$	RW	NB IBIST receiver mask. Reset only on POR.
—	31:0	0_{16}	RO	*Reserved*

26.11.78 AMB IBIST NBFIBRXSHFT and NBFIBRXLNERR Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-81 shows the format of the AMB IBIST NBFIBRXSHFT and NBFIBRXLNERR register.

TABLE 26-81 AMB IBIST NBFIBRXSHFT and NBFIBRXLNERR Register – NBFIBRXSHFT_NBFIBRXLNERR_REG (84 0000 $0ED8_{16}$) (Count 4 Step 4096)

Field	Bit	Initial Value	R/W	Description
—	64:46	0_{16}	RO	*Reserved*
nbfibrxshft	45:32	$3FFF_{16}$	RW	Receiver Inversion Shift register. Reset only on POR.
—	31:14	0_{16}	RO	*Reserved*
nbfibrxlnerr	13:0	0_{16}	RO	Receiver error status. Reset only on POR.

26.11.79 AMB IBIST NBFIBPATTBUF2 and NBFIBPATT2EN Register

This pair of 32-bit registers are documented in the FBDIMM AMB Specification.

TABLE 26-82 shows the format of the AMB IBIST NBFIBPATTBUF2 and NBFIBPATT2EN register.

TABLE 26-82 AMB IBIST NBFIBPATTBUF2 and NBFIBPATT2EN Register – NBFIBPATTBUF2_NBFIBPATT2EN_REG (84 0000 $0EE0_{16}$) (Count 4 Step 4096)

Field	Bit	Initial Value	R/W	Description
—	63:56	0_{16}	RO	*Reserved*
nbfibpattbuf2	55:32	$FD3302_{16}$	RW	IBIST Pattern Buffer 2. Reset only on POR.
—	31:14	0_{16}	RO	*Reserved*
nbfibpatt2en	13:0	0_{16}	RW	IBIST Pattern Buffer 2 enable. Reset only on POR.

26.11.80 Other DRAM Registers

Other DRAM registers are defined in other chapters, particularly in the Performance Instrumentation chapter, Power Management chapter, and HW Debug chapter.

CHAPTER 27

Power Management

27.1 SPARC Power Management

The OpenSPARC T2 virtual processor contains extensive power management features. Besides clock gating at the core, there is clock gating within functional units down to the L1 clock headers.

As a safety device, OpenSPARC T2 includes an ASI register to enable lower-level clock gating.

The strands on each physical core share the `ASI_SPARC_PWR_MGMT` register. The `ASI_SPARC_PWR_MGMT` register, described in TABLE 27-1, is a hyperprivileged, read-write register located at ASI $4E_{16}$, VA 0_{16}.

TABLE 27-1 `ASI_SPARC_PWR_MGMT` Register Format

Bit	Field	Remarks
63:16	—	*Reserved*
15	dc	If 1, enables power management in data cache.
14	ic	If 1, enables power management in instruction cache.
13	ifu_cmu	If 1, enables power management in instruction cache miss unit.
12	ifu_ftu	If 1, enables power management in instruction fetch logic excluding instruction cache.
11	ifu_ibu	If 1, enables power management in instruction buffers.
10	dec	If 1, enables power management in decoder.
9	pku	If 1, enables power management in pick unit.
8	lsu	If 1, enables power management in load-store unit excluding data cache.
7	exu	If 1, enables power management in integer execution unit.
6	fgu	If 1, enables power management in floating-point and graphics unit.
5	tlu	If 1, enables power management in trap unit.
4	gkt	If 1, enables power management in crossbar interface.
3	spu	If 1, enables power management in stream processing unit.

TABLE 27-1 ASI_SPARC_PWR_MGMT Register Format

Bit	Field	Remarks
2	pmu	If 1, enables power management in performance monitor unit.
1	mmu	If 1, enables power management in memory management unit.
0	misc	If 1′ enables power management in remainder of SPARC core.

Reserved bits read as zeroes and are ignored on writes. The POR value of this register is 0. Writes to this register take effect immediately; this register is synchronizing so that all subsequent activity in the core is performed with the new setting of the power management enables when the next instruction is fetched from the thread that issued the write.

27.2 CPU Throttle Control

The CPU_THROTTLE_CTL register, shown in TABLE 27-2, displays the current value of the three CPU_THROTTLE_CTL pins on OpenSPARC T2. The CPU_THROTTLE_CTL pins are controlled by an external agent that does thermal monitoring of the OpenSPARC T2 chip.

CPU throttling is done by limiting instruction issue for each physical core over an eight-cycle window. At the lowest throttle setting of 0, a physical core issues an instruction on all eight cycles of a window. With each increase in the throttle setting, the physical core issues one fewer instruction per eight cycles, until at the maximum throttle setting of 7, the physical core issues only one instruction each eight-cycle window.

TABLE 27-2 Chip CPU Throttle Control – CPU_THROTTLE_CTL ($98\ 0000\ 0828_{16}$)

Bit	Field	Initial Value	R/W	Description
63:3	—	0	RO	*Reserved*
2:0	throttle	0	RO	Controls the issuing of instructions to the pipeline across an 8-cycle window as follows (I – issue, N – no issue): 0 – I I I I I I I I 1 – I I I I I I I N 2 – I I I N I I I N 3 – I I N I I N I N 4 – I N I N I N I N 5 – N N I N N I N I 6 – N N N I N N N I 7 – N N N N N N N I

27.3 Memory Access Throttle Control

27.3.1 DRAM Open Bank Max Register

The DRAM Open Bank Max register controls the maximum number of banks that can be open across the entire memory. A single register was desired; however, for implementation reasons, a copy of this register is present for each memory controller. All four registers must be set to the same value for the power throttle feature to function properly. Setting the registers to different values will result in undefined behavior.

TABLE 27-3 shows the format of the DRAM Open Bank Max register.

TABLE 27-3 DRAM Open Bank Max Register – DRAM_OPEN_BANK_MAX_REG ($84\ 0000\ 0028_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:17	—	0	RO	*Reserved*
16:0	freq	$1FFFF_{16}$	RW	Maximum banks open across the entire memory at any given time. Reset only on POR.

27.3.2 DRAM Programmable Time Counter Register

This register controls a programmable time value in which max banks can be open.

TABLE 27-4 shows the format of the DRAM Programmable Time Counter register.

TABLE 27-4 DRAM Programmable Time Counter Register – DRAM_PROG_TIME_CNTR_REG ($84\ 0000\ 0048_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	0	RO	*Reserved*
15:0	bunch	$FFFF_{16}$	RW	The time value in DRAM clocks. Reset only on POR.

27.4 DRAM Refresh Asynchronicity

Memory refresh operations draw more current than normal memory operations. To avoid current spikes, and to ease the burden on the system power distribution, the OpenSPARC T2 memory controller allows memory refresh asynchronicity, guaranteeing that all refreshes are at least 100 nsec from any other refresh.

Each DRAM controller has a background refresh timer, programmable to trigger a refresh every *n* cycles. Each time it triggers, the controller will schedule a refresh for all banks within a single rank on its channel. Subsequent refreshes will cycle through the ranks attached to that channel. When a refresh is scheduled, it is a high-priority request, so it is guaranteed to be issued in a relatively small number of cycles.

To get asynchronicity, the refresh counters on the different channel controllers must be started up $n/4$ DRAM cycles apart, after which they will remain neatly out of sync.

CHAPTER **28**

Configuration and Diagnostics Support

28.1 ASI_LSU_CONTROL_REG

Each virtual processor has a hyperprivileged `ASI_LSU_CONTROL_REG` register at ASI 45_{16}, VA{63:0} = 0, which contains fields that control several memory-related hardware functions. Nonprivileged or supervisor access to this register causes a *privileged_action* trap.

The format of the register is shown in TABLE 28-1.

TABLE 28-1 LSU Control Register – ASI_LSU_CONTROL_REG (ASI 45_{16}, VA 0_{16})

Bit	Field	Initial Value	R/W	Description
63:35	—	0	RO	*Reserved*
34:33	mode	0	RW	Watchpoint mode. Controls whether VA, PA, or no addresses are checked for data watchpoints. TABLE 28-2 lists the settings of the mode field.
32:25	bm	0	RW	Data watchpoint byte mask. For virtual or physical address data watchpointing, the ASI_DMMU_VA_WATCHPOINT register contains the virtual or physical address of a 64-bit word to be watched. The 8-bit bm controls which byte(s) within the 64-bit word should be watched. If all 8 bits are cleared, the watchpoint is disabled. If the watchpoint is enabled and a data reference overlaps any of the watched bytes in the watchpoint mask, a *VA_watchpoint* or *PA_watchpoint* trap is generated.
24	re	0	RO	Data watchpoint Read and Write Enable. If re or we is set, a read or write that matches the virtual or physical address in ASI_DMMU_VA_WATCHPOINT and the vm byte masks causes a *VA_watchpoint* or *PA_watchpoint* trap. Both re and we can be set to place a watchpoint for either a read or write access. Atomic operations are considered both a read and a write, and watchpoints for atomics are enabled if re, we, or both are set.
23	we	0	RO	
22:5	—	0	RO	*Reserved*
4	se	0	RW	Speculation enable. If set, loads are predicted to hit in the primary cache, branches are predicted not taken, and the floating-point unit predicts exceptions and runs in a pipelined mode.
3	dm	0	RW	DMMU enable. If cleared, when HPSTATE.hpriv = 0 the DMMU treats all data addresses as real addresses and performs a real-to-physical translation. See *Translation* on page 122 for more details.

TABLE 28-2 Mode Field Settings

Mode Field	Watchpoint Function
00_2	Watchpoint disabled
01_2	
10_2	Match on data physical address
11_2	Match on data virtual address

TABLE 28-3 VA and PA Data Watchpoint Byte Mask Examples

Watchpoint Mask	Address of Bytes Watched 7654 3210
00_{16}	Watchpoint disabled
01_{16}	0000 0001
32_{16}	0011 0010
FF_{16}	1111 1111

TABLE 28-1 LSU Control Register – ASI_LSU_CONTROL_REG (ASI 45_{16}, VA 0_{16}) *(Continued)*

Bit	Field	Initial Value	R/W	Description
2	im	0	RW	IMMU enable. If cleared, when HPSTATE.hpriv = 0 and HPSTATE.red = 0 the IMMU treats all instruction addresses as real addresses and performs a real-to-physical translation. See *Translation* on page 122.
1	dc	0	RW	Dcache enable. If cleared, the primary data cache does not allocate a line on a miss.
0	ic	0	RW	Icache enable. If cleared, the primary instruction cache does not allocate a line on a miss. **Note:** The Dcache and Icache are still kept coherent by OpenSPARC T2 when the dc and ic bits are set to 0. This includes updating the Dcache when stores from a strand hit in the Dcache.

28.2 Watchpoint Support

OpenSPARC T2 supports both instruction VA and data VA/PA watchpoints.

28.2.1 ASI_DMMU_WATCHPOINT

Note | Also see the ASI_LSU_CONTROL_REG definition in *ASI_LSU_CONTROL_REG* on page 413, which contains control bits for data watchpoint support.

Each virtual processor has a hyperprivileged ASI_DMMU_WATCHPOINT register at ASI 58_{16}, VA{63:0} = 38_{16} that is used for controlling the address (or address range if bytes are masked) for data PA and VA watchpoints. The format of the register is shown in TABLE 28-4.

TABLE 28-4 DMMU Watchpoint – ASI_DMMU_WATCHPOINT (ASI 58_{16}, VA $38)_{16}$

Bit	Field	Initial Value	R/W	Description
63:48	va_high	X	RO	Ignores writes, returns value of VA{47} on reads.
47:40	va	X	RW	VA watchpoint address{47:40}.
39:3	vapa	X	RW	VA/PA watchpoint address{39:3}.
2:0	—	0	RO	Bits 2:0 of the VA/PA watchpoint address are all zeros, watchpointing is done for 8-byte memory locations.

Nonprivileged and supervisor access to this register causes a *privileged_action* trap.

Notes The VA_watchpoint trap is never generated while executing in hyperprivileged mode. This implies that the AS_IF_USER ASIs will not generate a *VA_watchpoint* trap when accessed in hyperprivileged mode, even though a *VA_watchpoint* trap might be generated when accessed in user mode. The PA_watchpoint trap can be taken in hyperprivileged mode.

For quadword accesses, VA and PA watchpoint checking is only done on the lower 8 bytes of the access. This implies that a *VA_watchpoint* or *PA_watchpoint* trap will only be generated for a quadword load if vapa{3} is set to zero.

For quadword accesses, VA and PA watchpoint checking is only done on the lower 8 bytes of the access. This implies that a *VA_watchpoint* or *PA_watchpoint* trap will only be generated for a quadword load if vapa{3} is set to zero.

Implementation Note The data VA watchpoint comparison is done only on VA{47:3}, and does not mask ASI_DMMU_WATCHPOINT{47:32} when PSTATE.am = 1. This implies that out-of-range accesses to the VA hole will generate a *VA_watchpoint* trap if bits 47:3 and the byte mask match for data. In addition, when PSTATE.am = 1, bits 47:32 in the VA will be zero and ASI_DMMU_WATCHPOINT needs to have bits 47:32 loaded with zeros to produce a data VA watchpoint match.

28.2.2 ASI_IMMU_VA_WATCHPOINT

Each physical core has a pair of hyperprivileged ASI_IMMU_VA_WATCHPOINT register at ASI 50_{16}, VA{63:0} = 38_{16} that are used for controlling the address and enabling of instruction VA watchpoints. Strands 0-3 on the physical core share one of the registers, and strands 4-7 on the physical core share the other register. The format of the register is shown in TABLE 28-5.

TABLE 28-5 IMMU VA Watchpoint – ASI_IMMU_VA_WATCHPOINT (ASI 50_{16}, VA $38)_{16}$

Bit	Field	Initial Value	R/W	Description
63:48	va_high	X	RO	Ignores writes, returns value of VA{47} on reads.
47:2	va	X	RW	VA watchpoint address {47:2}.
1	—	0	RO	*Reserved*
0	enable	0	RW	If 1, instruction VA watchpoint is enabled. If 0, instruction VA watchpoint is disabled.

Nonprivileged and supervisor access to this register causes a *privileged_action* trap.

Notes

The *instruction_VA_watchpoint* trap is never generated while executing in hyperprivileged mode.

No *instruction_VA_watchpoint* trap is generated by OpenSPARC T2 for real-to-physical translations (RA → PA in TABLE 12-15 on page 123).

Implementation Note

The instruction VA watchpoint comparison is done only on VA{47:2}, and does not mask `ASI_IMMU_VA_WATCHPOINT`{47:32} when PSTATE.am = 1. This implies that out-of-range accesses to the VA hole will generate an *instruction_VA_watchpoint* trap if bits 47:2 match. In addition, when PSTATE.am = 1, bits 47:32 in the VA will be zero and `ASI_IMMU_VA_WATCHPOINT` needs to have bits 47:32 loaded with zeros to produce an instruction VA watchpoint match.

28.3 Breakpoint Support

28.3.1 ASI_INST_MASK_REG

Each physical core has a pair of hyperprivileged `ASI_INST_MASK_REG` registers at ASI 42_{16}, VA{63:0} = 8_{16}. Strands 0–3 on the physical core share one of the registers, and strands 4–7 on the physical core share the other register. This register is used to disable execution of a particular instruction or class of instructions for diagnostic or debug purposes, under the control of HPSTATE.ibe. If HPSTATE.ibe = 1 and any of the enable fields are set to 1, execution of any instruction that matches all the enabled bits in the inst field of this register results in an *instruction_breakpoint* trap. Nonprivileged or supervisor access to this register causes a *privileged_action* trap.

TABLE 28-6 defines the format of the `ASI_INST_MASK_REG` register.

TABLE 28-6 SPARC Instruction Mask Register – `ASI_INST_MASK_REG` (ASI 42_{16}, VA 8_{16})

Bit	Field	Initial Value	R/W	Description
63:39	—	0	RO	*Reserved*
38	enb31_30	0	RW	Enable matching on INST 31:30.
37	enb29_25	0	RW	Enable matching on INST 29:25.
36	enb24_19	0	RW	Enable matching on INST 24:19.
35	enb18_14	0	RW	Enable matching on INST 18:14.

TABLE 28-6 SPARC Instruction Mask Register – ASI_INST_MASK_REG (ASI 42_{16}, VA 8_{16}) *(Continued)*

Bit	Field	Initial Value	R/W	Description
34	enb13	0	RW	Enable matching on INST 13.
33	enb12_5	0	RW	Enable matching on INST 12:5.
32	enb4_0	0	RW	Enable matching on INST 4:0.
31:0	inst	0	RW	Instruction pattern to be trapped.

Programming Note While there is a pair of instruction mask registers per physical processor core, each strand is under the control of its own HPSTATE.ibe bit. All HPSTATE.ibe bits for the strands in a physical processor core must be set to have the physical processor core fully trap on the instruction pattern.

28.3.2 Trap on Control Transfer

If PSTATE.tct is set, a taken control transfer will result in a *control_transfer_instruction* trap. Control transfers include conditional branches, jumps, done, and retry. The trap is precise and taken before the instruction executes. Once the trap is taken, PSTATE.tct is cleared. The virtual address of the control transfer instruction will be contained in TPC[TL].

28.4 Instruction and Data Cache Control

28.4.1 ASI_LSU_DIAG_REG

Each physical core has a hyperprivileged ASI_LSU_DIAG_REG register at ASI 42_{16}, VA{63:0} = 10_{16}. This register disables associativity in the primary instruction and/or data caches for diagnostic or debug purposes. Nonprivileged or supervisor access to this register causes a *privileged_action* trap.

TABLE 28-7 defines the format of the ASI_LSU_DIAG_REG register.

TABLE 28-7 LSU Diagnostic Register – ASI_LSU_DIAG_REG (ASI 42_{16}, VA 10_{16})

Bit	Field	Initial Value	R/W	Description
63:2	—	0	RO	*Reserved*
1	dassocdis	0	RW	If 1, the Dcache replacement algorithm is not LRU but instead uses bits 12:11 of the virtual address to determine the replacement way when a miss occurs
0	iassocdis	0	RW	If 1, the Icache replacement algorithm is not LRU but instead uses bits 13:11 of the virtual address to determine the replacement way when a miss occurs.

28.5 L1 I-Cache Diagnostic Access

28.5.1 ASI_ICACHE_INSTR

Each virtual processor shares a read/write, hyperprivileged ASI_ICACHE_INSTR register at ASI 66_{16}, VA 0_{16}– $7FF8_{16}$ which provides diagnostic access to the Icache data array.

TABLE 28-8 describes the VA format when accessing ASI_ICACHE_INSTR.

TABLE 28-8 ASI_ICACHE_INSTR Address Format

VA Bits	Field	Remarks
63:15	—	*Reserved*
14:12	way{2:0}	Selects cache way to be read or written.
11:6	index{5:0]	Selects cache index to be read or written.
5:3	word	Selects aligned 4 bytes out of 32 bytes in cache line to be read or written. (See below for data dependency on bit 4.)
2:0	—	*Reserved*

Note | The diagnostic VA is shifted left one bit from what would normally be used to index the cache to fetch an instruction (for example, bits 11:3 are used instead of bits 10:2).

The data field is interpreted as shown in TABLE 28-9.

TABLE 28-9 ASI_ICACHE_INSTR Register

Operation Type	Data Bits	Field	Remarks
Store if VA[4]==0	63:33	—	*Reserved*
	32	perrinj	**xor** this bit with generated parity bit
	31:0	instr{31:0}	32 bits of instruction data
Store if VA[4]==1	63:33	—	*Reserved*
	32	instr{0}	bit 0 of instruction data
	31	instr{1}	bit 1 of instruction data
	...	...	...
	1	instr{31}	bit 31 of instruction data
	0	perrinj	**xor** this bit with generated parity bit
Data read if VA[4]==0	63:33	—	*Reserved*
	32	parity	Parity bit read from array
	31:0	instr{31:0}	32 bits of instruction data read from array
Data read if VA[4]==1	63:33	—	*Reserved*
	32	instr{0}	bit 0 of instruction data read from array
	31	instr{1}	bit 1 of instruction data read from array
	...	...	...
	1	instr{31}	bit 31 of instruction data read from array
	0	parity	Parity bit read from array

Note Since hardware does not keep diagnostic stores to the data array consistent with the L2 reverse directory, the L2 cache, and other instruction and data caches, software must flush the L2 cache or otherwise restore system caches to a coherent state after performing a diagnostic store to the data array.

The bit ordering of the data to or from the instruction cache depends on the word address. Specifically, on read or write via this ASI, if bit 4 of the virtual address is set, then the parity and instruction bits (32:0) must be bit-reversed (i.e., bit 0 becomes bit 32, bit 1 becomes bit 31, etc.) to be consistent with functional read and write paths.

28.5.2 ASI_ICACHE_TAG

Each virtual processor has a shared, hyperprivileged, read-write ASI_ICACHE_TAG register located at ASI 67_{16}, VA 0_{16}–$FFE0_{16}$, which provides diagnostic access to the Icache tags and valid bit array. Diagnostic reads of the tag and valid bit arrays do not cause parity errors. The VA format is shown in TABLE 28-10.

TABLE 28-10 ASI_ICACHE_TAG Address Format

VA Bits	Field	Remarks
63:17	—	*Reserved*
16	perren	**xor**s this value with the computed tag parity bit (even parity is computed and stored; ignored on a read)
15	vb_err_en	Forces the slave copy of the valid bit to be the inversion of the master copy (ignored on a read)
14:12	way{2:0}	Identifies the way whose tag and valid bit will be accessed
11:6	index{5:0}	Identifies the index whose tag and valid bit will be accessed
5:0	—	*Reserved*

Note The diagnostic VA is shifted left one bit from what would normally be used to index the cache to fetch an instruction (for example, bits 11:3 are used instead of bits 10:2).

The data format is shown in TABLE 28-11.

TABLE 28-11 ASI_ICACHE_TAG Register

Operation Type	Data bits	Field	Remarks
Store	63:31	—	*Reserved*
	30:2	tag{39:11}	Contents of the tag array.
	1	valid{1}	valid bit master copy.
	0	—	*Reserved*. valid bit slave copy (ignored on write).
Load	63:32	—	*Reserved*
	31	Tag Parity	parity bit of the tag entry.
	30:2	tag{39:11}	Contents of the tag array.
	1	valid{1}	valid bit master copy.
	0	valid{0}	valid bit slave copy.

Notes Since hardware does not keep diagnostic stores to the tag and valid bit arrays consistent with the L2 reverse directory, the L2 cache, and other instruction and data caches, software must flush the L2 cache or otherwise restore system caches to a coherent state after performing a diagnostic store to the L1 tags and valid bit array.

The Icache Tag includes physical address bit 39, even though in normal system operation bit 39 will always be 0 for any cacheable access.

28.6 L1 D-Cache Diagnostic Access

28.6.1 ASI_DCACHE_DATA

Each virtual processor has a shared, hyperprivileged, read-write ASI_DCACHE_DATA register located at ASI 46_{16}, VA 0_{16}–$1FFFF8_{16}$ that is used for diagnostic accesses to the L1 data cache data array. Stores update 64 bits of the data array at one time and can optionally invert each computed byte parity bit independently. Hardware computes and stores even parity. Loads specify whether to load parity or data bits. Loads that specify parity bits load all eight byte parity bits at a time. Loads that specify data bits load 64 data bits at a time.

TABLE 28-12 describes the VA format for a store to ASI_DCACHE_DATA.

TABLE 28-12 ASI_DCACHE_DATA Store Address Format

VA Bits	Field	Remarks
63:21	—	*Reserved*
20:13	perrmask{7:0}	**xor**s this value with computed parity bits; bit 7 corresponds to parity for bits 63:56, bit 6 to bits 55:48, etc.
12:11	way{1:0}	Selects cache way to be written.
10:4	Index{6:0}	Selects cache index to be written.
3	doubleword	Selects aligned 8 bytes out of 16 bytes in cache line to be written.
2:0	—	*Reserved*

For a load to ASI_DCACHE_DATA, the VA is interpreted as follows.

TABLE 28-13 ASI_DCACHE_DATA Load Address Format

VA Bits	Field	Remarks
63:14	—	*Reserved*
13	data_notparity	If 1, selects data field to be returned by load; if 0, selects parity bits to be returned by load.
12:11	way{1:0}	Selects cache way to be written.
10:4	index{6:0}	Selects cache index to be read.
3	doubleword	Selects aligned 8 bytes out of 16 bytes in cache line to be read.
2:0	—	*Reserved*

The data field is interpreted as follows.

TABLE 28-14 ASI_DCACHE_DATA Format

Operation Type	Data Bits	Field	Remarks
Store	63:0	data{63:0}	Contains the 64-bit data field to be written to the data array.
Data read	63:0	data{63:0}	Contains the 64-bit data field read from the data array.
Parity read	63:8	—	*Reserved.* Read as all zeroes.
	7:0	parity{7:0}	Contains the 8 parity bits read from the data array.

Note | Since hardware does not keep diagnostic stores to the data array consistent with the L2 reverse directory, the L2 cache, and instruction and other data caches, software must flush the L2 cache or otherwise restore system caches to a coherent state after performing a diagnostic store to the data array.

28.6.2 ASI_DCACHE_TAG

Each virtual processor has a shared, read/write, hyperprivileged ASI_DCACHE_TAG register located at ASI 47_{16}, VA 0_{16}–$7FF0_{16}$. This register reads and writes the contents of the Dcache valid bit and tag arrays for diagnostic purposes.

The VA field is interpreted as follows.

TABLE 28-15 ASI_DCACHE_TAG Address Format

VA Bits	Field	Remarks
14	vb_err_en	Forces the slave copy of the valid bit to be the inversion of the master copy (ignored on a read).
13	perren	**xor**s this value with the computed tag parity bit (even parity is computed and stored; ignored on a read).
12:11	way{1:0}	Identifies the way whose tag and valid bit will be accessed.
10:4	index{6:0}	Identifies the index whose tag and valid bit will be accessed.
3:0	—	*Reserved*

The data field is interpreted as follows.

TABLE 28-16 ASI_DCACHE_TAG Format

Operation Type	Data Bits	Field	Remarks
Store	63:31	—	*Reserved*
	30:2	tag{39:11}	Contents of the tag array.
	1	valid{1}	valid bit master copy.
	0	—	*Reserved.* valid bit slave copy (ignored on write).
Load	63:32	—	*Reserved*
	31	Tag Parity	Parity bit of the tag entry.
	30:2	tag{39:11}	Contents of the tag array.
	1	valid{1}	valid bit master copy.
	0	valid{0}	valid bit slave copy.

Loads from this ASI do not cause parity errors if the stored parity is bad or if the valid bit copies differ from each other.

Note | Since hardware does not keep diagnostic stores to the tag and valid bit arrays consistent with the L2 reverse directory, the L2 cache, and instruction and other data caches, software must flush the L2 cache or otherwise restore system caches to a coherent state after performing a diagnostic store to the L1 tags and valid bit array.

28.7 Integer Register File

To read or write the IRF data portions, normal SPARC integer loads and stores or arithmetic instructions can be used. Instructions that read the IRF data may result in ECC errors. These ECC errors can be suppressed by setting CERER.irf or SETER.pscce to 0.

To read the ECC check bits for the IRF, software can load from the ASI_IRF_ECC_REG, described below.

28.7.1 ASI_IRF_ECC_REG

Each virtual processor has a hyperprivileged, read-only ASI_IRF_ECC_REG located at ASI 48_{16}, VA 0_{16}–$F8_{16}$. A load to this register with VA bits 7:3 set to the 5-bit register number to be read returns the ECC bits of the IRF register entry accessed by VA bits 4:0, in the current register window. Nonprivileged or supervisor access to this register causes a *privileged_action* trap. Writes to this register cause a *DAE_invalid_ASI*.

The format of the ASI_IRF_ECC_REG address is as follows.

TABLE 28-17 ASI_IRF_ECC_REG address

Bit	Field	Remarks
63:8	—	*Reserved*. Ignored
7:3	irf_index	Integer register number to be read. Note this should be the same as the index contained in the DSFARwhen an error occurs. See Table 25-11 on page 253 for details.
2:0	—	*Reserved*

When a read to this register occurs, the format of the returned data is as follows.

TABLE 28-18 ASI_IRF_ECC_REG format

Bit	Field	Remarks
63:8	—	*Reserved.* Reads return zeroes.
7:0	ecc	ECC bits read from IRF.

A read of this register never results in an ECC error.

28.8 Floating-Point Register File

To read or write the FRF data portions, normal SPARC FGU loads and stores or arithmetic instructions can be used. Instructions that read the FRF data may result in ECC errors. These ECC errors can be suppressed by setting CERER.frf or CETER.pscce to 0.

To read the ECC check bits for the FRF, software can load from the ASI_FRF_ECC_REG, described below.

28.8.1 ASI_FRF_ECC_REG

Each virtual processor has a hyperprivileged, read-only ASI_FRF_ECC_REG located at ASI 49_{16}, VA 0_{16}–$F8_{16}$. A load to this register with VA bits 7:3 set to the double-precision 6-bit register number returns the even and odd ecc bits for the even and odd halves of the FRF register entry accessed by VA bits 7:3. Nonprivileged or supervisor access to this register causes a *privileged_action* trap. Writes to this register cause a *DAE_invalid_ASI*. The format of the ASI_FRF_ECC_REG is as follows.

TABLE 28-19 ASI_FRF_ECC_REG Address

Bit	Field	Remarks
63:8	—	*Reserved*. Ignored
7:3	frf_index{4:0}	Index in the FRF register file to be read. The contents of this field specify the precise index of the FRF to be read, not the encoded SPARC V7 FRF register number; e.g., an FRF_index{4:0} value of 10110_2 will read FRF index 22.
2:0	—	*Reserved*. Ignored.

When a read to this register occurs, the format of the returned data is as follows.

TABLE 28-20 ASI_FRF_ECC_REG Format

Bit	Field	Remarks
63:14	—	*Reserved*. Reads return zeroes.
13:7	ecc_even	ECC bits read from FRF for even half of register.
6:0	ecc_odd	ECC bits read from FRF for odd half of register.

A read of this register never results in an ECC error.

28.9 Store Buffer — ASI_STB_ACCESS

Diagnostic access to the store buffer is complicated by the width of each entry and the fact that all stores (even diagnostic ASI stores) are written to the store buffer.

Each strand has a hyperprivileged, read-only ASI_STB_ACCESS register located at ASI $4A_{16}$, VA 0_{16}–$3F_{16}$. Diagnostic software can read any entry (belonging to its own strand) by using the ASI_STB_ACCESS register.

The store buffer is divided into three parts. The cam field contains bits 39:3 of the physical address, 8 byte marks, and a parity bit. The data array contains two subfields. The first field consists of a 64-bit data field and two 7-bit SEC/DED ecc fields (ECC is stored on a 32-bit word basis). The second field of the data array consists of encoded privilege-level information (encoded with hamming [?] distance of 2 between valid codes for a total of 3 bits). Finally, as will be explained shortly, there is a read that returns the current store buffer pointer for the thread.

To read a store buffer entry, diagnostic software issues a load to the ASI_STB_ACCESS register, specifying the entry and subfield of the store buffer to read in the virtual address of the load, as detailed below.

Software cannot write directly into a store buffer entry. All stores are placed into the store buffer. This includes stores to ASI registers as well as stores to memory and I/O space.

Software can inject an error into the store buffer by setting the stau or stdu bit in the ASI_ERROR_INJECT_REG. Hardware **xor**s the values of the computed ecc bits with ASI_ERROR_INJECT_REG.eccmask{6:0} if stdu is set, or **xor**s the computed parity for the address with ASI_ERROR_INJECT_REG.eccmask{6:0} if stau is set. If multiple bits are set, hardware **xor**s each field independently. Once software sets the enb_hp bit in the ASI_ERROR_INJECT_REG, all future stores will be written to the store buffer with errors injected according to which of the stau or stdu bits is set. Errors cease to be injected once software resets the enb_hp bit. Note that hardware will not inject an error for a store to the ASI_ERROR_INJECT_REG.

Software has control over the contents of any of the store buffer subfields. For example, the contents of the store data field and the type of store determine the settings of the data field data bits and byte mark bits. The privilege level of the store also determine the contents of the privilege level field.

The format of the va field for accesses to ASI_STB_ACCESS is detailed below.

TABLE 28-21 ASI_STB_ACCESS Address Format

Operation Type	Bit	Field	Remarks
Store buffer read	63:9	—	*Reserved*
	8:6	field	Determines subset of store buffer entry to be read as follows: 000 – Store buffer data 001 – Store buffer data ECC (14 bits) 010 – Store buffer control and address parity 011 – Store buffer address and byte marks 100 – Current store buffer pointer 101 – *Reserved* 110 – *Reserved* 111 – *Reserved*
	5:3	entry	Contains store buffer entry ID (of this thread's store buffer) to read.
	2:0	—	*Reserved*

The data field of a load or store to the ASI_STB_ACCESS field is detailed below.

TABLE 28-22 ASI_STB_ACCESS Format

Operation Type	Bit	Field	Remarks
STB data read	63:0	data	Contains data in the store buffer entry addressed by ASI_STB_ACCESS.
STB data ECC read	63:14	—	*Reserved.* Read as all zeroes.
	13:7	ecc_odd	ECC for bits 63:32 of the data field.
	6:0	ecc_even	ECC for bits 31:0 of the data field.
STB control read	63:4	—	*Reserved*. Read as all zeroes.
	3	c_p	Parity over bm_asi and PA{39:3}.
	2:0	control	Privilege level encoding: 000 – user; 011 – priv; 101 – hpriv; all others – parity error).
STB CAM read	63:48	—	Read as all zeroes.
	47:40	bm_asi	Byte marks{7:0} or ASI value.
	39:3	pa{39:3}	Contents of PA cam field.
	2:0	—	*Reserved.* Read as all zeroes.

28.10 Scratchpad Registers

Each strand has a hyperprivileged, read-only ASI_SCRATCHPAD_ACCESS register located at ASI 59_{16}, VA 0_{16}–78_{16}.

Diagnostic software can read the data bits or ecc bits using this register. The format and usage of this register are described below.

TABLE 28-23 ASI_SCRATCHPAD_ACCESS Address Format

Operation Type	Bit	Field	Remarks
Load	63:7	—	*Reserved*
	6	data_np	If 1, return the data entry contents; if 0, return the eight 8 ecc bits in bit positions 7:0 of the returned data field.
	5:3	index	Index in Scratchpad array to be read.
	2:0	—	*Reserved*

The data returned by an access to the ASI_SCRATCHPAD_ACCESS register is defined as follows.

TABLE 28-24 ASI_SCRATCHPAD_ACCESS Format

Operation Type	Data Bits	Field	Remarks
Data read	63:0	data{63:0}	Contains the 64-bit data field read from the data array.
ECC read	63:8	—	*Reserved*. Read as all zeroes
	7:0	ecc{7:0}	Contains the eight ecc bits read from the data array.

Diagnostic reads to the scratchpad array using this ASI never result in ECC errors.

28.11 Tick Compare

Each strand has a hyperprivileged, read-only ASI_TICK_ACCESS register located at ASI $5A_{16}$, VA 0_{16}–38_{16}.

Diagnostic software can read the data or ecc bits using this register.

Implementation Note The intdis bit of the various TICK_CMPR registers is inverted before being used in the ECC calculation.

The format and usage of this register are described below.

TABLE 28-25 ASI_TICK_ACCESS Address Format

Operation Type	Bit	Field	Remarks
Load	63:6	—	*Reserved*
	5	data_np	If 1, returns the data entry contents; if 0, returns the 8 ecc bits in bit positions 7:0 of the returned data field.
	4:3	index{1:0}	Index in TICK_CMPR array to be read: 00 – TICK_CMPR; 01 – STICK_CMPR; 10 – HSTICK_COMPARE; 11 – *Reserved*.
	2:0	—	*Reserved*

The data returned by an access to the ASI_TICK_ACCESS register is defined as follows.

TABLE 28-26 ASI_TICK_ACCESS Format

Operation Type	Data Bits	Field	Remarks
Data read	63:0	data{63:0}	Contains the 64-bit data field read from the data array
ECC read	63:8	—	*Reserved*. Read as all zeroes
	7:0	ecc{7:0}	Contains the 8 ecc bits read from the data array

Diagnostic reads to the TICK_CMPR array using this ASI never result in ECC errors.

In order to cause an ECC error, software can first set up the ASI_ERROR_INJECT register as described in *ASI_ERROR_INJECT_REG* on page 260. Then, software can write to the TICK_CMPR registers using normal instructions.

28.12 Trap Stack Array (TSA)

Each strand has a hyperprivileged, read-only ASI_TSA_ACCESS register located at ASI $5B_{16}$, VA 0_{16}–38_{16}.

Diagnostic software can read the ecc bits by using this register. The format and usage of this register are described below.

TABLE 28-27 ASI_TSA_ACCESS Address Format

Operation Type	Bit	Field	Remarks
Load	63:6	—	*Reserved*
	5:3	index{2:0}	Index in TSA to be read
	2:0	—	*Reserved*

The data returned by an access to the ASI_TSA_ACCESS register is defined as follows:.

TABLE 28-28 ASI_TSA_ACCESS Format

Operation Type	Data Bits	Field	Remarks
ECC read	63:16	—	*Reserved*. Read as all zeroes
	15:0	ecc{15:0}	Contains the 16 ecc bits read from the data array

Diagnostic reads to the TSA using this ASI never result in ECC errors.

To cause an ECC error, software can first set up the ASI_ERROR_INJECT register as described in *ASI_ERROR_INJECT_REG* on page 260. Then, software can write to the TSA using normal instructions.

The TSA stores several logical registers. TABLE 28-29 documents the internal organization of the array and the mapping of the contents of the logical registers to their physical locations within the array. Note that some bits of some entries of the array are reserved for hardware use. If a correctable ECC error occurs in these bits or an error occurs in the ecc bits themselves, software can correct the error by doing the following:

1. Disabling correctable ECC checking on the TSA by setting CERER.tsac to 0.
2. Reading any logical register in the entry with an error.
3. Writing this logical register back with the same data as was read.

This causes the ECC to be recalculated.

TABLE 28-29 TSA Entry Contents

Entry	Data Bits	Logical Description	Remarks
0	151		Not software-accessible; reserved for hardware VA out-of range detection in the event that a trap handler enables translation or changes PSTATE.am. If an error causes this bit to flip from 1 to 0 and the trap handler enables translation or changes PSTATE.am, it is possible that the hardware will fail to take an *instruction_address_range* or *instruction_real_range* trap after the done or retry. If an error causes this bit to flip from 0 to 1 and the trap handler enables translation or changes PSTATE.am, it is possible that the hardware will incorrectly take an *instruction_address_range* or *instruction_real_range* trap after the DONE or RETRY instruction.
	150		Not software-accessible; reserved for hardware VA out-of-range detection in the event that a trap handler enables translation or changes PSTATE.am. If an error causes this bit to flip from 1 to 0 and the trap handler enables translation or changes PSTATE.am, it is possible that the hardware will fail to take an *instruction_address_range* or *instruction_real_range* trap after the done or retry. If an error causes this bit to flip from 0 to 1 and the trap handler enables translation or changes PSTATE.am, it is possible that the hardware will incorrectly take an *instruction_address_range* or *instruction_real_range* trap after the DONE or RETRY instruction.

TABLE 28-29 TSA Entry Contents *(Continued)*

Entry	Data Bits	Logical Description	Remarks
	149:142	ECC for bits 151,150,133:68	
	141:134	ECC for bits 67:0	
	133	—	Not software-accessible; *reserved* for hardware usage
	132:131	TSTATE{41:40}	TSTATE1.GL{1:0}
	130:123	TSTATE1{39:32}	TSTATE1.CCR
	122:115	TSTATE1{31:24}	TSTATE1.ASI
	114	HTSTATE{10}	HTSTATE.ibe
	113	TSTATE1{17}	TSTATE1.PSTATE.cle
	112	TSTATE1{16}	TSTATE1.PSTATE.tle
	111	TSTATE1{20}	TSTATE1.PSTATE.tct
	110	HTSTATE1{2}	HTSTATE.priv
	109	HTSTATE1{5}	HTSTATE1.red
	108	TSTATE1{12}	TSTATE1.PSTATE.pef
	107	TSTATE1{11}	TSTATE1.PSTATE.am
	106	TSTATE1{10}	TSTATE1.PSTATE.priv
	105	TSTATE1{9}	TSTATE1.PSTATE.ie
	104	HTSTATE1{0}	HTSTATE1.tlz
	103:101	TSTATE1{2:0}	TSTATE1.CWP{2:0}
	100:92	TT1{8:0}	
	91:46	TPC1{47:2}	
	45:0	TNPC1{47:2}	
1–5		All	Same as entry 0; Trap Stack entries 2-6

TABLE 28-29 TSA Entry Contents *(Continued)*

Entry	Data Bits	Logical Description	Remarks
6	149:142	ECC for bits 133:67	
	141:134	ECC for bits 66:0	
	133:92	—	*Reserved*. Unused
	91:80	mondo_queue_tail{17:6}	
	79:68	dev_queue_tail{17:6}	
	67:46	—	*Reserved*. Unused
	45:34	mondo_queue_head{17:6}	
	33:22	dev_queue_head{17:6}	
	21:0	—	*Reserved*. Unused
7	149:142	ECC for bits 133:67	
	141:134	ECC for bits 66:0	
	133:92	—	*Reserved*. Unused
	91:80	res_err_queue_tail{17:6}	
	79:68	nonres_err_queue_tail{17:6}	
	67:46	—	*Reserved*. Unused
	45:34	res_err_queue_head{17:6}	
	33:22	nonres_err_queue_head{17:6}	
	21:0	—	*Reserved*. Unused

28.13 MMU Register Array (MRA)

Each strand has a hyperprivileged, read-only `ASI_MRA_ACCESS` register located at ASI 51_{16}, VA 0_{16}–38_{16}.

Diagnostic software can read the parity bits using this register. The format and usage of this register are described below.

TABLE 28-30 ASI_MRA_ACCESS Address Format

Operation Type	Bit	Field	Remarks
Load	63:6	—	*Reserved*
	5:3	index{2:0}	Index in MRA array to be read.
	2:0	—	*Reserved*

The data returned by an access to the ASI_MRA_ACCESS register is defined as follows.

TABLE 28-31 ASI_MRA_ACCESS Data Format

Operation Type	Data Bits	Field	Remarks
Parity read	63:2	—	*Reserved*. Read as all zeros.
	1:0	pty{1:0}	Contains the parity bits for each 41-bit subfield.

Diagnostic reads to the MRA using this ASI never result in parity errors.

To cause a parity error, software can first set up the ASI_ERROR_INJECT register as described in *ASI_ERROR_INJECT_REG* on page 260. Then, software can write to the MRA using ASI store instructions.

The MRA stores several MMU configuration registers. TABLE 28-32 documents the internal organization of the array and the mapping of the contents of the logical registers to their physical locations within the array. Note that bits 81:78 of the array are not used for entries 3:0. Software can correct any MRA parity error by reloading the entry from a “clean” copy in memory. Hardware generates the correct parity for the entry as it is written back to the array.

TABLE 28-32 MRA Entry Contents

Entry	Bit	Logical Description	Remarks
0	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81:77	—	*Reserved*
	76:75	z_tsb_config_0{62:61}	
	74:48	z_tsb_config_0{39:13}	
	47:39	z_tsb_config_0{8:0}	
	38	—	*Reserved*
	37:36	z_tsb_config_1{62:61}	
	35:9	z_tsb_config_1{39:13}	
	8:0	z_tsb_config_1{8:0}	
1	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81:77	—	*Reserved*
	76:75	z_tsb_config_2{62:61}	
	74:48	z_tsb_config_2{39:13}	
	47:39	z_tsb_config_2{8:0}	
	38	—	*Reserved*
	37:36	z_tsb_config_3{62:61}	
	35:9	z_tsb_config_3{39:13}	
	8:0	z_tsb_config_3{8:0}	
2	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81:77	—	*Reserved*
	76:75	nz_tsb_config_0{62:61}	
	74:48	nz_tsb_config_0{39:13}	
	47:39	nz_tsb_config_0{8:0}	
	38	—	*Reserved*
	37:36	nz_tsb_config_1{62:61}	
	35:9	nz_tsb_config_1{39:13}	
	8:0	nz_tsb_config_1{8:0}	

TABLE 28-32 MRA Entry Contents *(Continued)*

Entry	Bit	Logical Description	Remarks
3	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81:77	—	*Reserved*
	76:75	nz_tsb_config_2{62:61]	
	74:48	nz_tsb_config_2{39:13]	
	47:39	nz_tsb_config_2{8:0]	
	38	—	*Reserved*
	37:36	nz_tsb_config_3{62:61]	
	35:9	nz_tsb_config_3{39:13]	
	8:0	nz_tsb_config_3{8:0]	
4	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81	real_range_0{63]	Unused
	80:27	real_range_0{53:0]	
	26:0	physical_offset_0{39:13]	
5	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81	real_range_1{63]	Unused
	80:27	real_range_1{53:0]	
	26:0	physical_offset_1{39:13]	
6	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81	real_range_2{63]	Unused
	80:27	real_range_2{53:0]	
	26:0	physical_offset_2{39:13]	
7	83	Parity for bits 81:41	
	82	Parity for bits 40:0	
	81	real_range_3{63]	Unused
	80:27	real_range_3{53:0]	
	26:0	physical_offset_3{39:13]	

28.14 L2 Cache Registers

This section discusses L2 control and status registers.

28.14.1 L2 Control Register

Each cache bank has a control register. To enable the L2 cache, the dis bit must be set to 0 for all enabled banks. The L2 cache can be disabled by setting the dis bit for all enabled banks to 1. Operation when the dis bit of the enabled L2 banks are not all the same is undefined. Note that while the L2 cache can be disabled during normal operation, it must be completely flushed of dirty cache lines before being disabled because once disabled it will no longer participate in the coherence protocol (that is, when disabled, the L2 cache is treated as if all its contents are invalid). Likewise, reenabling the L2 cache requires that the L2 cache be completely invalidated before clearing the dis bit since the data in the L2 cache can be stale. In addition, before disabling the L2 cache, the L1 instruction and data caches must be disabled in all strands' ASI_LSU_CONTROL_REG registers, as the L1 caches cannot operate properly when the L2 cache is disabled. The L2 Control register is available at address A9 0000 0000_{16} or B9 0000 0000_{16}. Address bits 8:6 select the cache bank, address bits 31:9 and 5:3 are ignored (that is, the register aliases across the address range).

The L2 Control register format is shown in TABLE 28-33.

TABLE 28-33 L2 Control Register – L2_CONTROL_REG (A9 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
63:21	—	X	RO	*Reserved*
20:15	errorsteer	0	RW	Specifies the physical core (bits 20:18) and strand (bits 17:15) that receives all the L2 errors whose cause can't be linked to a specific virtual processor.
14:3	scrubinterval	0	RW	Interval between scrubbing of adjacent sets in L2 in processor core clocks. In units of 1M cycles.
2	scrubenable	0	RW	If set to 1, enable hardware scrub.
1	dmmode	0	RW	If set to 1, address bits 21 to 18 indicate the replacement way (direct-mapped mode). If set to 0, all L2 ways are enabled under LRU control.
0	dis	1	RW	If set to 1, disable the L2 cache. If set to 0, enable the L2 cache.

28.14.2 L2 Bank Available

The L2 Bank Available register format is shown below.

TABLE 28-34 L2 Bank Available – BANK_AVAIL ($80\ 0000\ 1018_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	rsvd	0	RO	*Reserved*
7:0	avail	FF_{16}	RO	L2 bank available programmed by eFuse.

28.14.3 L2 Bank Enable

The L2 Bank Enable register, described in TABLE 28-35, allows software to enable a legal combination of L2 banks. Initially this register is set identical to BANK_AVAIL. Software must program BANK_ENABLE to a legal combination of banks, as listed in TABLE 28-36.

Note | After programming BANK_ENABLE to a legal combination of banks, SW needs to initiate a warm reset to have the effect take place, i.e have all appropriate banks be enabled or disabled.

Note | The BANK_AVAIL.avail field is **and**ed with value written to BANK_ENABLE, preventing software from being able to enable a bank that is not available.

TABLE 28-35 L2 Bank Enable – BANK_ENABLE ($80\ 0000\ 1020_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	rsvd	0	RO	*Reserved*
7:0	enable	BANK_AVAIL.avail	RW	L2 banks enabled.

TABLE 28-36 Legal Values for BANK_ENABLE.enable.

Value	Description
03_{16}	2-bank mode
$0C_{16}$	2-bank mode
$0F_{16}$	4-bank mode
30_{16}	2-bank mode
33_{16}	4-bank mode
$3C_{16}$	4-bank mode
$C0_{16}$	2-bank mode
$C3_{16}$	4-bank mode

TABLE 28-36 Legal Values for BANK_ENABLE.enable.

CC_{16}	4-bank mode
$F0_{16}$	4-bank mode
FF_{16}	8-bank mode

TABLE 28-37 3 bank pair to 2 bank pair remap of BANK_ENABLE.enable.

BANK_ENABLE.enable	Remapped To
$3F_{16}$	$0F_{16}$
CF_{16}	$0F_{16}$
$F3_{16}$	$F0_{16}$
FC_{16}	$F0_{16}$

TABLE 28-36 specifies the following rules:

1. Two banks enabled: Any one of BA01, BA23, BA45, or BA67.
2. Four banks enabled: Any one of {BA01, BA23},{BA23, BA45}, {BA45, BA67}, {BA67, BA01}, {BA01,BA45}, or {BA23,BA67}.
3. Eight banks enabled: {BA01, BA23, BA45, BA67}.

Note | In case Software programs 3 bank pairs to be enabled in the L2 Bank Enable register in violation of the legal values indicated in TABLE 28-36 , Niagara II hardware remaps it to 2 bank pairs as per TABLE 28-37 on page 440 .

WARNING! | Because the L2 reverse directory can accommodate no more than 1/8 of the L1 cache lines per virtual processor, in the partial bank enabled configurations, the number of SPC cores enabled through CMP registers must be less than or equal to *n*, where *n* is the number of available banks.

Notes | Software needs to ensure PA{39} = 0 in 8-bank, PA{39:38} = 0_2 in 4-bank mode and PA{39:37} = 000_2 in 2-bank mode.

PA{39:38} = 0_2 in 4-bank mode and PA{39:37} = 000_2 in 2-bank mode.

Note Because a pair of L2 banks are directly connected to a single memory port, in a 2-bank configuration, only a single memory port will be usable. In a 4-bank configuration, only a pair of memory ports will be usable.

Note In 2 or 4 bank enabled modes, other than accesses to Bank Enable,Bank Enable Status,Index Hash Enable, Index Hash Enable Status registers which reside in NCU block, Niagara II would have CSR/Diagnostic accesses to all of disabled L2 banks aliased and remapped to enabled banks by SPC.

Implementation Notes In the 2-bank configuration, the L2 bank is selected by PA{6}, with PA{6} = 0 selecting the even bank and PA{6} = 1 selecting the odd bank. In the 4-bank configuration, the L2 index is selected by PA{7:6}. PA{7} selects between the pair of banks enabled, with PA{7} = 0 selecting the lower numerical pair of banks (except for the case where banks BA67 and BA01 are enabled, where PA{7} = 0 selects BA67), and PA{7} = 1 selecting the higher numerical pair of banks (again except for the case where banks BA67 and BA01 are enabled, where PA{7} = 1 selects BA01). PA{6} selects between the even and odd banks of the bank pair selected by PA{7}, with PA{6} = 0 selecting the even bank and PA{6} = 1 selecting the odd bank.

In the 2-bank configuration, the L2 index is formed from PA{15:7} and the tag from PA{37:16} (and PA{39:38} is zeroed out). The usual addr{10,9,5,4} accesses reverse directory CAM panel (1 of 16, each is 32 way), but PA{8:7} is used to access 8 out of the 32 ways in the directory CAM. In the 4-bank configuration, the L2 index is formed from PA{16:8} and the tag from PA{38:17} (and PA{39} is zeroed out). The usual addr{10,9,5,4} accesses reverse directory CAM panel (1 of 16, each is 32 way), but PA{8} accesses 16 out of the 32 ways in the directory CAM.

28.14.4 L2 Bank Enable Status

The Bank Enable Status register shows the status of the hardware signals generated as a result of the BANK_ENABLE register. Software can use this register to determine when the effects of a write to BANK_ENABLE are complete. The preview versions of the status change immediately after BANK_ENABLE is written, while the non-

preview version change only after the next warm reset when all appropriate banks have been enabled or disabled.SW needs to initiate a warm reset to have all appropriate banks have been enabled or disabled.

TABLE 28-38 L2 Bank Enable Status – BANK_ENABLE_STATUS (80 0000 1028_{16})

Bit	Field	Initial Value	R/W	Description
63:13	—	0	RO	*Reserved*
12	pm_preview	0	RO	L2 partial mode enable. Set if BANK_ENABLE is not FF_{16}.
11	ba67_preview	1	RO	Banks 6 and 7 enable. Set if bits 7:6 of BANK_ENABLE are 3_{16}.
10	ba45_preview	1	RO	Banks 4 and 5 enable. Set if bits 5:4 of BANK_ENABLE are 3_{16}.
9	ba23_preview	1	RO	Banks 3 and 2 enable. Set if bits 3:2 of BANK_ENABLE are 3_{16}.
8	ba01_preview	1	RO	Banks 1 and 0 enable. Set if bits 1:0 of BANK_ENABLE are 3_{16}.
7:5	—	0	RO	*Reserved*
4	pm	0	RO	L2 partial mode enabled. Set if 4 or 2 L2 banks are enabled.
3	ba67	1	RO	Banks 6 and 7 enabled. Set if L2 banks 7:6 are enabled.
2	ba45	1	RO	Banks 4 and 5 enabled. Set if L2 banks 5:4 are enabled.
1	ba23	1	RO	Banks 3 and 2 enabled. Set if L2 banks 3:2 are enabled.
0	ba01	1	RO	Banks 1 and 0 enabled. Set if L2 banks 1:0 are enabled.

28.14.5 L2 Index Hash Enable

The L2 has a hashing function available to relieve hot-spotting in certain L2 sets. Hashing is controlled through the L2_IDX_HASH_EN and L2_IDX_HASH_EN_STATUS registers. The hashing enable can be changed only following a warm reset. After a warm reset, the value written into L2_IDX_HASH_EN is sent the L2 and reflected in L2_IDX_HASH_EN_STATUS once the L2 completes hashing setup.

TABLE 28-39 L2 Index Hash Enable – L2_IDX_HASH_EN (80 0000 1030_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	enb_hp	0	RW	If 1, enable L2 index hashing following next warm reset. If 0, disable L2 index hashing following next warm reset.

Implementation Note The hashing function used by OpenSPARC T2 sets the effective PA{17:11} ← {(PA{32:28} **xor** PA{17:13}) **::** (PA{19:18} **xor** PA{12:11})}. This hashing function is unaffected by the BANK_ENABLE_STATUS register settings.

28.14.6 L2 Index Hash Enable Status

TABLE 28-40 L2 Index Hash Enable Status – L2_IDX_HASH_EN_STATUS (80 0000 1038_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	enb_hp	0	RO	Set if L2 index hashing is enabled.

28.14.7 Other L2 Registers

Other L2 Cache registers are defined in other chapters, particularly the error control and status registers are in the Error Handling chapter.

28.15 L2 Cache Flushing

OpenSPARC T2 provides Prefetch [address], 18_{16} (PrefetchICE) for invalidating and coherently committing an L2 cache line to memory. This PrefetchICE is only available when hyperprivileged. Execution of a PrefetchICE instruction while in user or privileged mode is a NOP. TABLE 28-41 lists the address format for the PrefetchICE.

TABLE 28-41 PrefetchICE Address Format

Bit	Field	Description
63:40	—	*Reserved*
39:37	key	Must be 011. Use of any other value places OpenSPARC T2 in an undefined state.
36:22[1]	—	*Reserved*, can be any value.
21:18[2]	way	Selects way in cache set.
17:9[3]	set	Selects cache set in bank.
8:6[4]	bank	Selects cache bank.
5:0	—	Reserved, can be any value.

1. When index hashing is enabled, bits 32:28 will affect the set selection.
2. When index hashing is enabled, bits 19:18 will affect the set selection.
3. When four L2 banks are enabled, set is specified in bits 16:8 and bit 17 may be any value, while when two banks are enabled, set is specified in bits 15:7 and bits 17:16 may be any value.
4. When four L2 banks are enabled, bank is specified in bits 7:6 only, while when two banks are enabled, bank is specified in bit 6 only.

Note The PrefetchICE address specifies which particular L2 cache way, set, and bank are committed to memory. To flush a specific address, software must either issue a PrefetchICE to all 16 possible ways or, if software can guarantee that the address will not be refetched during the flushing code, software can use diagnostic reads to find which way contains the cache address and then issue a PrefetchICE to that specific way. A series of PrefetchICE's to a particular L2 bank can be guaranteed complete if followed by any load to that bank (even diagnostic load). The completion of the load will indicate that the PrefetchICE instructions are complete.

Programming Notes When L2 index hashing is enabled (as specified in L2_IDX_HASH_EN_STATUS), software will need to generate the nonhashed index for use by the PrefetchICE. While bits 32:28 can be set to zero to remove their effect on hashing, bits 19:18 are part of the hash calculation, and the bits 12:11 of the desired set must be **xor**ed with bits 19:18 to generate the bits 12:11 used in the PrefetchICE. For example, if software desired to flush all 16 ways of bank 0, set 0, it would need to generate the following addresses: 60 0000 0000_{16}, 60 0004 0800_{16}, 60 0008 1000_{16}, 60 000C 1800_{16}, 60 0010 0000_{16}, 60 0014 0800_{16}, 60 0018 1000_{16}, 60 001C 1800_{16}, 60 0020 0000_{16}, 60 0024 0800_{16}, 60 0028 1000_{16}, 60 002C 1800_{16}, 60 0030 0000_{16}, 60 0034 0800_{16}, 60 0038 1000_{16}, 60 003C 1800_{16}.

When four or two L2 banks are enabled (as specified in BANK_ENABLE_STATUS), PrefetchICE will shift the set and bank bits (for 4 banks set is 16:8 and bank is 7:6, for 2-banks set is 15:7 and bank is 6). The way will always be specified in bits 21:18 and will not be shifted based on BANK_ENABLE_STATUS.

Note If a PrefetchICE instruction detects a tag parity error it issues a scrub of the tag array index and gets replayed after the scrub. It completes normally if there is no further tag parity error. If the PrefetchICE instruction detects a VUAD correctable error, the PrefetchICE instruction just completes as normal and the VUAD error gets silently corrected (without being logged). Since the VUAD error gets corrected early in the L2 pipeline, the PrefetchICE instruction always gets the correct value of the Dirty bits in its eviction pass and hence completes normally.

Note | A PrefetchICE instruction when used to invalidate a line from Niagara II L2 Cache might cause a tag parity error or data correctable/uncorrectable error to be detected and reported . In case the reporting has been turned off by software prior to the issue of the Prefetch ICE, the error would still be in pending state until the next miss is serviced by the L2 cache. In order not to have this error reported in the future, software should clear this internal error reporting pending state in L2 cache by forcing a miss to that particular L2 bank after the Prefetch ICE . One way to force the miss is to issue a load to the same physical address as the line which got invalidated by Prefetch ICE.

28.16 L2 Cache Diagnostic Access

This section describes the control registers and diagnostic access for the L2 cache.

Diagnostic accesses will functionally work in the midst of other operations, but the results of L2 diagnostic accesses are inherently somewhat indeterminate because the state of the L2 cache is a moving target. Of course, diagnostic writes have the capability of putting OpenSPARC T2 into an inconsistent or illegal state.

TABLE 28-42 shows the breakdown of the L2 address range.

TABLE 28-42 Breakdown of the L2 Diagnostic Address Range

Address Range (8 MSBs of the 40-bit Address)	Assigned to:	Comment
$A0_{16}$–$A3_{16}$/$B0_{16}$–$B3_{16}$	L2 Data	Diagnostic access to the L2 data array.
$A4_{16}$–$A5_{16}$/$B4_{16}$–$B5_{16}$	L2 Tag	Diagnostic access to the L2 tag array.
$A6_{16}$–$A7_{16}$/$B6_{16}$–$B7_{16}$	L2 Tag VUAD	Diagnostic access to the L2 VUAD array.
$A8_{16}$–AF_{16}/$B8_{16}$–BF_{16}	L2 Registers	Error, control, and status registers.

28.16.1 L2 Data Diagnostic Access

Diagnostic access to the L2 data array is done through 64-bit read/writes that access a 32-bit data subblock along with the corresponding 7-bit ecc. The format for addressing the entire L2 cache is shown in TABLE 28-43.

Note | Diagnostic loads of the L2 data do not check the ecc, and thus cannot generate an ECC error.

Programming Note L2 Data diagnostic addresses are not affected by L2 index hashing (as specified in L2_IDX_HASH_EN_STATUS). If index hashing is enabled and software wants to access a line corresponding to a specific PA, software must adjust the L2 Data diagnostic address accordingly.

Programming Note In case L2 is configured in partial bank mode, for a L2 data diagnostic access, the L2 index gets picked up from bits 16:8 in 4 bank mode and 15:7 in 2 bank mode from the L2 data diagnostic address packet which is similar to the way L2 index is constructed for any cacheable access to L2 in partial bank modes. Bits 17:6 should be the same as the original PA in partial bank mode.

TABLE 28-43 Format 6 L2 Data Diagnostic Addressing

Bit	Field	Description
63:40	—	*Reserved*
39:32	select	Must be $A0_{16}$, $A1_{16}$, $A2_{16}$, $A3_{16}$, $B0_{16}$, $B1_{16}$, $B2_{16}$, or $B3_{16}$ to select L2 data diagnostic access.
31:23	—	*Reserved*, can be any value (that is, the data diagnostic access is aliased throughout the A0 – A3/B0–B3 address range).
22	oddeven	Selects 32 bit word from 64 bit word selected by word field.
21:18	way	Selects way in cache set.
17:9	set	Selects cache set in bank.
8:6	bank	Selects cache bank.
5:3	word	Selects 64 bit word in 64 byte cache line.
2:0	—	All zero for 64-bit access.

The data format is shown in TABLE 28-44.

TABLE 28-44 23 L2 Diagnostic Data – L2_DIAG_DATA (A0 0000 0000_{16}) (Count 2, Step 4194304) (Count 393216, Step 8)

Bit	Field	Initial Value	R/W	Description
63:39	—	X	RO	*Reserved*
38:7	data	X	RW	Data.
6:0	ecc	X	RW	ECC value for data.

28.16.2 L2 Tag Diagnostic Access

Diagnostic access to the L2 tag array is done through 64 bit read/writes that access the tag along with the corresponding 6-bit ecc. The format for addressing the entire L2 cache is shown in TABLE 28-45.

Note | Diagnostic loads of the L2 tag do not check the ecc, and thus cannot generate an ECC error.

Programming Note | L2 Tag diagnostic addresses are not affected by either partial L2 banks (as specified in BANK_ENABLE_STATUS) or L2 index hashing (as specified in L2_IDX_HASH_EN_STATUS). If either index hashing is enabled or not all banks are enabled and software wants to access a line corresponding to a specific PA, software must adjust the L2 Tag diagnostic address accordingly. In case L2 is configured in partial bank mode, for a L2 tag diagnostic access, the L2 index gets picked up from 17:9 of the L2 Tag Diagnostic address packet . Hence software should precompute the physical index from the specific PA based on the partial bank configuration (16:8 for 4 bank and 15:7 for 2 bank) and present in on bits 17:9 of the L2 Tag diagnostic packet. However bits 8:6 should be the same as the original PA in partial bank mode.

TABLE 28-45 Format 7 L2 Tag Diagnostic Addressing

Bit	Field	Description
63:40	—	*Reserved*
39:32	select	Must be $A4_{16}$, $A5_{16}$, $B4_{16}$, or $B5_{16}$ to select L2 tag diagnostic access.
31:22	—	*Reserved*, can be any value (i.e., the tag diagnostic access is aliased throughout the A4 – A5/B4 – B5 address range).
21:18	way	Selects way in cache set.
17:9	set	Selects cache set in bank.
8:6	bank	Selects cache bank.
5:3	—	*Reserved*, can be any value.
2:0	—	All zero for 64-bit access.

The data format is shown in TABLE 28-46.

TABLE 28-46 24 L2 Diagnostic Tag – L2_DIAG_TAG (A4-0000-0000$_{16}$) (Count 49152, Step 64)

Bit	Field	Initial Value	R/W	Description
63:30	—	X	RO	*Reserved*
27:6	tag	X	RW	Tag, corresponds to addr{39:18}[1].
5:0	ecc	X	RW	ECC value for tag.

1. With the L2 in 4 bank mode, the tag contains addr{38:17} if index hashing is disabled, and {addr{38:18}, addr{32}^addr{17}} if index hashing is enabled. With the L2 in 2-bank mode, the tag contains addr{37:16} if index hashing is disabled, and {addr{36:18},addr{32:31} **xor** addr{17:16}} if index hashing is enabled.

Note | Restraint must be exercised when using this register to inject errors into the L2 tags. Since a tag correction takes much longer than an injection and since correction causes a retry for an access that discovers an error, repeatedly injecting errors at a high rate can prevent forward progress of another virtual processor accessing a line at that cache index (set). Injecting each error just once (potentially for many individual errors) or ensuring that the injections to the same index are at least 1000 cycles apart are both sufficient to avoid this problem.

28.16.3 L2 VUAD Diagnostic Access

Diagnostic access to the L2 VUAD array is done through a pair of address access ranges. The first accesses the valid and dirty bits for an entire set plus the ECC for those bits across the set via 64-bit read/writes. The format for addressing the entire L2 cache is shown in TABLE 28-47.

Note | Diagnostic loads of the VUAD do not check ecc, and thus cannot generate ECC errors.

Programming Note | L2 VUAD diagnostic addresses are not affected by L2 index hashing (as specified in L2_IDX_HASH_EN_STATUS). If index hashing is enabled software wants to access a line corresponding to a specific PA, software must adjust the L2 VUAD diagnostic address accordingly.

Programming Note In case L2 is configured in partial bank mode, for a L2 VUAD diagnostic access, the L2 physical index gets picked up from bits 16:8 in 4 bank mode and 15:7 in 2 bank mode from the L2 VUAD diagnostic address packet which is similar to the way L2 index is constructed for any cacheable access to L2 in partial bank modes.Bits 17:6 should be the same as the original PA in partial bank mode.

TABLE 28-47 Format 8 L2 VD Diagnostic Addressing

Bit	Field	Description
63:40	—	*Reserved*
39:32	select	Must be $A6_{16}$, $A7_{16}$, $B6_{16}$, or $B7_{16}$ to select L2 VD diagnostic access.
31:23	—	*Reserved*, can be any value (i.e., the VD diagnostic access is aliased throughout the A6–A7/B6–B7 address range where bit 22 is 1).
22	vdsel	Must be set to 1
21:18	—	*Reserved*, can be any value (i.e., the VD diagnostic access is aliased throughout the A6– A7/B6–B7 address range where bit 22 is 1).
17:9	set	Selects cache set in bank.
8:6	bank	Selects cache bank.
5:3	—	*Reserved*, can be any value.
2:0	rsvd4	All zero for 64 bit access

The data format is shown in TABLE 28-48.

TABLE 28-48 25 L2 Diagnostic VD – L2_DIAG_VD (A6 0040 0000_{16}) (count 4096 step 64)

Bit	Field	Initial Value	R/W	Description
63:39	—	X	RO	*Reserved*
38:32	vdecc	X	RW	ECC for all dirty and valid bits.
31:16	valid	X	RW	Valid bits for way 15 down to way 0.
15:0	dirty	X	RW	Dirty bits for way 15 down to way 0.

The second range accesses the AD bits for an entire set via 64 bit read/writes. The format for addressing the entire L2 cache is shown in TABLE 28-49.

TABLE 28-49 Format 9 L2 UA Diagnostic Addressing

Bit	Field	Description
63:40	—	*Reserved*
39:32	select	Must be $A6_{16}$, $A7_{16}$, $B6_{16}$, or $B7_{16}$ to select L2 UA diagnostic access.
31:23	—	*Reserved*, can be any value (i.e., the UA diagnostic access is aliased throughout the A6–A7/B6–B7 address range where bit 22 is 0).

TABLE 28-49 Format 9 L2 UA Diagnostic Addressing *(Continued)*

Bit	Field	Description
22	vdsel	Must be set to 0.
21:18	—	*Reserved*, can be any value (i.e., the UA diagnostic access is aliased throughout the A6–A7/B6–B7 address range where bit 22 is 0).
17:9	set	Selects cache set in bank.
8:6	bank	Selects cache bank.
5:3	—	*Reserved*, can be any value.
2:0	rsvd4	All zero for 64 bit access.

The data format is shown in TABLE 28-50.

TABLE 28-50 26 L2 Diagnostic UA – L2_DIAG_UA (A6 000 00000_{16}) (Count 4096 Step 64)

Bit	Field	Initial Value	R/W	Description
63:39	—	X	RO	*Reserved*
38:32	uaecc	X	RW	ECC for all used and alloc bits.
31:16	used	X	RW	Used bits for way 15 down to way 0.
15:0	alloc	X	RW	Allocated bits for way 15 down to way 0.

28.17 Built-In Self-Test (BIST)

28.17.1 L2 BIST Control

On OpenSPARC T2, each L2 bank has its BIST control coming from the Test Control Unit (TCU). The software-visible BIST state of each L2 bank is also available in TCU. Hence the L2 Control register at address A8 0000 0000_{16}/B8 0000 0000_{16} is reserved and returns a 64-bit zero value when read.

CHAPTER 29

Hardware Debug Support

29.1 SPARC Core Debug Features

Each physical OpenSPARC T2 SPARC core contains several hardware debug features. Some (breakpoints and watchpoints, for example) have already been mentioned in Chapter 28, *Configuration and Diagnostics Support*.

29.1.1 Shadow Scan

Each physical OpenSPARC T2 SPARC core supports the ability to capture a subset of each strand's state for inspection via a shadow scan facility. The shadow scan is invoked by JTAG commands (see Appendix F, *JTAG (IEEE 1149.1) Scan Interface*). Please refer to section *Shadow Scan Chains* on page 556 for Shadow Scan in OpenSPARC T2.

The TCU continually specifies a strand ID to each physical OpenSPARC T2 SPARC core. In response, the physical core atomically captures the state as described in the following table in a scan string. The TCU then accesses the scan string and captures it in a JTAG-visible register for presentation over the JTAG interface.

TABLE 29-1 lists the state that can be gathered from each running virtual processor.

TABLE 29-1 SPARC Shadow Scan State

Data Bits	Field	Remarks
117:72	VA{47:2}	Virtual address of last instruction executed by that strand
71	ibe	HPSTATE.ibe
70	cle	PSTATE.cle
69	tle	PSTATE.tle
68	tct	PSTATE.tct

TABLE 29-1 SPARC Shadow Scan State

Data Bits	Field	Remarks
67	hpriv	HPSTATE.hpriv
66	red	HPSTATE.red
65	pef	PSTATE.pef
64	am	PSTATE.am
63	priv	PSTATE.priv
62	ie	PSTATE.ie
61	tlz	HPSTATE.tlz
60:58	TL{2:0}	TL
57:12	TPC{47:2}	TPC for the last trap
11:3	TT{8:0}	TT for the last trap
2:0	TL_FOR_TT{2:0}	TL for the last trap

VA is updated when an instruction is executed. When the TCU changes the strand ID that controls which strand's state is captured, the VA field can capture spurious values. Between the time that the TCU updates the strand ID and the time that the new strand executes an instruction, the VA field reflects either the VA of the last instruction that executed on the previous strand ID or a spurious value captured during the strand ID transition. After the first instruction executes on the new strand, the VA field correctly reflects the VA for the new strand. There is no valid bit or strand ID captured in the shadow scan to distinguish when the VA has updated for the new strand.

HPSTATE, PSTATE, and TL are updated dynamically. Thus, they correctly reflect updates after a trap, a DONE, a RETRY, or a software write to `%tl`. These fields also update immediately after the TCU changes the strand ID.

TPC, TT, and TL_FOR_TT, however, are updated only when the strand takes a trap. They are not updated for DONE, RETRY, or software writes to `%tl`. For example, if the processor traps from TL = 0 to TL = 1 to TL = 2 and then uses DONE and/or RETRY to get back to TL = 0, shadow scan will still reflect TT{2}, TPC{2}, and TL_FOR_TT will still be 2. Similarly, if the processor traps out to TL = 2 and then software writes TL to 1 or 0, shadow scan will still show TT{2}, TPC{2}, and TL_FOR_TT will still be 2. Simiarly, after TCU changes the strand ID, these fields reflect state for the previous strand ID until the new strand takes a trap. There is no valid bit or strand ID captured in the shadow scan to distinguish when these fields have updated for the new strand.

On the JTAG bus, bit 117 appears first, followed by bit 116, sequentially down to and including bit 0.

If multiple traps occur after the state was atomically captured into the shadow scan string, but while the shadow scan string is being scanned, only the state belonging to the last trap remains to be (potentially) captured on the next capture point. Due to the length of time to perform a shadow scan relative to the time between traps, sampling via shadow scan can miss several traps.

All eight core shadow scans are scanned serially as one chain, with strand 0 closest to TDI and strand 7 closest to TDO. Any strand marked unavailable in the CMP STRAND_AVAILABLE register will not be included when scanned via TDI to TDO. The shadow scan chain for a given strand is placed in that strand's second scan chain during ATPG test mode; they are accessible otherwise only via JTAG shadow scan instructions (that is, not during JTAG serial scan).

29.1.2 SPARC Debug Event Control Register

There is one debug control register per physical OpenSPARC T2 SPARC core (ASI_DECR). Each event type field in the ASI_DECR contains two bits that encode the type of response for that event as shown in TABLE 29-2.

TABLE 29-2 ASI_DECR Debug Event Response Type Encoding

DECR Event Response Type Encoding	Response If Debug Event Occurs
00	Debug event disabled.
01	Soft stop.
10	Hard stop.
11	Pulse TRIGOUT/ Watchpoint.

All strands of a physical OpenSPARC T2 SPARC core share a hyperprivileged, read/write, Debug Event Control register located at ASI 45_{16}, VA 8_{16}. The DECR controls the debug event response (hard stop or soft stop or watchpoint) for an associated core debug event if that event occurs.

The format of the ASI_DECR register is described in TABLE 29-3.

TABLE 29-3 ASI_DECR Register (ASI 45_{16}, VA 8_{16})

Bit	Field	Initial Value	R/W	Description
63:62	iwa_de	0 (preserved across warm and debug reset)	RW	Instruction breakpoint match debug event enable.
61:60	iva_de	0 (preserved across warm and debug reset)	RW	Instruction virtual address match debug event enable.
59:58	dva_de	0 (preserved across warm and debug reset)	RW	Data virtual address match debug event enable.
57:56	dpa_de	0 (preserved across warm and debug reset)	RW	Data physical address match debug event enable.
55:54	tct_de	0 (preserved across warm and debug reset)	RW	Trap on Control Transfer debug event enable.
53:52	pe_de	0 (preserved across warm and debug reset)	RW	Precise error event (an event which would be recorded in the I-SFSR or D-SFSR) debug event enable.
51:50	de_de	0 (preserved across warm and debug reset)	RW	Disrupting error event (an event which would be recorded in the DESR) debug event enable.
49:48	df_de	0 (preserved across warm and debug reset)	RW	Deferred error event (an event which would be recorded in the DFESR) debug event enable.
47:46	pm_de	0 (preserved across warm and debug reset)	RW	Performance monitor event which causes a performance counter to wrap debug event enable.
45:0	—	0	RO	*Reserved*

29.1.3 Disable Overlap and Single-Stepping Modes

Each core can disable overlap of instruction execution within all threads. In this mode, each thread issues one instruction, waits for the instruction to commit and any associated memory operations to be globally observed, then fetches and executes the next instruction. This mode essentially disables pipelining for all of a physical core's strands.

Additionally, each core has the capability of executing one instruction from each enabled strand. This is referred to as single-step.

These two features are controlled through JTAG only by the TAP_DOSS_ENABLE, TAP_DOSS_MODE, TAP_SS_REQUEST, TAP_DOSS_STATUS, TAP_CS_MODE, TAP_CS_STATUS JTAG commands described in TABLE F-1 on page 545.

29.1.4 ASI_RST_VEC_MASK

All physical cores share a hyperprivileged, read/write ASI_RST_VEC_MASK register located as ASI 45_{16}, VA 18_{16}. Reserved bits read as zero and are ignored on write. The contents of this register are preserved across warm reset. In normal operation,

resets trap to the *RSTVADDR* **or** (TT ← 5) (FF FFFF FFF0 $0000_{16}x0$), which maps to ROM. To enhance repeatability, OpenSPARC T2 can direct resets to RAM instead. To redirect resets to RAM, hyperprivileged software can set the vec_mask bit of the ASI_RST_VEC_MASK register.

TABLE 29-4 ASI_RST_VEC_MASK Register (ASI 45_{16}, VA 18_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	vec_mask	0 (preserved across warm and debug reset)	RW	If 1, set *RSTV* to 0000 0000 0000 0000_{16}. If 0 set *RSTV* to FFFF FFFF F000 0000_{16}.

29.2 SOC Debug Features

29.2.1 SOC Debug Event Control Register

There is one debug control register for SOC (SOC_DECR). SOC_DECR controls the debug response type (hard stop or watchpoint) for an associated SOC debug event if that event occur. Each event type field in SOC_DECR contains two bits that encode the type of response for that event as shown in the following table.

TABLE 29-5 SOC_DECR Debug Event Response Type Encoding

DECR Event Response Type Encoding	Response If Debug Event Occurs
00	Debug event disabled
01	*Reserved*
10	Hard-stop
11	Pulse TRIGOUT/Watchpoint

The format of the SOC_DECR is described in the following table.

TABLE 29-6 SOC_DECR Register – SOC_DECR ($86\ 0000\ 0010_{16}$)

Bit	Field	Initial Value	R/W	Description
63:22	—	0	RO	*Reserved*
21:20	se_de	0 (preserved across warm and debug reset)	RW	SOC (SII, SIO, NCU, DMU) error debug event enable.
19:18	me_de	0 (preserved across warm and debug reset)	RW	MCU error debug event enable.
17:16	l2e_de	0 (preserved across warm and debug reset)	RW	L2 error debug event enable.
15:14	l2b7_de	0 (preserved across warm and debug reset)	RW	L2 PA match bank 7 debug event enable.
13:12	l2b6_de	0 (preserved across warm and debug reset)	RW	L2 PA match bank 6 debug event enable.
11:10	l2b5_de	0 (preserved across warm and debug reset)	RW	L2 PA match bank 5 debug event enable.
9:8	l2b4_de	0 (preserved across warm and debug reset)	RW	L2 PA match bank 4 debug event enable.
7:6	l2b3_de	0 (preserved across warm and debug reset)	RW	L2 PA match bank 3 debug event enable.
5:4	L2B2_DE	0 (preserved across warm and debug reset)	RW	L2 PA match bank 2 debug event enable.
3:2	L2B1_DE	0 (preserved across warm and debug reset)	RW	L2 PA match bank 1 debug event enable.
1:0	L2B0_DE	0 (preserved across warm and debug reset)	RW	L2 PA match bank 0 debug event enable.

29.2.2 L2 Cache Debug Features

29.2.2.1 L2 Address Mask and Compare Registers

Each L2 Cache bank has a pair of registers per bank to control debug based on PA matching. The L2_MASK_REG controls which bits are compared against the values set in the L2_COMP_REG. A debug event is asserted if data and L2_MASK_REG = L2_COMP_REG and data is valid.

Since there are eight L2 cache banks in OpenSPARC T2, there are eight debug events based on PA matching. Debug response for these events are controlled by SOC_DECR{15:0}.

TABLE 29-7 L2 Address Mask Register – L2_MASK_REG (AF 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
63:52	—	0	RO	*Reserved*
51:48	ttype	0 (preserved across warm and debug reset)	RW	ttype{3:0} mask.
47:46	—1	0	RO	*Reserved*
45:40	vcid	0 (preserved across warm and debug reset)	RW	Virtual processor ID mask.
39:34	—	0	RO	*Reserved* (Physical Address{39:34} ignored.)
33:2	addr	0 (preserved across warm and debug reset)	RW	Physical Address{33:2} mask.
1:0	—3	0	RO	*Reserved* (Physical Address{1:0} ignored.)

TABLE 29-8 L2 Address Compare Register – L2_COMP_REG (BF 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
63:52	—	0	RO	*Reserved*
51:48	ttype	0 (preserved across warm and debug reset)	RW	ttype{3:0} compare value.
47:46	—1	0	RO	*Reserved*
45:40	vcid	0 (preserved across warm and debug reset)	RW	Virtual processor ID compare value.

TABLE 29-8 L2 Address Compare Register – L2_COMP_REG (BF 0000 0000_{16}) (Count 8 Step 64)

Bit	Field	Initial Value	R/W	Description
39:34	—	0	RO	*Reserved* (Physical Address{39:34} ignored.)
33:2	addr	0 (preserved across warm and debug reset)	RW	Physical Address{33:2} compare value.
1:0	—3	0	RO	*Reserved* (Physical Address{1:0} ignored.)

29.2.2.2 L2 Shadow Scan

Shadow scan for L2 Error registers is controlled via JTAG. The contents to be captured in the shadow scan are as shown in TABLE 29-9. Refer also to *Shadow Scan Chains* on page 556 for L2 shadow scan.

TABLE 29-9 L2 Shadow Scan State

Data Bits	Remarks
141:84	L2 Error Status register.
83:36	Notdata error register.
35:0	FE/UE/CE Error Address register.

All eight L2 Tag shadow scan contents are captured at the same time, and are available at TDO with L2T7 first and L2T0 last . JTAG instructions to support L2 Tag shadow scan are shown in TABLE D-1.

On the JTAG bus, bits appear in the order :
15:0,31:16,47:32,63:48,79:64,95:80,111:96,127:112,141:128.

29.2.3 Debug Event Trigger Enables

The registers described in this section contain the masks for generating debug events in SOC_DECR.

29.2.3.1 DRAM Debug Event Trigger Enable Register

Each DRAM controller has a register that contains the debug event trigger enable for DRAM controller related debug events.

TABLE 29-10 shows the format of the DRAM Debug Trigger Enable register.

TABLE 29-10 DRAM Debug Event Trigger Enable Register – DRAM_DBG_TRG_EN_REG ($84\ 0000\ 0230_{16}$) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:5	—0	0_{16}	RO	*Reserved*
4	dtm_mask1	0_{16}	RW	If set to 1, mask off CRC data for FBD Channel 1 going to Debug bus. Reset only on POR.
3	dtm_mask0	0_{16}	RW	If set to 1, mask off CRC data for FBD Channel 0 going to Debug bus. Reset only on POR.
2	dbg_trig_en	0_{16}	RW	Trigger enable for DRAM errors. dbg_en is cleared when triggered. Reset only on POR.
1	mask_err	0_{16} (preserved across WMR and DBR)	RW	If set to 1, all LFSR mismatches on ALERT frame patterns coming in from the AMB are masked.
0	kp_lnk_up	0_{16} (preserved across WMR and DBR)	RW	If set to 1, MCU keeps sending out sync pulses on the southbound links during the entire duration of the debug/warm reset, thereby keeping the links enabled during the duration of the debug/warm reset. Clearing this bit to 0 after the debug/warm reset takes the MCU's FBDIMM interface state machine to L0 state, where it gets ready to dispatch new read/write requests from SPC's/SOC to the DIMMs.

29.2.3.2 NCU Debug Event Trigger Enable Register

The NCU block has a register that contains the debug event trigger enable for NCU-related debug events.

The format of the NCU Debug Event Trigger Enable register is shown in TABLE 29-11.

TABLE 29-11 NCU Debug Event Trigger Enable Register – NCU_DBG_TRG_EN_REG ($80\ 0000\ 4000_{16}$)

Bit	Field	Initial Value	RW	Description
63:1	—	0	RO	*Reserved*
0	dbg_trig_en	0_{16}	RW	Trigger enable for SOC Error Status register errors.

29.2.3.3 L2 Debug Event Trigger Enable Register

The L2 Debug Event Trigger Enable is described in *L2 Error Enable Register* on page 282.

29.3 TCU Debug Support

The TCU handles debug events requests from the SPARC virtual processors directly or from the SOC. The response to these requests is to stop the clocks (hard or soft) or pass the watchpoint signal to the I/O pins. A set of registers is provided in the TCU to assist in control of these responses to debug event requests.

TCU is not designed to handle burst CSR read requests from SPARC virtual processors, that is, a CSR read request cannot be followed immediately by another CSR read request, otherwise the second one may be dropped and no read data will be returned and the thread issuing the second request may hang. Users should program the second CSR read request after the data for the first one has returned. In the case of multiple SPARC threads accessing TCU CSRs, some mechanism (such as a semaphore lock) should be used to guarantee only one thread accesses any TCU CSR register at a given time.

29.3.1 Action in Response to a Soft-Stop Event

If the DECR bits for a particular event are configured for a soft-stop (set to 01_2), and that event occurs, the following sequence of operations results. The OpenSPARC T2 core or SOC sends a soft stop request to the TCU. The TCU lowers the strand_running inputs for all strands. The strand stops issuing instructions and waits for all core activity to quiesce. "Quiesce" means that all in-flight instructions have completed (or taken exceptions), and all memory references issued by the core have been globally observed. Each strand lowers STRAND_RUNNING_STATUS as it quiesces. Once all strands of a physical core quiesce, the TCU subsequently stops the OpenSPARC T2 core's clocks.

The cycle when the stop occurs is a function of the value of the TCU cycle counter (refer to *TCU Cycle Counter Register* on page 462) as well as the transmission delay from the core to the TCU and from the TCU to the clock network in the core. If the TCU cycle counter is non-zero when the core generates a soft-stop request, the TCU will decrement the cycle counter until it reaches 0. When it reaches 0, the TCU will stop the core's clocks (note that it may take several cycles before the processor clocks stop after the counter reaches 0 due to the propagation delay from the TCU to the core clock network).

29.3.2 Action in Response to a Hard-Stop Event

If the DECR bits for an event are set to 10_2 and that event occurs, the OpenSPARC T2 core requests the TCU to stop the clocks as soon as the TCU cycle counter reaches 0. TCU does not wait for SPC to quiesce before stopping the clocks to the SPCs on a hard stop request.

Note | A hard stop request from any SPC or SOC would have TCU stopping the clocks to the entire chip.

29.3.3 Action in Response to an External Hard Stop

TRIGIN pin when asserted from the system will have TCU initiate a hard stop of OpenSPARC T2 based on the values programmed in the TCU Clock CLK Stop Delay register and TCU Clock Domain Stop register. Please refer to TABLE F-15 on page 555 for description of TCU Clock CLK Stop Delay register.

29.3.4 TCU Debug Event Counter Register

The 32-bit TCU Debug Even Counter register must be zero before the cycle counter is enabled. If it is non-zero, then each SPC debug event request received at the TCU will decrement it by 1, and each SOC debug event request received at the TCU will decrement it by 4; when zero is reached, the cycle counter will begin decrementing with the next debug event request. No differentiation is made regarding debug event requests, so it is up to the user to ensure that only one type of debug event is enabled when using the debug event counter. Debug event requests consist of soft stop, hard stop, and watchpoint requests from SPC, and hard stop, watchpoint requests from SOC blocks. These debug events have been described in ASI_DECR and SOC_DECR registers.

The Debug Event Counter is only recognized when TCU_DEBUG_CONTROL_REG.enable = 0. When TCU_DEBUG_CONTROL_REG.enable = 1, the debug event counter is disabled. The debug event counter is accessed with JTAG instruction TAP_DE_COUNT; default value upon reset is zero. Refer to *TCU Debug Control Register* on page 462 for the description of the TCU Debug Control register.

The format of the TCU Debug Event Counter register is shown in TABLE 29-12.

TABLE 29-12 TCU Debug Event Counter Register – TCU_DEBUG_EVENT_COUNTER_REG ($85\ 0000\ 0011 8_{16}$)

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RO	*Reserved*
31:0	counter	0	RW	Debug event counter.

29.3.5 TCU Cycle Counter Register

The 64-bit TCU Cycle Counter register counts off of OpenSPARC T2 core clock and can delay the response to a debug event. For example, if the TCU receives a hard-stop request, the cycle counter will begin counting down with each CMP clock cycle, and when it reaches zero, the hard stop will be performed. All debug event requests from the SPARC cores or a hard-stop request from SOC logic will be delayed by the cycle counter.

These actions are only valid when TCU_DEBUG_CONTROL_REG.enable = 0. For behavior when TCU_DEBUG_CONTROL_REG.enable = 1, see the next section. The cycle counter is loaded with JTAG instruction TAP_CYCLE_COUNT; default value on reset is zero.

The format of the TCU Cycle Counter register is shown in TABLE 29-13.

TABLE 29-13 TCU Cycle Counter Register – TCU_CYCLE_COUNTER_REG ($85\ 0000\ 0100_{16}$)

Bit	Field	Initial Value	R/W	Description
63:0	counter	0	RW	Cycle counter.

29.3.6 TCU Debug Control Register

The TCU has a 4-bit register to control responses to debug events, the TCU Debug Control Register (DCR). When bit 2 of TCU DCR is 0, the cycle counter and debug event counters perform as described above.

When bit 2 of the TCU DCR is set to 1, the lower 32 bits of the cycle counter are treated as a reset counter. In this mode, the reset counter is decremented with each core clock cycle after the power-on reset (POR) sequence ends. Once zero is reached, a watchpoint, a hard clock stop, or a clock stretch can be performed, or the upper 32 bits of the cycle counter can then be used. In this mode (bit 2 of TCU DCR = 1) the debug event counter will be ignored.

Note | A hard stop request from any SPC or SOC would have TCU stopping the clocks to the entire chip.

The behavior of the debug event and cycle counters is determined by the values in the TCU DCR as specified in TABLE 29-14. The TCU DCR is loaded with JTAG instruction TAP_TCU_DCR; default value on reset is 0.

The format of the TCU Debug Control Register is shown in TABLE 29-14.

TABLE 29-14 TCU Debug Control Register – TCU_DEBUG_CONTROL_REG ($85\ 0000\ 0108_{16}$)

Bit	Field	Initial Value	R/W	Description
63:4	—	0	RO	*Reserved*
3	soft_stop	0_{16}	RW	Enables all strands to soft-stop upon a debug event.
2	enable	0_{16}	RW	Trigger enable for the action field.
1:0	action	0_{16}	RW	Refer to Description in TABLE 29-15, below.

TABLE 29-15 TCU Debug Actions

soft_stop{3}	enable{2}	action{1:0}	Description
0/1	0	xx	Debug event and cycle counter recognize SPC debug events.
x	1	00	Watchpoint pulsed.
x	1	01	Hard stop and watchpoint pulsed.
x	1	10	Clock stretch and watchpoint pulsed.
x	1	11	Clock stretch and watchpoint, followed by hard stop and watchpoint.

The soft_stop bit 3 when set will cause TCU to soft-stop all SPCs when any SPC requests a soft stop. It is only active when bit 2 is 0. The following actions are valid when bit 2 of the TCU DCR is set to 1:

- **Watchpoint.** If the TCU DCR bits 2:0 are set to 100, then a single pulse of an external chip pin (TRIGOUT) will occur when the reset counter reaches zero. The upper 32 bits of the cycle counter are ignored.
- **Hard stop.** A hard clock stop will be performed if the TCU DCR bits 2:0 are set to 101, as specified in *Action in Response to a Hard-Stop Event* on page 461, and a watchpoint pulse generated, when the reset counter reaches zero. The upper 32-bits of the cycle counter are ignored.
- **Clock Stretch.** If the TCU DCR bits 2:0 are set to 110, then a clock-stretch signal will be pulsed out of the TCU when the reset counter reaches zero, and a watchpoint pulse will also be generated. The upper 32 bits of the cycle counter are ignored.

- **Clock Stretch then Hard Stop.** If the TCU DCR bits 2:0 are set to 111, then when the reset counter reaches zero, a clock stretch will be triggered and a watchpoint pulse will also be generated. Then the upper 32 bits of the cycle counter will be allowed to count down to trigger a clock hard stop and a second watchpoint will also be generated.

29.3.7 SW/JTAG Trigger Output Register

The format of the SW/JTAG Trigger Output register is shown in TABLE 29-16.

TABLE 29-16 SW/JTAG Trigger Output Register – TCU_TRIGOUT_REG ($85\ 0000\ 0110_{16}$)

Bit	Field	Initial Value	R/W	Description
63:1	—	0	RO	*Reserved*
0	trigout	0_{16}	RW	Pulses TRIGOUT when written to 1; reset automatically after being written.

29.4 Debug Port Support

OpenSPARC T2 has a 166-pin-wide debug port that is used as an observability vehicle to promote repeatability, tester characterization, chip debug, and general SPC and SOC debug. The debug port can be enabled through software and JTAG access to the Debug Port Configuration register, described in *Debug Port Configuration Register* on page 468. The debug port can be configured into any one of six observability modes. The following are the different observability modes of the debug port based on bits 3:1 of the Debug Port Configuration register:

- **000_2: SOC observability mode.** OpenSPARC T2 reset state (Reset State Machine Output), MCU,SII → L2,L2 → SIO signals to help chip debug (sent out on 159 pins).
- **001_2: Tester charac/SPC debug mode.** {SPC_id,thread_id} on per L2 bank basis and SPC instruction commit status on per SPC basis, sent out on 160 pins.
- **010_2: Repeatability mode.** SII and NCU inputs from DMU on debug port double-pumped on 166 pins.
- **011_2: Core and SOC debug.** SII and NCU inputs from DMU and SPC instruction commit status on per SPC basis.
- **100_2:**
- **101_2:**
- **110_2–111_2:** Reserved for future use.

The following sections describe these modes in detail.

29.4.1 SOC Observability Mode

This mode will be used to capture a variety of critical SOC signals which will be helpful to debug OpenSPARC T2. The following is the breakup of the signals in this mode.

- 5-bit encoded state for reset state machine (has 20 states) from RST block to MIO block to monitor reset state on the tester and LA. Sent out at sys_clk frequency from reset block in OpenSPARC T2 (feedthrough in MIO) on 5 pins.
- Each MCU will send the following *new* signals to DBG block, which will be useful to debug MCU hangs/scheduler issues or MCU error handling issues on both FBDIMM channel errors and ECC errors. These signals will all be synchronized by MCU to the iol2clk domain and sent to DBG block. This leads to a total of 21 wires per MCU. Since there are 4 MCUs, this will lead to a total of 84 wires to DBG block from all MCUs together. DBG block will drive this information out on 84/2 = 42 pins of the debug port @ io2xclk.
  - mcu_dbg_rd_req_in_0{3:0} – Read request from L2 bank 0 to MCU (id + valid).
  - mcu_dbg_rd_req_in_1{3:0} – Read request from L2 bank 1 to MCU (id + valid).
  - mcu_dbg_rd_request_out{4:0} – Read ACK from MCU to L2 bank 0 or 1 (id + valid + dest_l2_bank).
  - mcu_dbg_wr_req_in_0 – Write request valid from L2 bank 0.
  - mcu_dbg_wr_req_in_1 – Write request valid from L2 bank 1.
  - mcu_dbg_wr_req_out{1:0} – 0, 1, 2, 3 writes completed at DRAM indication. (MCU dispatches up to a maximum of three writes on any cycle on two FBDIMM channels; then samples information coming FBDIMM channels to see if there were any errors. If no errors were reported, MCU interprets this as all writes completed.)
  - mcu_dbg_mecc_err – MCU has detected an MECC error on a L2 read or scrub.
  - mcu_dbg_secc_err – MCU has detected a SECC error on a L2 read or scrub.
  - mcu_dbg_fbd_err – MCU has detected a FBDIMM channel error.
  - mcu_dbg_err_mode – FBDIMM interface logic has gone into error handling mode. This bit stays on until error handling complete.
- SII and SIO will send the following signals to DBG block, which will be useful to debug L2 hang cases (SII sent DMA request to L2, L2 never sends an ACK or data return back):
  - sii_dbg_l2t[0:7]_req{1:0} – Req type encoded on 2 bits from sii to each l2t bank (00 – no request; 01 – RDD; 10 – WRI; 11 – WR8).
  - l2t[0:7]_dbg_sii_iq_dequeue – L2 dequeue from IQ.
  - l2t[0:7]_dbg_sii_wib_dequeue – L2 dequeue from IOWB.
  - l2b[0:7]_dbg_sio_ctag_vld – response valid from L2 to SIO.
  - l2b[0:7]_dbg_sio_ack_type – Read or write ACK from L2 to SIO

- l2b[0:7]_dbg_sio_ack_test – ACK to DMU

Which leads to a total of (7 × 8) = 56 wires for all L2 banks together @ 1.4 GHz to DBG block. DBG block will drive this information out on 56 × 2 = 112 pins of the debug port @ io2xclk.

Thus, the total number of debug pins that will be used up in the SOC observability mode will be 42 + 112 = 154.

29.4.2 Tester Characterization / SPC Debug Mode

The signals that will be observed on the debug port in this mode will be used for general SPC debug and tester characterization of multithreaded diagnostics and also for SPC speed binning on the tester. Each SPC will have four signals driven to the DBG block, and each L2 bank will have six signals driven to the DBG block. All these signals will be at CMP clk frequency, that is, 1.4 GHz nominal. Eight virtual processors and eight L2 banks will lead to a total of (4 + 6) x 8 = 80 signals at 1.4 GHz driven to the DBG block. Since the debug port will drive the signals out @ io2xclk, the DBG block will sample two consecutive cycles of these 80-bit wires and drive out 160 signals @ io2xclk to the debug pins for LA sampling.

For each SPC, these four wires are chosen as follows. Since each core has two thread groups, we have the following encoding per thread group, using 2 bits/thread group:

- 00_2 – Instruction noncommitted
- 01_2 – Control Transfer instruction committed in pipe
- 10_2 – Integer or FPU instruction committed in pipe
- 11_2 – Load/Store instruction committed in pipe

That is, every instruction committed per cycle in each thread group will be visible. for each L2 bank, the six wires are vcid{5:0} {spc_id{2:0}, thread_id{2:0}} of each crossbar packet to that bank on every cycle.

The combination of these two groups of signals will be adequate to keep track of execution of instructions in both single and multithreaded diagnostics on the tester and also could be useful for SPC speed binning on the tester.

29.4.3 Repeatability Mode

In this mode, a total of 353 signals (in iol2clk domain) will be routed to DBG block (from NIU and DMU). From , 166 wires will get driven at io2xclk to the debug pins. These signals capture both inbound DMA and PIO returns in OpenSPARC T2 to SII and NCU and will be used as bus traces for checkpoint/replay scheme in OpenSPARC T2. These 353 signals and rate conversion to debug port frequency are listed below.

1. dmu_ncu_wrack_vld;
2. dmu_ncu_wrack_tag{3:0};
3. dmu_ncu_stall; // total 6 bits @ iol2clk = 3 pins @ io2xclk (DDR)
4. dmu_ncu_vld;
5. dmu_ncu_data{31:0}; // 33 bits get driven over four clocks. Eight clocks minimum before next set of four clocks, so total of 132 bits to be emptied over 12 iol2clks, that is, 66 bits DDR over 12 clocks, that is, 6 pins @ io2xclk (DDR).
6. niu_ncu_stall;
7. niu_ncu_vld;
8. niu_ncu_data{31:0}; // 34 bits @ iol2clk = 17 pins @ io2xclk (DDR)
9. dmu_sii_hdr_vld;
10. dmu_sii_reqbypass;
11. dmu_sii_datareq;
12. dmu_sii_datareq16;
13. dmu_sii_data{127:0};
14. dmu_sii_be{15:0}; // 148 bits @ iol2clk = 74 pins @ io2xclk (DDR)
15. niu_sii_hdr_vld;
16. niu_sii_reqbypass;
17. niu_sii_datareq;
18. niu_sii_data{127:0};
19. niu_sio_dq; // 132 bits @ iol2clk = 66 pins @ io2xclk (DDR)

Total = 66 + 74 + 17 + 3 + 6 = 166 pins @ io2xclk (DDR)

29.4.4 Core and SOC Debug mode.

SII and NCU inputs from DMU and SPC instruction commit status on per SPC basis:

1. dmu_ncu_wrack_vld;
2. dmu_ncu_wrack_tag{3:0};
3. dmu_ncu_stall; // total 6 bits @ iol2clk = 3 pins @ io2xclk(DDR)
4. dmu_ncu_vld;
5. dmu_ncu_data{31:0}; // 33 bits get driven over four clocks. Eight clocks minimum before next set of four clks, so a total of 132 bits will be emptied over 12 iol2clks, that is, 66 bits DDR over 12 clocks, that is, 6 pins @ io2xclk (DDR)
6. dmu_sii_hdr_vld;
7. dmu_sii_reqbypass;
8. dmu_sii_datareq;
9. dmu_sii_datareq16;
10. dmu_sii_data{127:0};
11. dmu_sii_be{15:0}; // 148 bits @ iol2clk = 74 pins @ io2xclk (DDR)

Each SPC will have four signals driven to the DBG block at core clk frequency. Since there are eight cores, this will lead to a total of (4) x 8 = 32 signals @ l2 clk driven to DBG block. Since the debug port will drive the signals out @ io2xclk, the DBG block will sample two consecutive cycles of these 32-bit wires and drive out 64 signals @ io2xclk to the debug pins for LA sampling.

For each SPC, these four wires are chosen as follows.

Since each core has 2 thread groups, we have the following encoding per thread group using 2 bits/thread group:

- 00_2 – Instruction non committed
- 01_2 – Control Transfer instruction committed in pipe
- 10_2 – Integer or FPU instruction committed in pipe
- 11_2 – Load/Store instruction committed in pipe

Thus total number of pins used to drive out debug info in this mode = (3 + 6 + 74 + 64) = 147.

29.4.5 Debug Port Configuration Register

The Debug Port Configuration register enables the debug port in one of six modes.

TABLE 29-17 shows the format of the Debug Port Configuration register.

TABLE 29-17 Debug Port Configuration – DEBUG_PORT_CONFIG ($86\ 0000\ 0000_{16}$)

Bit	Field	Initial Value	R/W	Description
63:62	imp_ctrl	0 (preserved on WMR/ DBR)	RW	MIO driver impedance control: 11 – Strong driver 10 – Nominal driver 01 – Weak driver 00 – Low power driver
61:10	—	0	RO	*Reserved*
9:5	—	0 (preserved on WMR/ DBR)		*Reserved*

TABLE 29-17 Debug Port Configuration – DEBUG_PORT_CONFIG ($86\ 0000\ 0000_{16}$) *(Continued)*

Bit	Field	Initial Value	R/W	Description
4	debug_train	0 (preserved on WMR/ DBR)	RW	When set to 1, enables training for Debug port in modes 000, 001, 010, and 011.
3:1	debug_conf	0 (preserved on WMR/ DBR)	RW	Debug Port Configuration: 000 – SOC observability 001 – Tester characterization/SPC debug 010 – Repeatability 011 – CORE_SOC debug 100 – 101 – 110–111 – *Reserved*
0	debug_en	0 (preserved on WMR/ DBR)	RW	When set to 1, enables Debug port output drivers.

29.4.6 Debug Port Training Sequence

The data coming out from OpenSPARC T2 on the debug port needs to be aligned properly in the Logic Analyzer connected to the debug port after cancelling out signal to clock and signal to signal skews. To support this, OpenSPARC T2 would provide training sequence in all the debug port modes. For all modes, this sequence will be a repetitive pattern of three 1's, followed by a single 0. This asymmetrical pattern will ease the alignment and deskewing of the data bits in the LA in case the skew for some bits is as large as one cycle.

29.4.7 IO Quiesce Control Register

The IO Quiesce Control register format is shown in TABLE 29-18.

TABLE 29-18 IO Quiesce Control Register – IO_QUIESCE_CONTROL ($86\ 0000\ 0008_{16}$)

Bit	Field	Initial Value	R/W	Description
63:4	—	X	RO	*Reserved*
3	—	X		*Reserved*

TABLE 29-18 IO Quiesce Control Register – IO_QUIESCE_CONTROL ($86\ 0000\ 0008_{16}$)

Bit	Field	Initial Value	R/W	Description
2	dmu_stall_done	X	RO	Set to 1 by hardware when DMU stall completes in hardware. Cleared by hardware when DMU_STALL cleared from 1 to 0 by SW.
1	—	0 (preserved on WMR/ DBR)		*Reserved*
0	dmu_stall	0 (preserved on WMR/ DBR)	RW	When set to 1, causes DMU traffic to stall. When cleared to 0 from 1, causes DMU traffic to resume.

29.5 Repeatability Support

To effectively run processor tests in the post-silicon phase with or without the presence of I/O (on OpenSPARC T2's XAUI and PCI_EX interfaces) and to debug them, we need to have a high level of repeatability within OpenSPARC T2's synchronous clock domains. These include the core clock domain (covers SPARC cores, crossbar, L2's, portions of SII, SIO, NCU), the DRAM domain (covers MCU logic before SerDes), and the I/O clock domain (rest of SII, SIO, NCU).

This will allow us to run a group of tests many times, with slightly different starting parameters (for example, SPARC threads starting at slightly different times or with different cache initialization) that shouldn't affect the outcome, looking for failing corner cases. When a failing case is found, the test and the particular seed parameters will be used to simulate the test in the pre-silicon environment, to see what caused the failure.

The overall approach involves very close interaction between some debug software (part of Hypervisor software) and OpenSPARC T2 chip hardware. This is commonly known as the checkpoint/replay mechanism, where the debug software will periodically put the synchronous portion of the chip (as described before) into an `idle` state (idle all threads other than one, and also stall I/O into the synchronous domain) at what are called checkpoints. Once the synchronous portion of the chip is put into this `idle` state, the debug software will dump all software-visible state of the machine to memory, and then initiate a debug reset of OpenSPARC T2.

The debug software will initiate by writing a 1 to the dbr_gen bit in reset_gen register.

29.5.1 Debug Reset

The debug reset is a flavor of l in OpenSPARC T2 which is identical to the functionality of warm reset other than it will not reset the MCU FBDIMM links.

Notes

Program counter continues to advance for some time after the debug reset request from the RST block till the reset takes effect on the SPC blocks.

PCI_EX device has a hardcoded timeout of 50 msec, so the duration of the back pressure exerted by OpenSPARC T2 during checkpoint and debug reset should be well below 50 msec so as not to starve any access from the device by more than that amount (to avoid PCI_EX errors and system panics). This means that checkpoint snap, debug reset, FBDIMM initialization and resumption of I/O traffic to synchronous domain of OpenSPARC T2 should finish well before 50 msec.

OpenSPARC T2, like previous Sun processors, keeps a fair amount of architected state unchanged for warm reset. Also contents of arrays (TLBs, L1/L2 caches, etc.) are unchanged. Please refer to Table 11-13 in section 11.8 for a list of OpenSPARC T2 software-visible state that will be lost over debug reset and will need to be retrieved after debug reset. So before invoking debug reset, software should copy the software-visible state that loses value over debug reset to memory, and retrieve it back from memory after the reset.

The duration of the debug reset is small enough (in the range of 40 µsecs), so to address the data integrity in DRAM during the debug reset, OpenSPARC T2 will either (1) address it through self-refresh during the debug reset or (2) autorefresh in small intervals before going to debug reset and then doing some in small intervals after coming out of debug reset. Besides, OpenSPARC T2 would keep the FDIMM links up during debug reset and keep issuing sync pulses to the AMBs.

After debug reset, the reset vector will be fetched from memory from a different location (000000020_{16}) from a regular reset. This is because the boot code for a debug reset will be different from a regular reset. The boot code will do several things at the beginning including program the Memory refresh registers, reinstate the software-visible state to the state before reset for those states that lose value over debug reset), before enabling all threads to start executing.

In normal operation POR and warm reset both trap to the $RSTVADDR \mid 20_{16}$ (FFFF FFFF F000 0020_{16}), which maps to ROM. To enhance repeatability, OpenSPARC T2 will have the capability of directing POR, WMR, or DBR to RAM. To POR or WMR or DBR from RAM at location (0 0000 0020_{16}), hyperprivileged software can set the `ASI_WMR_VEC_MASK` register.

The idea is that by capturing the software-visible state of OpenSPARC T2 (in the synchronous domain of the chip) on the last checkpoint before the failure and by initializing the synchronous portion of OpenSPARC T2 to known state, we can create a common starting point between silicon and the synchronous portion of the chip in the pre-silicon environment. Then by running the same code sequence on the SPARC core from the last checkpoint to the failure point and capturing the I/O traffic to the SII, NCU inputs (synchronous I/O interface of OpenSPARC T2: debug port mode 000_2) from DMU on the debug port lossless and feeding it back to the same nodes in the pre-silicon environment, we can create the event sequence in pre-silicon environment leading to the failure.

Note | For checkpoint/replay, we do not need to observe the FBDIMM interface on the debug port. This is because once the links are trained, data will always come back to the MCU data return FIFO in a fixed latency from the time of issue of the request. After link training, MCU logic will record this latency (in terms of MCU clocks) in the MCU Channel Read Latency register. So the debug software can probe this value and feed that same latency to the equivalent point in the pre-silicon environment and thereby achieve cycle accuracy with respect to silicon without having to probe the FBDIMM interface.

Thus, checkpoint/replay approach intrudes on the state of the machine in the context of the tests or applications running on the chip, in that it periodically halts all threads and I/O and takes the machine to `reset` state. This might change the timing of events to cause the bug to manifest itself later than usual, but eventually it will, with millions of cycles of instructions executed in between checkpoints. And when it does, it can be recreated in pre-silicon environment.

29.5.2 Keeping FBDIMM Links Up During Debug Reset

If debug reset did reset the whole MCU, the FBDIMM links will have to be retrained after reset deassertion, and this will change the FBDIMM data round-trip latency for subsequent requests until the next debug reset. Debug software can live with this by reading the MCU Channel Read Latency register after every debug reset, or MCU needs to keep sending sync pulses during the debug reset.

To support the latter, MCU will keep a small amount of logic running during warm/debug reset while the rest of it gets reset through the flush mechanism. This logic will consist of (1) logic to keep the links enabled and generate sync pulses in a fixed repetitive manner under software control and (2) logic to keep incrementing the read pointers of the northbound MCU FIFOs and two synchronizers per FIFO (this way, during debug/warm reset, the read and the write pointers constantly increment and are always offset by 2: delay through the two synchronizers).

Also each MCU would support two new CSR bits, located in the DRAM Debug Event Trigger Enable register for software to control this feature:

- **kp_lnk_up.**

  When written to $_2$,

  - Keeps the southbound links enabled during the duration of the debug reset to send out the sync pulses.
  - Selects the output of the sync pulse gen logic in the new MCU control module to generate sync pulses.

  When written to 0,

  - Selects the output of the regular sync pulse gen logic in MCU.
  - Clears the counter for the regular sync pulse gen logic in MCU.
  - Takes the MCU FBDIMM interface state machine to L0 state, where it is ready to dispatch new read/write requests to the DIMMs.

- **mask_err.**

  When written to 1,

  - Makes MCU mask all the errors it normally detects on LFSR mismatches on ALERT frame patterns coming in from AMB.

  Cleared by MCU hardware 4K cycles after reset when the LFSRs are realigned by the MCU.

Note | Both kp_lnk_up and mask_err bits are protected on warm reset/ debug reset.

Thus, the interaction between software and hardware to achieve determinism on FMDIMM interface after debug reset is as follows:

1. After making sure no transactions are pending in MCU, software sets kp_lnk_up and mask_err just before initiating debug reset.
2. Debug reset happens. The entire MCU gets reset other than the control logic module, which has its clock running keeping the sync pulses going and the FIFO read pointer incrementing every cycle.
3. Debug reset finishes. The MCU FBDIMM interface state machine comes up in DISABLED state. Sync ACKs keep coming, but since the mask_err bit is set, no errors are flagged. MCU logic counts 4K cycles after reset and realigns the LFSRs and clears the mask_err bit.
4. After a certain time T1 (but always fixed from the deassertion of debug reset), software writes a 0 to the kp_lnk_up bit. This clears the sync pulse gen counter, takes the FBDIMM interface state machine to L0 state, and selects the sync pulse gen counter output to generate the sync pulses.
5. After a time T2 from the point where software wrote kp_lnk_up with 0, the first fetch is issued on the southbound link. T2 should be the same all the time.

Note | The FBDIMM links would be retrained after every warm reset.

29.5.3 I/O Quiescing in OpenSPARC T2 During Checkpoint

An inherent requirement for checkpoint/replay in OpenSPARC T2 is to stall I/O to the synchronous domain of the chip (SII and NCU inputs). This is part of the effort to get the chip to a quiescent state on every checkpoint before dumping software-visible state and asserting debug reset to get the synchronous portion of the chip to a known state.

This I/O quiescing will get implemented in OpenSPARC T2 under software control by having the DBG module contain a CSR bit (dmu_stall) in OpenSPARC T2 I/O Quiesce Control register (refer to *IO Quiesce Control Register* on page 469) which software can set to 1's by writing a 1 to them. Once these bits are set, the debug module in OpenSPARC T2 will assert a signal, called dbg_dmu_stall to DMU.

On seeing the assertion of these signals, DMU should suspend all transactions to SII and NCU at any convenient point and send back niu_dbg_stall_ack and dmu_dbg_stall_ack to the debug module after they have received all pending ACKs and data returns from SIU and NCU. At the point at which these two ACKs are sent to the DBG block, NCU and DMU → SII, NCU interfaces will be considered as having quiesced. This applies to interrupts also. DMU should not send any interrupt requests to NCU or SII after having sent the ACKs. On sampling dmu_dbg_stall_ack signals, DBG block will set dmu_stall_done bits in the two I/O Quiesce Control registers. The debug software that will have been polling these status bits will then see that both bits are set and will proceed to dump software-visible state of machine to memory and then initiate a debug reset.

Note that even during the time this interface is quiesced, the Xaui and PCI_EX interface SerDes links are active and running.

After debug reset, the reset code will clear the dmu_stall CSR bits in DBG block, which will cause DBG block to assert a signal to DMU called dbg_dmu_resume. On receiving these "resume" signals, DMU will unquiesce their respective interfaces with NCU and continue issuing transactions to NCU.

29.6 Clock/PLL Observability

OpenSPARC T2 has two pins called PLL_CHAR_OUT{1:0} dedicated for PLL/clock observability output.

APPENDIX **A**

Programming Guidelines

A.1 Multithreading

In OpenSPARC T2, each physical core contains eight strands. The strands are divided into two thread groups, with strands 0–3 occupying one thread group and strands 4–7 occupying the other.

Within a thread group, among the available strands, the least recently picked strand is selected for execution every cycle. Thus, up to two instructions can be picked each cycle.

Since each physical core has only one load/store unit and one floating-point and graphics unit, only one load/store or FGU instruction may be picked each cycle. One thread group can issue a load/store instruction while the other thread group issues an FGU instruction. Arbitrating between the two thread groups is done with a least-recently-picked mechanism, to ensure fairness.

Since context switching is built into the OpenSPARC T2 pipeline (via the D and P stages), strands are switched each cycle with no pipeline stall penalty (except when resource collisions occur, such as when both thread groups require the load/store unit or the FGU).

In normal operation, OpenSPARC T2 speculates that most control-transfer instructions will be "not taken" and that loads hit in the L1 data cache. An enable bit, accessible to hyperprivileged software, controls whether OpenSPARC T2 speculates on these instructions or not (see `ASI_LSU_CONTROL_REG` description in *`ASI_LSU_CONTROL_REG`* on page 413).

The following instructions change a strand from available to unavailable until hardware determines that their input/execution requirements can be satisfied, assuming speculation is enabled:

- CALL, DONE, RETRY, JMPL
- LDFSR, LDXFSR, STFSR, STXFSR
- All WRPR, WR, WRHPR

- All RDPR, RD, RDHPR
- SAVE(D), RESTORE(D), RETURN, FLUSHW (all register-window management)
- All MUL and DIV
- MULX, UMUL, SMUL, POPC
- MEMBAR#, FLUSH
- FCMP
- All memory operations to/from alternate space
- All atomic (load-store) operations
- PREFETCH

If speculation is not enabled, the following instruction types also change a strand from available to unavailable until hardware determines that their execution requirements can be satisfied:

- All control transfer instructions
- All loads

A.2 Instruction Latency

TABLE A-1 lists the minimum single-strand instruction latencies for OpenSPARC T2. When multiple strands are executing, some or much of the additional latency for multicycle instructions will be overlapped with execution of the additional strands.

TABLE A-1 OpenSPARC T2 Instruction Latencies *(1 of 8)*

Opcode	Description	Latency	Notes
ADD (ADDcc)	Add (and modify condition codes)	1	
ADDC (ADDCcc)	Add with carry (and modify condition codes)	1	
ALIGNADDRESS	Calculate address for misaligned data access	1	
ALIGNADDRESSL	Calculate address for misaligned data access (little-endian)	1	
ALLCLEAN	Mark all windows as clean	25	
AND (ANDcc)	Logical **and** (and modify condition codes)	1	
ANDN (ANDNcc)	Logical **and not** (and modify condition codes)	1	
ARRAY{8,16,32}	3-D address to blocked byte address conversion	6	
Bicc	Branch on integer condition codes	1 not-taken, 6 taken	
BMASK	Write the GSR.mask field	25	

TABLE A-1 OpenSPARC T2 Instruction Latencies *(2 of 8)*

Opcode	Description	Latency	Notes
BPcc	Branch on integer condition codes with prediction	1 not-taken, 6 taken	
BPr	Branch on contents of integer register with prediction	1 not-taken, 6 taken	
BSHUFFLE	Permute bytes as specified by the GSR.mask field	6	
CALL	Call and link	6	
CASA	Compare and swap word in alternate space	20-30	Done in L2 cache
CASXA	Compare and swap doubleword in alternate space	20-30	Done in L2 cache
DONE	Return from trap	6	
EDGE{8,16,32}{L}{N}	Edge boundary processing {little-endian} {non-condition-code altering}	6	
FABS(s,d)	Floating-point absolute value	6	
FADD(s,d)	Floating-point add	6	
FALIGNDATA	Perform data alignment for misaligned data	6	
FANDNOT1{s}	Negated src1 **and** src2 (single precision)	6	
FANDNOT2{s}	src1 **and** negated src2 (single precision)	6	
FAND{s}	Logical **and** (single precision)	6	
FBPfcc	Branch on floating-point condition codes with prediction	1 not-taken, 6 taken	
FBfcc	Branch on floating-point condition codes	1 not-taken, 6 taken	
FCMP(s,d)	Floating-point compare	6	
FCMPE(s,d)	Floating-point compare (exception if unordered)	6	
FCMPEQ{16,32}	Four 16-bit / two 32-bit compare: set integer dest if src1 = src2	6	
FCMPGT{16,32}	Four 16-bit / two 32-bit compare: set integer dest if src1 > src2	6	
FCMPLE{16,32}	Four 16-bit / two 32-bit compare: set integer dest if src1 ≤ src2	6	
FCMPNE{16,32}	Four 16-bit / two 32-bit compare: set integer dest if src1 ≠ src2	6	
FDIV(s,d)	Floating-point divide	19 SP, 33 DP	

TABLE A-1 OpenSPARC T2 Instruction Latencies *(3 of 8)*

Opcode	Description	Latency	Notes
FEXPAND	Four 8-bit to 16-bit expand	6	
FiTO(s,d)	Convert integer to floating-point	6	
FLUSH	Flush instruction memory	variable	
FLUSHW	Flush register windows	25	
FMOV(s,d)	Floating-point move	6	
FMOV(s,d)cc	Move floating-point register if condition is satisfied	6	
FMOV(s,d)R	Move floating-point register if integer register contents satisfy condition	6	
FMUL(s,d)	Floating-point multiply	6	
FMUL8SUx16	Signed upper 8- x 16-bit partitioned product of corresponding components	6	
FMUL8ULx16	Unsigned lower 8- x 16-bit partitioned product of corresponding components	6	
FMUL8x16	8- x 16-bit partitioned product of corresponding components	6	
FMUL8x16AL	Signed lower 8- x 16-bit lower α partitioned product of 4 components	6	
FMUL8x16AU	Signed upper 8- x 16-bit lower α partitioned product of 4 components	6	
FMULD8SUx16	Signed upper 8- x 16-bit multiply → 32-bit partitioned product of components	6	
FMULD8ULx16	Unsigned lower 8- x 16-bit multiply → 32-bit partitioned product of components	6	
FNAND{s}	Logical **nand** (single precision)	6	
FNEG(s,d)	Floating-point negate	6	
FNOR{s}	Logical **nor** (single precision)	6	
FNOT1{s}	Negate (1's complement) src1 (single precision)	6	
FNOT2{s}	Negate (1's complement) src2 (single precision)	6	
FONE{s}	One fill (single precision)	6	
FORNOT1{s}	Negated src1 **or** src2 (single precision)	6	
FORNOT2{s}	src1 **or** negated src2 (single precision)	6	
FOR{s}	Logical **or** (single precision)	6	
FPACKFIX	Two 32-bit to 16-bit fixed pack	6	
FPACK{16,32}	Four 16-bit/two 32-bit pixel pack	6	

TABLE A-1 OpenSPARC T2 Instruction Latencies *(4 of 8)*

Opcode	Description	Latency	Notes
FPADD{16,32}{s}	Four 16-bit/two 32-bit partitioned add (single precision)	6	
FPMERGE	Two 32-bit to 64-bit fixed merge	6	
FPSUB{16,32}{s}	Four 16-bit/two 32-bit partitioned subtract (single precision)	6	
FsMULd	Floating-point multiply single to double	6	
FSQRT(s,d)	Floating-point square root	19 SP, 33 DP	
FSRC1{s}	Copy src1 (single precision)	6	
FSRC2{s}	Copy src2 (single precision)	6	
F(s,d)TO(s,d)	Convert between floating-point formats	6	
F(s,d)TOi	Convert floating point to integer	6	
F(s,d)TOx	Convert floating point to 64-bit integer	6	
FSUB(s,d)	Floating-point subtract	6	
FXNOR{s}	Logical **xnor** (single precision)	6	
FXOR{s}	Logical **xor** (single precision)	6	
FxTO(s,d)	Convert 64-bit integer to floating-point	6	
FZERO{s}	Zero fill (single precision)	6	
ILLTRAP	Illegal instruction		
INVALW	Mark all windows as CANSAVE	6	
JMPL	Jump and link	6	
LDBLOCKF	64-byte block load	32	
LDD	Load doubleword	3	
LDDA	Load doubleword from alternate space	variable	
LDDF	Load double floating-point	3	
LDDFA	Load double floating-point from alternate space	variable	
LDF	Load floating-point	3	
LDFA	Load floating-point from alternate space	variable	
LDFSR	Load floating-point state register lower	1-8	pre-sync to previous FGU op from that thread
LDSB	Load signed byte	3	

TABLE A-1 OpenSPARC T2 Instruction Latencies *(5 of 8)*

Opcode	Description	Latency	Notes
LDSBA	Load signed byte from alternate space	variable	
LDSH	Load signed halfword	3	
LDSHA	Load signed halfword from alternate space	variable	
LDSTUB	Load-store unsigned byte	3	
LDSTUBA	Load-store unsigned byte in alternate space	variable	
LDSW	Load signed word	3	
LDSWA	Load signed word from alternate space	variable	
LDUB	Load unsigned byte	3	
LDUBA	Load unsigned byte from alternate space	variable	
LDUH	Load unsigned halfword	3	
LDUHA	Load unsigned halfword from alternate space	variable	
LDUW	Load unsigned word	3	
LDUWA	Load unsigned word from alternate space	variable	
LDX	Load extended	3	
LDXA	Load extended from alternate space	variable	
LDXFSR	Load extended floating-point state register	1-8	pre-sync to previous FGU op from that thread
MEMBAR	Memory barrier	variable	
MOVcc	Move integer register if condition is satisfied	1	
MOVr	Move integer register on contents of integer register	1	
MULScc	Multiply step (and modify condition codes)		
MULX	Multiply 64-bit integers	5	
NOP	No operation	1	
NORMALW	Mark other windows as restorable	25	
OR (ORcc)	Inclusive-**or** (and modify condition codes)	1	
ORN (ORNcc)	Inclusive-**or not** (and modify condition codes)	1	
OTHERW	Mark restorable windows as other	6	
PDIST	Distance between eight 8-bit components	6	1 per 2 cycles

TABLE A-1 OpenSPARC T2 Instruction Latencies *(6 of 8)*

Opcode	Description	Latency	Notes
POPC	Population count	5	
PREFETCH	Prefetch data	variable	>6
PREFETCHA	Prefetch data from alternate space	variable	>6
RDASI	Read ASI register	variable	
RDASR	Read ancillary state register	variable	
RDCCR	Read condition codes register	variable	
RDFPRS	Read floating-point registers state register	variable	
RDHPR	Read hyperprivileged register	variable	
RDPC	Read program counter	variable	
RDPR	Read privileged register	variable	
RDTICK	Read TICK register	variable	
RDY	Read Y register	variable	
RESTORE	Restore caller's window	6	
RESTORED	Window has been restored	6	
RETRY	Return from trap and retry		
RETURN	Return	7	
SAVE	Save caller's window	6	
SAVED	Window has been saved	6	
SDIV (SDIVcc)	32-bit signed integer divide (and modify condition codes)	12-41	
SDIVX	64-bit signed integer divide	12-41	
SETHI	Set high 22 bits of low word of integer register	1	
SIAM	Set interval arithmetic mode	6	
SIR	Software-initiated reset		
SLL	Shift left logical	1	
SLLX	Shift left logical, extended	1	
SMUL (SMULcc)	Signed integer multiply (and modify condition codes)	5	
SRA	Shift right arithmetic	1	
SRAX	Shift right arithmetic, extended	1	
SRL	Shift right logical	1	
SRLX	Shift right logical, extended	1	

TABLE A-1 OpenSPARC T2 Instruction Latencies *(7 of 8)*

Opcode	Description	Latency	Notes
STB	Store byte	1	
STBA	Store byte into alternate space		
STBAR	Store barrier	variable	
STBLOCKF	64-byte block store	16	Assuming store buffer empty when STBLOCKF decodes
STD	Store doubleword	1	
STDA	Store doubleword into alternate space		
STDF	Store double floating-point	1	
STDFA	Store double floating-point into alternate space		
STF	Store floating-point	1	
STFA	Store floating-point into alternate space		
STFSR	Store floating-point state register	1-8	pre-sync to previous FGU op from that thread
STH	Store halfword	1	
STHA	Store halfword into alternate space		
STPARTIALF	Eight 8-bit/4 16-bit/2 32-bit partial stores	1	
STW	Store word	1	
STWA	Store word into alternate space		
STX	Store extended	1	
STXA	Store extended into alternate space	variable	
STXFSR	Store extended floating-point state register		
SUB (SUBcc)	Subtract (and modify condition codes)	1	
SUBC (SUBCcc)	Subtract with carry (and modify condition codes)	1	
SWAP	Swap integer register with memory	20-30	Done in L2 cache
SWAPA	Swap integer register with memory in alternate space	20-30	Done in L2 cache

TABLE A-1 OpenSPARC T2 Instruction Latencies *(8 of 8)*

Opcode	Description	Latency	Notes
TADDcc (TADDccTV)	Tagged add and modify condition codes (trap on overflow)	1	If no trap, 6 if trap
TSUBcc (TSUBccTV)	Tagged subtract and modify condition codes (trap on overflow)	1	If no trap, 6 if trap
Tcc	Trap on integer condition codes (with 8-bit sw_trap_number, if bit 7 is set, trap to hyperprivileged)	1	If no trap, 6 if trap
UDIV (UDIVcc)	Unsigned integer divide (and modify condition codes)	12-41	
UDIVX	64-bit unsigned integer divide	12-41	
UMUL (UMULcc)	Unsigned integer multiply (and modify condition codes)	5	
WRASI	Write ASI register		
WRASR	Write ancillary state register	variable	
WRCCR	Write condition codes register	25	
WRFPRS	Write floating-point registers state register	25	
WRHPR	Write hyperprivileged register	15	
WRPR	Write privileged register	variable	
WRY	Write Y register	25	
XNOR (XNORcc)	Exclusive-**nor** (and modify condition codes)	1	
XOR (XORcc)	Exclusive-**or** (and modify condition codes)	1	

APPENDIX **B**

IEEE 754 Floating-Point Support

OpenSPARC T2 conforms to the SPARC V9 Appendix B (IEEE Std 754-1985 Requirements for SPARC-V9) recommendation.

Note | OpenSPARC T2 detects tininess before rounding.

B.1 Special Operand Handling

The OpenSPARC T2 FGU follows the UltraSPARC I/UltraSPARC II handling of special operands instead of that used in OpenSPARC T1. While OpenSPARC T1 provides full hardware support for subnormal operands and results, OpenSPARC T2 generates an *fp_exception_other* exception (with FSR.ftt = unfinished_FPop) in some cases. In addition, OpenSPARC T2 implements a nonstandard floating-point mode (enabled when FSR.ns = 1), whereas OpenSPARC T1 does not.

The FGU generates +∞, −∞, +largest number, −smallest number (depending on round mode) for overflow cases for multiply, divide, and add operations.

For higher-to-lower precision conversion instructions FdTOs:

- Overflow, underflow, and inexact exceptions can be raised
- Overflow is treated the same way as an unrounded add result: Depending on the round mode, we will either generate the properly signed infinity or largest number.
- Underflow for subnormal or gross underflow results: (see *Subnormal Handling* on page 494).

For conversion to integer instructions {F<s | d>TOi, F<s | d>TOx}: OpenSPARC T2 follows *The SPARC Architecture Manual-Version 9* (appendix B.5, pg 246.

For NaN's: OpenSPARC T2 follows *The SPARC Architecture Manual-Version 9* appendix B.2 (particularly Table 27) and B.5, pg 244-246.

- Please note that Appendix B applies to those instructions listed in IEEE 754 section 5: "All conforming implementations of this standard shall provide operations to add, subtract, multiply, divide, extract the sqrt, find the remainder, round to integer in fp format, convert between different fp formats, convert between fp and integer formats, convert binary<->decimal, and compare. Whether copying without change of format is considered an operation is an implementation option."
- The instructions involving copying/moving of fp data (FMOV, FABS, and FNEG) will follow earlier UltraSPARC implementations by doing the appropriate sign bit transformation but will not cause an invalid exception nor do a rs2 = SNaN to rd = QNaN transformation.
- Following UltraSPARC II/UltraSPARC III implementations, all Fpops as defined in V9 will update cexc. All other instructions will leave cexc unchanged.

The remainder of this section gives examples of special cases to be aware of that could generate various exceptions.

B.1.1 Infinity Arithmetic

Let "num" be defined as unsigned in the following tables.

B.1.1.1 One Infinity Operand Arithmetic

- Do not generate exceptions.

TABLE B-1 One-Infinity Operations That Do Not Generate Exceptions

Cases
+∞ plus +num = +∞
+∞ plus -num = +∞
-∞ plus +num = –∞
-∞ plus -num = –∞
+∞ minus +num = +∞
+∞ minus -num = +∞
-∞ minus +num = –∞
-∞ minus –num = –∞
+∞ multiplied by +num = +∞
+∞ multiplied by -num = –∞
–∞ multiplied by +num = –∞
–∞ multiplied by –num = +∞
+∞ divided by +num = +∞
+∞ divided by –num = –∞
–∞ divided by +num = –∞
–∞ divided by –num = +∞

TABLE B-1 One-Infinity Operations That Do Not Generate Exceptions *(Continued)*

Cases
+num divided by +∞ = +0 +num divided by -∞ = –0 –num divided by +∞ = –0 –num divided by –∞ = +0
FsTOd, FdTOs (+∞) = +∞ FsTOd, FdTOs (–∞) = –∞
sqrt(+∞) = +∞
+∞ divided by +0 = +∞ +∞ divided by –0 = –∞ –∞ divided by +0 = –∞ –∞ divided by –0 = +∞
Any arithmetic operation involving infinity as one operand and a QNaN as the other operand: V9, B.2.2, Table 27. (± ∞) OPERATOR (QNaN2) = QNaN2 (QNaN1) OPERATOR (± ∞) = QNaN1
Compares when other operand is not a NaN treat infinity just like a regular number: +∞ = +∞, +∞ > anything else; –∞ = –∞, –∞ < anything else. Effects following instructions: V9 fp compares (rs1 and/or rs2 could be ± ∞): * FCMPE * FCMP
Compares when other operand is a QNaN, SPARC V9 A.13, B.2.1; fcc value = unordered = 11_2 FCMP(s,d) (± ∞) with (QNaN2) – no invalid exception FCMP(s,d) (QNaN1) with (±∞) – no invalid exception

- Could generate exceptions

TABLE B-2 One Infinity Operations That Could Generate Exceptions

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, Appendix B.5[1]	IEEE_754 7.1	
F<s | d>TOi (+∞) = invalid	IEEE_754 invalid	$2^{31}-1$
F<s | d>TOx (+∞) = invalid	IEEE_754 invalid	$2^{63}-1$
F<s | d>TOi (–∞) = invalid	IEEE_754 invalid	-2^{31}
F<s | d>TOx (–∞) = invalid	IEEE_754 invalid	-2^{63}
V9, B.2.2	IEEE_754 7.1	(No NaN operand result)
sqrt(–∞) = invalid	IEEE_754 invalid	QNaN
+∞ multiplied by +0 = invalid	IEEE_754 invalid	QNaN
+∞ multiplied by –0 = invalid	IEEE_754 invalid	QNaN
–∞ multiplied by +0 = invalid	IEEE_754 invalid	QNaN
–∞ multiplied by –0 = invalid	IEEE_754 invalid	QNaN
V9, B.2.2, Table 27[2] Any arithmetic operation involving infinity as one operand and SNaN as the other operand except copying/moving data	IEEE_754 7.1	(One operand, a SNaN)
(± ∞) OPERATOR (SNaN2)	IEEE_754 invalid	QSNaN2
(SNaN1) OPERATOR (± ∞)	IEEE_754 invalid	QSNaN1
V9, A.13, B.2.1[2] Any compare operation involving infinity as one operand and a SNaN as the other operand:	IEEE_754 7.1	
FCMP<s | d> (± ∞) with (SNaN2)	IEEE_754 invalid	fcc value = unordered = 11_2
FCMP<s | d> (SNaN1) with (± ∞)	IEEE_754 invalid	fcc value = unordered = 11_2
FCMPE<s | d> (± ∞) with (SNaN2)	IEEE_754 invalid	fcc value = unordered = 11_2
FCMPE<s | d> (SNaN1) with (± ∞)	IEEE_754 invalid	fcc value = unordered = 11_2
V9, A.13[2] Any compare & generate exception operation involving infinity as 1 operand and a QNaN as the other operand:	IEEE_754 7.1	
FCMPE<s | d> (± ∞) with (QNaN2)	IEEE_754 invalid	fcc value = unordered = $2'b11_2$
FCMPE<s | d> (QNaN1) with (± ∞)	IEEE_754 invalid	fcc value = unordered = $2'b11_2$

1. Similar invalid exceptions also included in SPARC V9 B.5 are generated when the source operand is a NaN(QNaN or SNaN) or a resulting number that cannot fit in 32-bit[64-bit] integer format: (large positive argument $\geq 2^{31}[2^{63}]$ or large negative argument $\leq -(2^{31}+1)[-(2^{63}+1)]$

2. Note that in the IEEE 754 standard, infinity is an exact number; so this exception could also applies to non-infinity operands as well. Also note that the invalid exception and SNaN to QNaN transformation does not apply to copying/moving fpops (FMOV, FABS, FNEG).

B.1.1.2 Two Infinity Operand Arithmetic

- Do not generate exceptions

TABLE B-3 Two Infinity Operations That Do Not Generate Exceptions

Cases
+∞ plus +∞ = +∞ –∞ plus –∞ = –∞
+∞ minus –∞ = +∞ –∞ minus +∞ = –∞
+∞ multiplied by +∞ = +∞ +∞ multiplied by –∞ = –∞ –∞ multiplied by +∞ = –∞ –∞ multiplied by –∞ = +∞
Compares treat infinity just like a regular number: +∞ = +∞, +∞ > anything else; –∞ = –∞, –∞ < anything else. Affects following instructions: V9 fp compares (rs1 and/or rs2 could be ± ∞): * FCMPE * FCMP

■ Could generate exceptions

TABLE B-4 Two Infinity Operations That Generate Exceptions

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, B.2.2	IEEE_754 7.1	(No NaN operand result)
+∞ plus –∞ = invalid	IEEE_754 invalid	QNaN
–∞ plus +∞ = invalid	IEEE_754 invalid	QNaN
+∞ minus +∞ = invalid	IEEE_754 invalid	QNaN
–∞ minus –∞ = invalid	IEEE_754 invalid	QNaN
V9, B.2.2	IEEE_754 7.1	(No NaN operand result)
+∞ divided by +∞ = invalid	IEEE_754 invalid	QNaN
+∞ divided by –∞ = invalid	IEEE_754 invalid	QNaN
–∞ divided by +∞ = invalid	IEEE_754 invalid	QNaN
–∞ divided by –∞ = invalid	IEEE_754 invalid	QNaN

B.1.2 Zero Arithmetic

TABLE B-5 Zero Arithmetic Operations that generate exceptions

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, B.2.2 & 5.1.7.10.4	IEEE_754 7.1	(No NaN operand
+0 divided by +0 = invalid	IEEE_754 invalid	result)
+0 divided by –0 = invalid	IEEE_754 invalid	QNaN
–0 divided by +0 = invalid	IEEE_754 invalid	QNaN
–0 divided by –0 = invalid	IEEE_754 invalid	QNaN
		QNaN
V9. 5.1.7.10.4	IEEE_754 7.2	
+num divided by +0 = divide by zero	IEEE_754 div_by_zero	+∞
+num divided by –0 = divide by zero	IEEE_754 div_by_zero	–∞
–num divided by +0 = divide by zero	IEEE_754 div_by_zero	–∞
–num divided by –0 = divide by zero	IEEE_754 div_by_zero	+∞
V9, B.2.2 Table 27[1] Any arithmetic operation involving zero as 1 operand and a SNaN as the other operand except copying/moving data	IEEE_754 7.1	(One operand, a SNaN)
(± 0) OPERATOR (SNaN2)	IEEE_754 invalid	QSNaN2
(SNaN1) OPERATOR (± 0)	IEEE_754 invalid	QSNaN1

1.In this context, 0 is again another exact number; so this exception could also applies to non-zero operands as well. Also note that the invalid exception and SNaN to QNaN transformation does not apply to Copying/moving data instructions (FMOV, FABS, FNEG)

TABLE B-6 Interesting Zero Arithmetic Sign Result Case

Cases
+0 plus –0 = +0 for all round modes except round to –infinity where the result is –0.
sqrt (–0) = –0

B.1.3 NaN Arithmetic

- Do not generate exceptions

TABLE B-7 NaN Arithmetic Operations That Do Not Generate Exceptions

Cases
V9, B.2.1: Fp convert to wider NaN transformation FsTOd (QNaN2) = QNaN2 widened FsTOd (7FD1 0000_{16}) = 7FFA 2000 0000 0000_{16} FsTOd (FFD1 0000_{16}) = FFFA 2000 0000 0000_{16}
V9, B.2.1: Fp convert to narrower NaN transformation FdTOs (QNaN2) = QNaN2 narrowed FdTOs (7FFA 2000 0000 0000_{16}) = 7FD 1000_{16} FdTOs (FFFA 2000 0000 0000_{16}) = FFD 1000_{16}
V9, B.2.2 Table 27 Any noncompare arithmetic operations. Result takes sign of QNaN pass through operand. [Note this rule is applicable to sqrt(QNaN2) = QNaN2 as well]. (± num) OPERATOR (QNaN2) = QNaN2 (QNaN1) OPERATOR (± num) = QNaN1 (QNaN1) OPERATOR (QNaN2) = QNaN2

- Could Generate Exceptions

TABLE B-8 NaN Arithmetic Operations That Could Generate Exceptions

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, B.2.1: Fp convert to wider NaN transformation FsTOd (SNaN2) = QSNaN2 widened	IEEE_754 7.1	
FsTOd (7F91 0000_{16}) = 7FFA 2000 0000 0000_{16} FsTOd (FF91 0000_{16}) = FFFA 2000 0000 0000_{16}	IEEE_754 invalid	QSNaN2 widened
V9, B.2.1: Fp convert to narrower NaN transformation FdTOs (SNaN2) = QSNaN2 narrowed	IEEE_754 7.1	
FdTOs (7FF2 2000 0000 0000_{16}) = 7FD 1000_{16} FdTOs (FFF2 2000 0000 0000_{16}) = FFD 1000_{16}	IEEE_754 invalid	QSNaN2 narrowed

TABLE B-8 NaN Arithmetic Operations That Could Generate Exceptions *(Continued)*

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, B.2.2 Table 27 Any noncompare arithmetic operations except copying/moving (FMOV, FABS, FNEG) [Note this rule applies to sqrt(SNaN2) = QNaN2 and invalid exception as well]	IEEE_754 7.1	
(± num) OPERATOR (SNaN2)	IEEE_754 invalid	QSNaN2
(SNaN1) OPERATOR (± num)	IEEE_754 invalid	QSNaN1
(SNaN1) OPERATOR (SNaN2)	IEEE_754 invalid	QSNaN2
(QNaN1) OPERATOR (SNaN2)	IEEE_754 invalid	QSNaN2
(SNaN1) OPERATOR (QNaN2)	IEEE_754 invalid	QSNaN1
V9, Appendix B.5	IEEE_754 7.1	
F<s|d>TOi (+QNaN) = invalid	IEEE_754 invalid	$2^{31}-1$
F<s|d>TOi (+SNaN) = invalid	IEEE_754 invalid	$2^{31}-1$
F<s|d>TOx (+QNaN) = invalid	IEEE_754 invalid	$2^{63}-1$
F<s|d>TOx (+SNaN) = invalid	IEEE_754 invalid	$2^{63}-1$
F<s|d>TOi (–QNaN) = invalid	IEEE_754 invalid	-2^{31}
F<s|d>TOi (–SNaN) = invalid	IEEE_754 invalid	-2^{31}
F<s|d>TOx (–QNaN) = invalid	IEEE_754 invalid	-2^{63}
F<s|d>TOx (–SNaN) = invalid	IEEE_754 invalid	-2^{63}

B.1.4 Special Inexact Exceptions

OpenSPARC T2 follows SPARC V9 5.1.7.10.5 (IEEE_754 Section 7.5) and sets FSR_inexact whenever the rounded result of an operation differs from the infinitely precise unrounded result.

Additionally, there are a few special cases to be aware of:

TABLE B-9 Fp ↔ Int Conversions With Inexact Exceptions

Cases	Possible Exception	Result (in addition to accrued exception) if tem is cleared
V9, A.14: Fp convert to 32-bit integer when source operand lies between $-(2^{31}-1)$ and 2^{31} but is not exactly an integer. FsTOi, FdTOi.	IEEE_754 7.5 IEEE_754 inexact	 An integer number
V9, A.14: Fp convert to 64-bit integer when source operand lies between $-(2^{63}-1)$ and 2^{63} but is not exactly an integer. FsTOx, FdTOx.	IEEE_754 7.5 IEEE_754 inexact	 An integer number
V9, A.15: Convert integer to fp format when 32-bit integer source operand magnitude is not exactly representable in single precision (23-bit mantissa). Note, even if the operand is $> 2^{24}-1$, if enough of its trailing bits are zeros, it may still be exactly representable. FiTOs.	IEEE_754 7.5 IEEE_754 inexact	 An SP number
V9, A.15: Convert integer to fp format when 64-bit integer source operand magnitude is not exactly representable in single precision (23-bit mantissa). Note, even if the operand is $> 2^{24}-1$, if enough of its trailing bits are zeros, it may still be exactly representable. FxTOs.	IEEE_754 7.5 IEEE_754 inexact	 An SP number
V9, A.15: Convert integer to fp format when 64-bit integer source operand magnitude is not exactly representable in double precision (52-bit mantissa). Note, even if the operand is $> 2^{53}-1$, if enough of its trailing bits are zeros, it may still be exactly representable. FxTOd.	IEEE_754 7.5 IEEE_754 inexact	 A DP number

B.2 Subnormal Handling

The OpenSPARC T2 FGU follows the UltraSPARC I/UltraSPARC II subnormal handling instead of that used in OpenSPARC T1. While OpenSPARC T1 provides full hardware support for subnormal operands and results, OpenSPARC T2 generates an unfinished_FPop trap type in some cases. In addition, OpenSPARC T2 implements a nonstandard floating-point mode, whereas OpenSPARC T1 does not.

OpenSPARC T2 provides limited subnormal support in hardware when in standard mode (FSR.ns = 0) or interval arithmetic mode (GSR.im = 1) [Note that when GSR.im = 1, regardless of FSR.ns, OpenSPARC T2 operates in standard mode.]:

- OpenSPARC T2 supports full subnormal operand handling for single and double precision fp compares;
- OpenSPARC T2 supports gross underflow results for fp-to-fp conversions from higher to lower precision (FdTOs);
- OpenSPARC T2 supports gross underflow results in hardware for FMUL(s,d) and FDIV(s,d) which gives 90% of the optimal underflow performance at a fraction of the cost to completely support subnormal operands and results;
- For those instructions without any subnormal support, an unfinished trap is taken.

OpenSPARC T2 supports the following in nonstandard mode ((FSR.ns = 1) and (GSR.im = 0)):

- Subnormal operands and results are flushed to zero with the same sign, and execution is allowed to proceed without incurring the performance cost of an unfinished trap.

TABLE B-11 and TABLE B-12 show how each instruction type is explicitly handled.

Handling of the FMUL<s | d>, FDIV<s | d>, FdTOs instructions requires a few additional definitions:

- Let Signr = sign of result, RP = round to +infinity, RM = round to –infinity. Define RND as round mode bits. In standard mode, these can have two different sources:

  When in typical standard mode ((FSR.ns = 0) and (GSR.im = 0)), RND = FSR.rd

  When in interval arithmetic mode (GSR.im = 1), RND = GSR.irnd
- Let E(rs1) = biased exponent of rs1 operand, and E(rs2) = biased exponent of rs2 operand
- Let Er = unnormalized and unrounded biased exponent result

  For FMUL<s | d>: Er = E(rs1) + E(rs2) – EBIAS(P)

  For FDIV<s | d>: Er = E(rs1) – E(rs2) + EBIAS(P) – 1

  For FdTOs: Er = E(rs2) – EBIAS(P_rs2) + EBIAS(P_rd), where P_rs2 is the larger precision of the source and P_rd is the smaller precision of the destination
- Let Ef = final normalized and rounded biased exponent result

■ Define constants dependent on precision type (see TABLE B-10)

TABLE B-10 Subnormal Handling Constants Per Destination Precision Type

Precision (P)	Number of exponent field bits	Exponent Bias (EBIAS)	Exponent Max (EMAX)	Exponent Gross Underflow (EGUF)
Single	8	127	255	–25
Double	11	1023	2047	–54

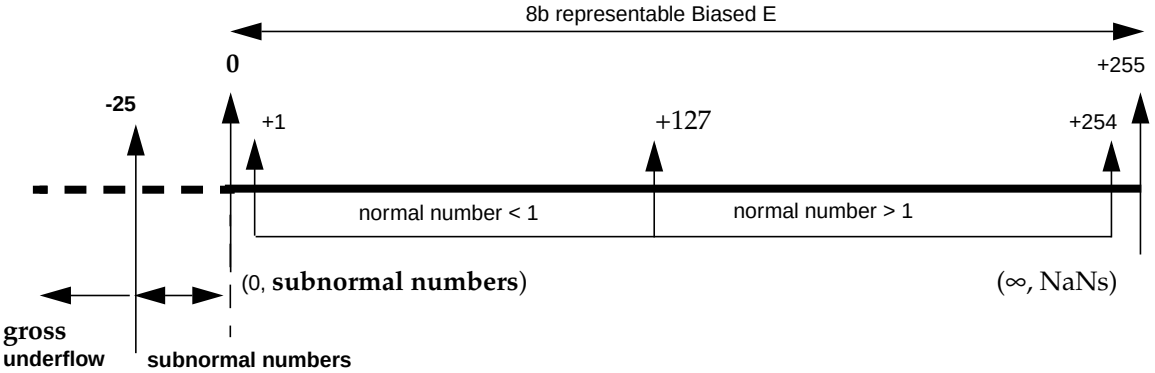


FIGURE B-1 Single Precision Unbiased vs. Biased Subnormals and Gross Underflow

■ Note that even though 0 ≤ [E(rs1) or E(rs2)] ≤ 255 for each single precision biased operand exponent, the *computed* Er can be 0 ≤ Er ≤ 255 or can even be negative. For example, for a FMULS instruction:

If E(rs1) = E(rs2) = +127, then Er = 127 + 127 –127 = +127

If E(rs1) = E(rs2) = 0, then Er = 0 + 0 - 127 = –127

■ Following the sections 5.1.7.6, 5.1.7.8, 5.1.7.9, and figures in 5.1.7.10 of *The SPARC Architecture Manual-Version 9*, Overflow Result is defined as follows:

> If the appropriate trap enable masks are not set (FSR.ofm = 0 *and* FSR.nxm = 0), then set aexc and cexc overflow and inexact flags: FSR.ofa = 1, FSR.nxa = 1, FSR.ofc = 1, 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), then only an IEEE overflow trap is generated: FSR.ftt = 1. The particular cexc bit that is set diverges from previous UltraSPARC I/ UltraSPARC II implementations to follow the SPARC V9 section 5.1.7.9 errata:
>
> 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.

- Following the sections 5.1.7.6, 5.1.7.8, 5.1.7.9 and figures in 5.1.7.10 of *The SPARC Architecture Manual-Version 9*, Gross Underflow Zero Result is defined as follows:

  Result = 0 (with correct sign)

  If the appropriate trap enable masks are not set (FSR.ufm=0 *and* FSR.nxm = 0), then set aexc and cexc underflow and inexact flags: FSR.ufa = 1, FSR.nxa = 1, FSR.ufc = 1, FSR.nxc = 1. No trap is generated.

  If any or both of the appropriate trap enable masks are set (FSR.ufm = 1 *or* FSR.nxm = 1), then only an IEEE underflow trap is generated: FSR.ftt = 1. The particular cexc bit that is set diverges from UltraSPARC I/ UltraSPARC II implementations to follow the SPARC V9 section 5.1.7.9 errata:

  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.

- Subnormal handling is overridden for the following cases:
  - Result is a QNaN or SNaN — by *The SPARC Architecture Manual-Version 9* Appendix B.2.2 (Table 27).

    Define "OP_NaN" as instruction uses a SNaN or QNaN operand.

    Examples:

    subnormal + SNaN = QNaN with invalid exception (No unfinished trap in standard mode and no FSR.nx in nonstandard mode)

    subnormal + QNaN = QNaN, no exception (No unfinished trap in standard mode and no FSR.nx in nonstandard mode)

  - Result already generates an exception.

    Define "OP_lt_0" as instruction uses an operand less than zero.

    Examples:

    sqrt(number less than zero) = invalid

  - Result is infinity.

    Define "OP_inf" as instruction uses infinity operand.

    Examples:

    subnormal $+\infty = \infty$ (No unfinished trap in standard mode and no FSR.nx in nonstandard mode)

    subnormal $\times \infty = \infty$ in standard mode; subnormal $\times \infty$ = QNaN with invalid exception in nonstandard mode since subnormal is flushed to zero.

  - Result is zero.

    Define "OP_0" as instruction uses a zero operand

Example:

subnormal × 0 = 0 (No unfinished trap in standard mode, and no FSR.nx in nonstandard mode)

B.2.1 One or Both Subnormal Operands

TABLE B-11 One or Both Subnormal Operands Handling *(1 of 3)*

Instructions	(FSR.ns = 0) *or* (GSR.im = 1) Standard Mode	(FSR.ns = 1) *and* (GSR.im = 0) Nonstandard mode

Single/Double Precision add, subtract [FADD<s|d>, FSUB<s|d>]	**if** (**not** (OP_NaN **or** OP_inf)) {
	generate unfinished trap
	}
		if (**not**(OP_NaN **or** OP_inf)) {
		executes w/subnormal operand flushed to 0 with the same sign
		FSR.nx ← 1
		}

Single/Double Precision FPCOMPARE [FCMP<s|d>, FCMPE<s|d>]	**if** (**not** OP_NaN) {
	execute the compare using the subnormal operand(s)
	}
		if (**not** OP_NaN) {
		executes the compare using the subnormal operand(s)[3]
		}

Single/Double Precision multiply [FMUL<s|d>]	**if** (**not** (OP_NaN **or** OP_inf **or** OP_0)) {
	If ((Er > EGUF(P))
	or
	(Er ≤ EGUF(P) **and** Signr=0 **and** RND=RP)
	or
	(Er ≤ EGUF(P) **and** Signr=1 **and** RND=RM))
	{generate unfinished trap}
	else {
	generate gross underflow zero result[1] }
	}
		if (**not**(OP_NaN **or** OP_0) {
		if (**not**(OP_inf)) {
		executes w/subnormal operand flushed to 0 with the same sign
		FSR.nx ← 1
		} **else** { // 1 op is subnormal, other is ∞
		executes w/subnormal operand flushed to 0 with the same sign
		FSR.nv ← 1 & return QNaN
		}
		}

TABLE B-11 One or Both Subnormal Operands Handling *(Continued)* *(2 of 3)*

Instructions	(FSR.ns = 0) *or* (GSR.im = 1) Standard Mode	(FSR.ns = 1) *and* (GSR.im = 0) Nonstandard mode

Single/double precision divide [FDIV<s|d>]	**if** (**not** (OP_NaN **or** OP_inf **or** OP_0)) {
	  if (**not** (Ef > EMAX(P))) {
	  // not overflow
	  if ((Er > EGUF(P))
	  or
	  (Er ≤ EGUF(P) **and** (Signr=0)
	  and (RND=RP))
	  or
	  (Er ≤ EGUF(P) **and** (Signr=1)
	  and (RND=RM)))
	  {
	  generate unfinished trap
	  } **else** {
	  generate gross-underflow zero result[1]
	  }
	  } **else** {
	  generate overflow result[2]
	  }
	}
		if (**not**(OP_NaN) {
		  if (**not**(OP_inf **or** OP_0) {
		  executes w/subnormal operand flushed to 0 with the same sign
		  if (rs1 and rs2 are flushed to zero) {
		  FSR.nv ← 1 // 0 ÷ 0 is invalid
		  } **else if** (rs2 divisor is flushed to zero) {
		  FSR.dz ← 1
		  } **else** {
		  FSR.nx ← 1
		  }
		  } **else if** (OP_inf) { // 1 op is subnormal,
		  // other = ∞
		  executes w/subnormal operand flushed to 0 with the same sign
		  // 0 ÷ ∞ is 0, and ∞ ÷ 0 is ∞
		  No exceptions are set.
		  Even if divisor is flushed to zero: no need to set FSR.dz
		  } **else if** (OP_0) { // 1 op subnorm, other = 0
		  executes w/subnormal operand flushed to 0 with the same sign
		  // 0 ÷ 0 is invalid exception
		  FSR.nv ← 1 // overrides FSR.dz
		  }
		}

Single to double precision multiply [FSMULD]	**if** (**not** (OP_NaN **or** OP_inf **or** OP_0)) {
	  generate unfinished trap
	}
		if (**not**(OP_NaN **or** OP_0) {
		  if (**not**(OP_inf)) {
		  executes w/subnormal operand flushed to 0 with the same sign
		  Set FSR.nx
		  } **else** { // 1 op is subnormal, other is ∞
		  executes w/subnormal operand flushed to 0 with the same sign
		  Set FSR.nv & return QNaN
		  }
		}

TABLE B-11 One or Both Subnormal Operands Handling *(Continued) (3 of 3)*

Instructions	(FSR.ns = 0) *or* (GSR.im = 1) Standard Mode	(FSR.ns = 1) *and* (GSR.im = 0) Nonstandard mode

Single/Double Precision square root [FSQRT(s,d)] (*note: single operand instruction)	**if** (**not** (OP_NaN **or** OP_inf)) {
	  if (OP_lt_0) {
	  FSR.nv ← 1 & return QNaN
	  } **else** {
	  generate unfinished trap
	  }
	}
		if (**not** (OP_NaN **or** OP_inf)) {
		  executes w/subnormal operand flushed to 0 with the same sign
		  if (OP_lt_0) { //that is, subnormal
		  FSR.nx ← 1; return –0
		  } **else if** (OP_eq_0) { //that is, 0
		  No exception; return 0 with same sign
		  } **else** {
		  FSR.nx ← 1
		  }
		}

Fp to Int and Fp Single to Double conversion [FsTOx, FdTOx, FsTOi, FdTOi, FsTOd] (*note single operand instructions)	**if** (**not** (OP_NaN **or** OP_inf)) {
	  generate unfinished trap
	}
		if (**not** (OP_NaN **or** OP_inf)) {
		  executes w/subnormal operand flushed to 0 with the same sign
		  FSR.nx ← 1
		}

Fp Double to Single conversion [FdTOs] (*note single operand instruction)	**if** (**not** (OP_NaN **or** OP_inf)) {
	  if ((Signr = 0 **and** RND = RP)
	  or
	  (Signr = 1 **and** RND = RM)) {
	  generate unfinished trap
	  } **else** {
	  generate gross underflow zero result[1]
	  }
	}
		if (**not** (OP_NaN **or** OP_inf)) {
		  executes w/subnormal operand flushed to 0 with the same sign
		  FSR.nx ← 1
		}

1. See gross underflow zero result definition on page 497.
2. See overflow definition on page 497.
3. This errata (not flushing subnormal operands to zero for only single and double precision fpcompares) ensures OpenSPARC T2 is backward compatible with UltraSPARC I/UltraSPARC II and UltraSPARC III.

B.2.2 Normal Operand(s) Giving Subnormal Result

TABLE B-12 Subnormal Result Handling for Two Normal Operands

Instructions	(FSR.ns = 0) *or* (GSR.im = 1)	(FSR.ns = 1) *and* (GSR.im = 0)
Single/Double Precision add, subtract [FADD<s|d>), FSUB<s|d>]	Generate unfinished trap	Generate gross underflow zero result[1]

Single/Double Precision multiply [FMUL<s|d>] and divide [FDIV<s|d>], Fp double-to-single conversion [FdTOs] (*note single operand instruction)	**if** (**not** (Ef >= 1)) { // that is, not subnormal intermediate result that rounded to normalized result
	if ((1 > Er > EGUF(P))
	or
	(Er ≤ EGUF(P) **and** Signr = 0 **and** RND = RP)
	or
	(Er ≤ EGUF(P) **and** Signr = 1 **and** RND = RM)) {
	generate unfinished trap
	}
	else {
	generate gross underflow zero result[1]
	}
	} **else** {
	generate normalized result
	}
		Generate gross underflow zero result[1]

1. See gross underflow zero result definition on page 497.

- For those instructions found in TABLE B-11 but not in TABLE B-12, TABLE B-11 is sufficient for their subnormal handling; so the additional rules in TABLE B-12 need not be applied.
- Multiplies that include a conversion from a smaller to larger precision (FSMULD) are not included in TABLE B-12 along with FMUL<s|d> because the larger precision result's exponent range is sufficient to represent a number that would have underflowed in the smaller precision's exponent range.

APPENDIX **C**

Differences From OpenSPARC T1

C.1 General Architectural and Microarchitectural Differences

OpenSPARC T2 follows the CMT philosophy of OpenSPARC T1, but adds more execution and cryptography capability to each physical core, as well as significant system-on-a-chip components and an enhanced L2 cache. The following lists the microarchitectural differences:

- Physical core consists of two integer execution pipelines and a single floating-point pipeline. OpenSPARC T1 had a single integer execution pipeline and all cores shared a single floating-point pipeline.
- Each physical core in OpenSPARC T2 supports eight strands, which all share the floating-point pipeline. The eight strands are partitioned into two groups of four strands, each of which shares an integer pipeline. OpenSPARC T1 shared the single integer pipeline among four strands.
- Pipeline in OpenSPARC T2 is eight stages, two stages longer than OpenSPARC T1.
- Instruction cache is 8-way associative, compared to 4-way in OpenSPARC T1.
- The L2 cache is 4 Mbyte, 8-banked and 16-way associative, compared to 3 Mbyte, 4-banked and 12-way associative in OpenSPARC T1.
- Data TLB is 128 entries, compared to 64 entries in OpenSPARC T1.
- The memory interface in OpenSPARC T2 supports fully buffered DIMMS (FBDs), providing higher capacity and memory clock rates, as described in Chapter 26.
- The OpenSPARC T2 memory channels support a single-DIMM option for low-cost configurations.

C.2 ISA Differences

There are a number of ISA differences between OpenSPARC T2 and OpenSPARC T1, as follows:

- OpenSPARC T2 fully supports all VIS 2.0 instructions. OpenSPARC T1 supports a subset of VIS 1.0 plus SIAM (the remainder of VIS 1.0 and 2.0 instructions trap to software for emulation, on OpenSPARC T1).
- OpenSPARC T2 supports the CMP spec, as described in Chapter 14. OpenSPARC T1 has its own version of CMP control/status registers. OpenSPARC T2 consists of eight physical cores, with eight virtual processors per physical core.
- OpenSPARC T2 does not support OpenSPARC T1's `idle` state or its idle, halt, or resume messages. Instead, OpenSPARC T2 supports parking and unparking as specified in the CMP chapter of *UltraSPARC Architecture 2007*. Note that parking is similar to OpenSPARC T1's `idle` state. OpenSPARC T2 does support an equivalent to the `halt` state, which on OpenSPARC T1 was entered via writing to HPR $1E_{16}$. However, OpenSPARC T2 does not support OpenSPARC T1's STRAND_STS_REG ASR, which holds the strand state. Halted state is not software-visible on OpenSPARC T2.
- OpenSPARC T2 does not support the INT_VEC_DIS register (which allowed any OpenSPARC T1 strand to generate an interrupt, reset, idle, or resume message to any strand). Instead, an alias to `ASI_INTR_W` is provided, which allows only the generation of an interrupt to any strand.
- OpenSPARC T2 supports ALLCLEAN, INVALW, NORMALW, OTHERW, POPC, and FSQRT<s|d> in hardware.
- OpenSPARC T2 has a floating-point unit that generates *fp_unfinished_operation* for most denorm cases and supports a nonstandard mode that flushes denorms to zero, as described in Appendix B. OpenSPARC T1 handles denorms in hardware, never generates an *FP_unfinished_operation* trap and does not support a nonstandard mode.
- OpenSPARC T2 generates an *illegal_instruction* trap on any quad-precision FP instruction, while OpenSPARC T1 generates an *fp_exception_other* trap on numeric and move-FP-quad instructions. See Table 5-2 on page 30.
- OpenSPARC T2 generates a *privileged_action* exception upon attempted access to hyperprivileged ASIs by privileged software. OpenSPARC T1 takes a *data_access_exception* exception in that case.
- OpenSPARC T2 supports PSTATE.tct; OpenSPARC T1 did not.
- OpenSPARC T2 implements SAVE similar to all previous UltraSPARC processors. OpenSPARC T1 implements a SAVE that updates the locals in the new window to be the same as the locals in the old window, and swaps the *in*s (*out*s) of the old window with the *out*s (*in*s) of the new widow.

- PSTATE.am masking details differ between OpenSPARC T1 and OpenSPARC T2, as described in Section 11.1.8, *Address Masking (Impdep #125)*, on page 89.
- OpenSPARC T2 implements PREFETCH fcn = 18_{16} as a prefetch invalidate cache entry, for efficient software cache flushing, as described in Section 28.15, *L2 Cache Flushing*, on page 443.

C.3 MMU Differences

The OpenSPARC T2 MMU is described in Chapter 12. The OpenSPARC T2 and OpenSPARC T1 MMUs differ as follows:

- OpenSPARC T2 has a 128-entry DTLB, whereas OpenSPARC T1 has a 64-entry DTLB.
- OpenSPARC T2 supports a pair of primary context registers and a pair of secondary context registers. OpenSPARC T1 supports a single primary context and single secondary context register.
- OpenSPARC T2 does not support a locked bit in the TLBs. OpenSPARC T1 supports a locked bit in the TLBs.
- OpenSPARC T2 supports only the sun4v TTE format for I/D-TLB Data-In and Data-Access registers. OpenSPARC T1 supports both the sun4v and the sun4u TTE formats.
- OpenSPARC T2 is compatible with UltraSPARC Architecture 2007 with regard to multiple flavors of data access exception (*DAE_**) and instruction access exception (*IAE_**). OpenSPARC T1 uses the UltraSPARC single flavor of *data_access_exception* and *instruction_access_exception*, indicating the "flavors" indicated in the SFSR register.
- OpenSPARC T2 supports a hardware Table Walker to accelerate ITLB and DTLB miss handling.
- The number and format of TSB configuration and pointer registers differs between OpenSPARC T1 and OpenSPARC T2. OpenSPARC T2 uses physical addresses for TSB pointers, OpenSPARC T1 uses virtual addresses for TSB pointers.
- OpenSPARC T1 and OpenSPARC T2 support the same four page sizes (8 Kbyte, 64 Kbyte, 4 Mbyte, 256 Mbyte). OpenSPARC T2 supports an *unsupported_page_size* trap when an illegal page size is programmed into TSB registers or attempted to be loaded into the TLB. OpenSPARC T1 forces an illegal page size being programmed into TSB registers to be 256 Mbytes and generates a *data_access_exception* trap when a page with an illegal size is loaded into the TLB.
- OpenSPARC T2 adds a demap real operation, which demaps all pages with r = 1 from the TLB.

- OpenSPARC T2 supports an I-TLB probe ASI.
- Autodemapping of pages in the TLBs only demaps pages of the same size or of a larger size in OpenSPARC T2, as described in Section 12.11.3, *I-/D-Demap Context (Type = 1)*, on page 149. In OpenSPARC T1, autodemap demaps pages of the same size, larger size, or smaller size.
- OpenSPARC T2 supports detection of multiple hits in the TLBs as described in Section 25.7.1.1, *ITLB Tag Multiple Hit Error (ITTM)*, on page 226 and Section 25.7.2.1, *DTLB Tag Multiple Hit Error (DTTM)*, on page 228.

C.4 Performance Instrumentation Differences

Both OpenSPARC T1 and OpenSPARC T2 provide access to hardware performance counters through the PIC and PCR registers. However, the events captured by the hardware differ significantly between OpenSPARC T1 and OpenSPARC T2, with OpenSPARC T2 capturing a much larger set of events, as described in Chapter 10. OpenSPARC T2 also has support to count events in hyperprivileged mode; OpenSPARC T1 did not.

In addition, the implementation of *pic_overflow* differs between OpenSPARC T1 and OpenSPARC T2. OpenSPARC T1 provides a disrupting *pic_overflow* on the instruction following the one that caused the overflow event. OpenSPARC T2 provides a disrupting *pic_overflow* on an instruction that generates the event but is within an epsilon number of event-generating instructions from the actual overflow.

Both OpenSPARC T2 and OpenSPARC T1 support DRAM performance counters.

C.5 Reset Differences

TBD

C.6 Error Handling Differences

Error handling differs quite a bit between OpenSPARC T2 and OpenSPARC T1. OpenSPARC T1 primarily employs hardware correction of errors, whereas OpenSPARC T2 primarily employs software correction of errors, as described in Chapter 25.

- OpenSPARC T2 uses the *store_error, sw_recoverable_error, data_access_error, instruction_access_error, internal_processor_error, hw_corrected_error, instruction_access_MMU_error,* and *data_access_MMU_error* traps for error handling. OpenSPARC T1 uses the *data_access_error, instruction_access_error, internal_processor_error, hw_corrected_error,* and *data_error* traps.
- OpenSPARC T2 IRF and FRF ECC errors are handled in software. OpenSPARC T1 corrects single-bit transient errors in hardware.
- OpenSPARC T2 has the ability to disable both error reporting and error traps. OpenSPARC T1 only has the ability to disable error traps.
- OpenSPARC T2 takes a deferred *store_error* trap on store buffer uncorrectable errors. OpenSPARC T1 does not have error correction on its store buffers.
- OpenSPARC T2 generates a trap on multiple hits in the ITLB, DTLB, I-cache, or D-cache. OpenSPARC T1 simply uses one of the matching entries.
- OpenSPARC T2 protects its MMU register array with parity, taking a trap if an error is detected during a tablewalk. OpenSPARC T1 MMU registers are not protected by parity. OpenSPARC T2 MMU error handling is described in Section 25.7.1, *ITLB Errors*, on page 226, Section 25.7.2, *DTLB Errors*, on page 228, and Section 25.7.11, *MMU Register Array (MRAU)*, on page 243.
- OpenSPARC T2 protects the TICK (TICK, STICK, HSTICK) compare registers, scratchpad registers, and trap stack registers with SECDED ECC, taking a trap if an error is detected while accessing the registers. OpenSPARC T1 left these registers unprotected by ECC.
- OpenSPARC T2 supports NotData in the L2 cache (NotData is not supported in memory in either OpenSPARC T1 or OpenSPARC T2).
- OpenSPARC T2 protects the vuad bits by SECDED ECC. OpenSPARC T1 protects the vuad bits by parity.

C.7 Power Management Differences

Both OpenSPARC T2 and OpenSPARC T1 support memory access throttling. The mechanisms for supporting CPU throttling differ between OpenSPARC T1 and OpenSPARC T2. OpenSPARC T2 power management is described in Chapter 27.

C.8 Configuration, Diagnostic, and Debug Differences

OpenSPARC T2 configuration and diagnostic support is described in Chapter 28. Debug support is described in Chapter 29. OpenSPARC T2 additions over OpenSPARC T1 include:

- OpenSPARC T2 supports instruction VA watchpoints.
- OpenSPARC T2 supports PA watchpoints.
- OpenSPARC T2 supports the *control_transfer_instruction* trap.
- OpenSPARC T2 implements Prefetch fcn = 18_{16} as a prefetch invalidate cache entry, for efficient software L2 cache flushing. In OpenSPARC T1, flushing of a cache line requires entering "direct-mapped replacement mode," where the L2 LRU is overridden by the address and then forcing out all 12-ways in a set via a displacement with the proper address.
- OpenSPARC T2 supports diagnostic access to the integer register file, store buffers, scratchpad, TICK (TICK, STICK, HSTICK) compare, trap stack, and MMU register arrays.
- OpenSPARC T2 does not require the diagnostic virtual address to match a valid tag for `ASI_DCACHE_DATA`.

APPENDIX **D**

Caches and Cache Coherency

D.1 Cache and Memory Interactions

This appendix describes various interactions between the caches and memory, and the management processes that an operating system must perform to maintain data integrity in these cases. In particular, it discusses the following:

- Invalidation of one or more cache entries—when and how to do it
- Differences between cacheable and noncacheable accesses
- Ordering and synchronization of memory accesses
- Accesses to addresses that cause side effects (I/O accesses)
- Nonfaulting loads
- Cache sizes, associativity, replacement policy, etc.

D.2 Cache Flushing

Data in the level-1 (read-only or writethrough) caches can be flushed by invalidating the entry in the cache (in a way that also leaves the L2 directory in a consistent state). Modified data in the level-2 (writeback) cache must be written back to memory when flushed.

Cache flushing is required in the following cases:

- **I-cache:** Flush is needed before executing code that is modified by a local store instruction. This is done with the FLUSH instruction. Flushing the I-cache with ASI accesses (Section 28.5, *L1 I-Cache Diagnostic Access*, on page 419) does *not* work, because it will leave the I-cache and the L2 directory inconsistent, thus breaking coherency and leading to the possibility of data corruption.

- **D-cache:** Flush is needed when a physical page is changed from (physically) cacheable to (physically) noncacheable. This is done with a displacement flush (*Displacement Flushing*, below).
- **L2 cache:** Flush is needed for stable storage. Examples of stable storage include battery-backed memory and transaction logs. The recommended way to perform this is by using the PrefetchICE instruction (see Section 28.15, *L2 Cache Flushing*, on page 443). Alternatively, this can be done by a displacement flush (see the next section). Flushing the L2 cache flushes the corresponding blocks from the I- and D-caches, because OpenSPARC T2 maintains inclusion between the L2 and L1 caches.

D.2.1 Displacement Flushing

Cache flushing of the L2 cache or the D-cache can be accomplished by a displacement flush. This is done by placing the cache in direct-map mode, and reading a range of read-only addresses that map to the corresponding cache line being flushed, forcing out modified entries in the local cache. Care must be taken to ensure that the range of read-only addresses is mapped in the MMU before starting a displacement flush; otherwise, the TLB miss handler may put new data into the caches. In addition, the range of addresses used to force lines out of the cache must not be present in the cache when starting the displacement flush. (If any of the displacing lines are present before starting the displacement flush, fetching the already present line will *not* cause the proper way in the direct-mapped mode L2 to be loaded; instead, the already present line will stay at its current location in the cache.)

Note | Diagnostic accesses to the L2 cache can be used to invalidate a line, but they are not an alternative to PrefetchICE or displacement flushing. L2 diagnostic accesses do not cause invalidation of L1 lines (breaking L1 inclusion) and modified data in the L2 cache will not be written back to memory using these ASI accesses. See Section 28.16, *L2 Cache Diagnostic Access*, on page 445.

D.2.2 Memory Accesses and Cacheability

Note | Atomic load-store instructions are treated as both a load and a store; they can be performed only in cacheable address spaces.

In OpenSPARC T2, all memory accesses are cached in the L2 cache (as long as the L2 cache is enabled). The cp bit in the TTE corresponding to the access controls whether the memory access will be cached in the primary caches (if cp = 1, the access is cached in the primary caches; if cp = 0 the access is not cached in the primary caches). Atomic operations are always performed at the L2 cache.

D.2.3 Coherence Domains

Two types of memory operations are supported in OpenSPARC T2: cacheable and noncacheable accesses, as indicated by the page translation. Cacheable accesses are inside the coherence domain; noncacheable accesses are outside the coherence domain.

SPARC V9 does not specify memory ordering between cacheable and noncacheable accesses. OpenSPARC T2 maintains TSO ordering, regardless of the cacheability of the accesses, relative to other access by processors. (Ordering of processor accesses relative to DMA accesses roughly follows PCI ordering rules. See *I/O Ordering Rules* on page 522.)

See the *The SPARC Architecture Manual-Version 9* for more information about the SPARC V9 memory models.

On OpenSPARC T2, a MEMBAR `#Lookaside` is effectively a NOP and is not needed for forcing order of stores vs. loads to noncacheable addresses.

D.2.3.1 Cacheable Accesses

Accesses that fall within the coherence domain are called cacheable accesses. They are implemented in OpenSPARC T2 with the following properties:

- Data resides in real memory locations.
- They observe the supported cache coherence protocol.
- The unit of coherence is 64 bytes at the system level (coherence between the virtual processors and I/O), enforced by the L2 cache.
- The unit of coherence for the primary caches (coherence between multiple virtual processors) is the primary cache line size (16 bytes for the data cache, 32 bytes for the instruction cache), enforced by the L2 cache directories.

D.2.3.2 Noncacheable and Side-Effect Accesses

Accesses that are outside the coherence domain are called noncacheable accesses. Accesses of some of these memory (or memory mapped) locations may result in side effects. Noncacheable accesses are implemented in OpenSPARC T2 with the following properties:

- Data may or may not reside in real memory locations.
- Accesses may result in program-visible side effects; for example, memory-mapped I/O control registers in a UART may change state when read.
- Accesses may not observe supported cache coherence protocol.
- The smallest unit in each transaction is a single byte.

Noncacheable accesses are all strongly ordered with respect to other noncacheable accesses (regardless of the e bit). Speculative loads with the e bit set cause a *DAE_so_page* trap.

Note | The side-effect attribute does not imply noncacheability.

D.2.3.3 Global Visibility and Memory Ordering

To ensure the correct ordering between the cacheable and noncacheable domains, explicit memory synchronization is needed in the form of MEMBARs or atomic instructions. CODE EXAMPLE D-1 illustrates the issues involved in mixing cacheable and noncacheable accesses.

CODE EXAMPLE D-1 Memory Ordering and MEMBAR Examples

Assume that all accesses go to non-side-effect memory locations.

```
Process A:
While (1)
{


   Store D1:data produced
1  MEMBAR #StoreStore (needed in PSO, RMO)
   Store F1:set flag
   While F1 is set (spin on flag)
   Load F1
2 MEMBAR #LoadLoad | #LoadStore (needed in RMO)

  Load D2
}

Process B:
While (1)
{

  While F1 is cleared (spin on flag)

      Load F1
2 MEMBAR #LoadLoad | #LoadStore (needed in RMO)

  Load D1

  Store D2
1 MEMBAR #StoreStore (needed in PSO, RMO)

Store F1:clear flag
}
```

Note | A MEMBAR #MemIssue or MEMBAR #Sync is needed if ordering of cacheable accesses following noncacheable accesses must be maintained for RMO cacheable accesses.

Due to load and store buffers implemented in OpenSPARC T2, CODE EXAMPLE D-1 may not work for RMO accesses without the MEMBARs shown in the program segment.

Under TSO, loads and stores (except block stores) cannot pass earlier loads, and stores cannot pass earlier stores; therefore, no MEMBAR is needed.

Under RMO, there is no implicit ordering between memory accesses; therefore, the MEMBARs at both #1 and #2 are needed.

D.2.4 Memory Synchronization: MEMBAR and FLUSH

The MEMBAR (STBAR in SPARC V8) and FLUSH instructions provide for explicit control of memory ordering in program execution. MEMBAR has several variations; their implementations in OpenSPARC T2 are described below. See the references to "Memory Barrier," "The MEMBAR Instruction," and "Programming With the Memory Models," in *The The SPARC Architecture Manual-Version 9* for more information.

D.2.4.1 MEMBAR #LoadLoad

All loads on OpenSPARC T2 switch a strand out until the load completes. Thus, MEMBAR #LoadLoad is treated as a NOP on OpenSPARC T2.

D.2.4.2 MEMBAR #StoreLoad

MEMBAR #StoreLoad forces all loads after the MEMBAR to wait until all stores before the MEMBAR have reached global visibility. MEMBAR #StoreLoad behaves the same as MEMBAR #Sync on OpenSPARC T2.

D.2.4.3 MEMBAR #LoadStore

All loads on OpenSPARC T2 switch a strand out until the load completes. Thus, MEMBAR #LoadStore is treated as a NOP on OpenSPARC T2

D.2.4.4 MEMBAR #StoreStore and STBAR

Stores on OpenSPARC T2 maintain order in the store buffer. Thus Membar #StoreStore is treated as a NOP on OpenSPARC T2.

Notes STBAR has the same semantics as MEMBAR #StoreStore; it is included for SPARC-V8 compatibility.

OpenSPARC T2 block stores and block-init stores are RMO. If a program needs to maintain order between RMO stores to different L2 cache lines, it should use a MEMBAR #Sync.

D.2.4.5 MEMBAR #Lookaside

Loads and stores to noncacheable addresses are "self-synchronizing" on OpenSPARC T2. Thus MEMBAR #Lookaside is treated as a NOP on OpenSPARC T2.

Note For SPARC V9 compatibility, this variation should be used before issuing a load to an address space that cannot be snooped,

D.2.4.6 MEMBAR #MemIssue

MEMBAR #MemIssue forces all outstanding memory accesses to be *completed* before any memory access instruction after the MEMBAR is issued. It must be used to guarantee ordering of cacheable accesses following noncacheable accesses. For example, I/O accesses must be followed by a MEMBAR #MemIssue before subsequent cacheable stores; this ensures that the I/O accesses reach global visibility (as viewed by other strands) before the cacheable stores after the MEMBAR.

Since loads are already self-synchronizing, Membar #MemIssue just needs to drain the store buffer (and receive all the store ACKs) before allowing memory operations to issue again. This is the same operation as OpenSPARC T2's Membar #Sync.

D.2.4.7 MEMBAR #Sync (Issue Barrier)

Membar #Sync forces all outstanding instructions and all deferred errors to be completed before any instructions after the MEMBAR are issued.

Note MEMBAR #Sync is a costly instruction; unnecessary usage may result in substantial performance degradation.

D.2.4.8 Self-Modifying Code (FLUSH)

The SPARC V9 instruction set architecture does not guarantee consistency between code and data spaces. A problem arises when code space is dynamically modified by a program writing to memory locations containing instructions. Dynamic

optimizers, LISP programs, and dynamic linking require this behavior. SPARC V9 provides the FLUSH instruction to synchronize instruction and data memory after code space has been modified.

In OpenSPARC T2, FLUSH behaves like a store instruction for the purpose of memory ordering. In addition, all instruction fetch (or prefetch) buffers are invalidated. The issue of the FLUSH instruction is delayed until previous (cacheable) stores are completed. Instruction fetch (or prefetch) resumes at the instruction immediately after the FLUSH.

D.2.5 Atomic Operations

SPARC V9 provides three atomic instructions to support mutual exclusion. These instructions behave like both a load and a store but the operations are carried out indivisibly. Atomic instructions may be used only in the cacheable domain.

An atomic access with a restricted ASI in unprivileged mode (PSTATE.priv = 0) causes a *privileged_action* trap. An atomic access with a noncacheable address causes a *data_access_exception* trap (with SFSR.ft = 4, atomic to page marked noncacheable). An atomic access with an unsupported ASI causes a *DAE_invalid_ASI* trap. TABLE D-1 lists the ASIs that support atomic accesses.

TABLE D-1 ASIs That Support SWAP, LDSTUB, and CAS

ASI Name
ASI_NUCLEUS{_LITTLE}
ASI_AS_IF_USER_PRIMARY{_LITTLE}
ASI_AS_IF_USER_SECONDARY{_LITTLE}
ASI_AS_IF_PRIV_PRIMARY{_LITTLE}
ASI_AS_IF_PRIV_SECONDARY{_LITTLE}
ASI_AS_IF_PRIV_NUCLEUS{_LITTLE}
ASI_PRIMARY{_LITTLE}
ASI_SECONDARY{_LITTLE}
ASI_REAL{_LITTLE}

Notes

Atomic accesses with nonfaulting ASIs are not allowed, because these ASIs have the load-only attribute.

For all atomics, allocation is done to the L2 cache only and will invalidate the L1s.

D.2.5.1 SWAP Instruction

SWAP atomically exchanges the lower 32 bits in an integer register with a word in memory. This instruction is issued only after store buffers are empty. Subsequent loads interlock on earlier SWAPs.

D.2.5.2 LDSTUB Instruction

LDSTUB behaves like SWAP, except that it loads a byte from memory into an integer register and atomically writes all 1's (FF_{16}) into the addressed byte.

D.2.5.3 Compare and Swap (CASX) Instruction

Compare-and-swap combines a load, compare, and store into a single atomic instruction. It compares the value in an integer register to a value in memory; if they are equal, the value in memory is swapped with the contents of a second integer register. All of these operations are carried out atomically; in other words, no other memory operation may be applied to the addressed memory location until the entire compare-and-swap sequence is completed.

D.2.6 Nonfaulting Load

A nonfaulting load behaves like a normal load, except that

- It does not allow side-effect access. An access with the `e` bit set causes a *DAE_so_page* trap.
- It can be applied to a page with the `nfo` bit set; other types of accesses will cause a *DAE_NFO_page* trap.

Nonfaulting loads are issued with `ASI_PRIMARY_NO_FAULT{_LITTLE}` or `ASI_SECONDARY_NO_FAULT{_LITTLE}`. A store with a `NO_FAULT` ASI causes a *DAE_invalid_ASI* trap.

When a nonfaulting load encounters a TLB miss, the operating system should attempt to translate the page. If the translation results in an error (for example, address out of range), a 0 is returned and the load completes silently.

Typically, optimizers use nonfaulting loads to move loads before conditional control structures that guard their use. This technique potentially increases the distance between a load of data and the first use of that data, to hide latency; it allows for more flexibility in code scheduling. It also allows for improved performance in certain algorithms by removing address checking from the critical code path.

For example, when following a linked list, nonfaulting loads allow the null pointer to be accessed safely in a read-ahead fashion if the operating system can ensure that the page at virtual address 0_{16} is accessed with no penalty. The `nfo` (nonfault access

only) bit in the MMU marks pages that are mapped for safe access by nonfaulting loads but can still cause a trap by other, normal accesses. This allows programmers to trap on wild pointer references (many programmers count on an exception being generated when accessing address 0_{16} to debug code) while benefitting from the acceleration of nonfaulting access in debugged library routines.

D.3 L1 I-Cache

The L1 Instruction cache is 16 Kbytes, physically tagged and indexed, with 32-byte lines, and 8-way associative with random replacement. The format used to index the cache is shown in TABLE D-2.

TABLE D-2 L1 Instruction Cache Addressing

Bit	Field	Description
39:11	tag	Tag for cache line.
10:5	set	Selects cache set containing the cache line.
4:2	instr	Selects 32-bit instruction in cache line.
1:0	—	Always 0 for access to 32-bit instructions.

D.3.1 LFSR Replacement Algorithm

Details TBD.

D.3.2 Direct-Mapped Mode

The I-cache direct-mapped mode (see Section 28.4.1, *ASI_LSU_DIAG_REG*, on page 418) works by forcing all replacements to the “way” identified by bits [13:11] of the virtual address. Since lines already present are not affected but only new lines brought into the cache are affected, it is safe to turn on (or off) the direct-mapped mode at any time.

D.3.3 I-Cache Disable

Clearing the I-cache enable bit (see Section 28.1, *ASI_LSU_CONTROL_REG*, on page 413) stops all accesses to the I-cache for that strand. All fetches will miss, and the returned data will not fill the I-cache. Invalidates will still be serviced while the I-cache is disabled.

D.4 L1 D-Cache

The L1 Data cache is 8 Kbytes, writethrough, physically tagged and indexed, with 16-byte lines, and 4-way associative with true LRU replacement. The format used to index the cache is shown in TABLE D-3.

TABLE D-3 L1 Data Cache Addressing

Bit	Field	Description
39:11	tag	Tag for cache line.
10:4	set	Selects cache set containing the cache line.
3:0	data	Selects data byte(s) in cache line.

D.4.1 LRU Replacement Algorithm

The D-cache replacement algorithm is true least-recently-used (LRU). Six bits are maintained for each cache index.

D.4.2 Direct-Mapped Mode

In direct-mapped mode, the D-cache (see Section 28.4.1, *ASI_LSU_DIAG_REG*, on page 418) works by changing the replacement algorithm from LRU to instead use two bits of index (address[12:11]) to select the "way." Since lines already present are not affected but only new lines brought into the cache are affected, it is safe to turn on (or off) the direct-mapped mode at any time.

Note that if the D-cache is in direct-mapped mode, and a parity error occurs, the way replaced will be the way which experienced the parity error. This overrides the index selected by the address in direct-mapped mode.

D.4.3 D-Cache Disable

The D-cache may be disabled by setting dc = 0 in the ASI_LSU_CONTROL_REGISTER (see Section 28.1, *ASI_LSU_CONTROL_REG*, on page 413). When disabled, accesses to the D-cache behave as follows. A load which hits in the D-cache ignores the cached data, and fetches the data from L2. A load which misses in the cache fetches the data from L2, but does not allocate the line in the data cache. Stores that miss in the data cache never allocate in the data cache (as normal). Stores that hit in the data cache are performed in the L2, then update the data cache (as normal).

Even if the D-cache is disabled, L2 still keeps the D-cache coherent. Invalidations caused by L2 replacements, stores from other cores, or DMA stores from I/O activity which hit data in the D-cache cause those lines to be invalidated.

To get the D-cache fully disabled, the dc bit must be off on all strands in the virtual processor, and the D-cache must be flushed in a way that doesn't bring new lines back in. This can be done by storing (from a different core) to each line that is in the D-cache, or by displacement flushing the L2 cache so that inclusion will force all D-cache lines to be invalidated.

D.5 L2 Cache

The L2 combined instruction/data cache is 4 Mbytes, writeback, physically tagged and indexed, with 64B lines, 8-banked, and 16-way associative with pseudo-LRU replacement. The format used to index the full cache is shown in TABLE D-4.

TABLE D-4 L2 Cache Addressing (8 banks)

Bit	Field	Description
39:18	tag	Tag for cache line.
17:9	set	Selects cache set containing the cache line. Bit positions listed are used when L2_IDX_HASH_EN.enb_hp = 0. If L2_IDX_HASH_EN.enb_hp = 1, the set selection is done using PA{17:13} **xor** PA{32:28}, PA{19:18C **xor** PA{12:11}, PA{10:9}.
8:6	bank	Selects bank containing the cache line.
5:0	data	Selects data byte(s) in the cache line.

OpenSPARC T2 also supports 4-banked and 2-banked modes to assist in the recovery of partially good die. TABLE D-5 and TABLE D-6 show the format used to index the cache in these reduced modes.

TABLE D-5 L2 Cache Addressing (4 banks)

Bit	Field	Description
38:17	tag	Tag for cache line. Bit positions listed are used when L2_IDX_HASH_EN.enb_hp = 0. If L2_IDX_HASH_EN.enb_hp = 1, the tag stored is PA{38:18}, PA{17} **xor** PA{32}.
16:8	set	Selects cache set containing the cache line. Bit positions listed are used when enb_hp=0. If L2_IDX_HASH_EN.enb_hp = 1, the set selection is done using PA{16:13}^PA{31:28}, PA{19:18} **xor** PA{12:11}, PA{10:8}.
7:6	bank	Selects bank containing the cache line.
5:0	data	Selects data byte(s) in the cache line.

TABLE D-6 L2 Cache Addressing (2 banks)

Bit	Field	Description
37:16	tag	Tag for cache line. Bit positions listed are used when L2_IDX_HASH_EN.enb_hp = 0. If L2_IDX_HASH_EN.enb_hp = 1, the tag stored is PA{37:18}, PA{17:16} **xor** PA{32:31}.
15:7	set	Selects cache set containing the cache line. Bit positions listed are used when L2_IDX_HASH_EN.enb_hp = 0. If L2_IDX_HASH_EN.enb_hp = 1, the set selection is done using PA{15:13} **xor** PA{30:28}, PA{19:18} **xor** PA{12:11}, PA{10:7}.
6	bank	Selects bank containing the cache line.
5:0	data	Selects data byte(s) in the cache line.

D.5.1 NRU Replacement Algorithm

A used-bit scheme is used to implement an NRU (Not Recently Used) replacement. The used bit is set each time a cache line is accessed or when initially fetched from memory. If setting the used-bit causes all used bits (at an index) to be set, the remaining used bits are cleared instead.

In addition, each line has an allocate bit (a), which is set while a line is in a multicycle operation. This can be a cache fill, in which case the a bit gets set when the location is allocated, and the a bit gets cleared when the location is filled with memory data. Alternatively, this could be a multipass operation, either an atomic operation or a subword store (which requires read-modify-write); where the a bit is set on the first pass and cleared on the second/final pass. Any line that has the a bit set is ineligible for replacement.

Each L2 bank has a single rotating replacement pointer, which is the "starting point" to find the "way" to replace. On a miss, the L2 looks for the first line at that index with both u bit and a bit clear, starting with the "way" pointed at by the replacement pointer. If all lines have u bit or a bit set, all u bits are cleared and the scan repeated. The replacement pointer is then rotated forward one "way."

Since the replacement pointer is used by all sets of the L2, replacement is somewhat more random than if each set/index had its own replacement pointer. The replacement pointer is incremented on any L2 miss that causes a cache fill (that is, not DMA reads or full-line DMA writes). The replacement pointer is only reset (put to a known state) by POR, warm reset, or debug reset.

Valid bits do not affect the NRU replacement. In normal use, the only case that creates a line marked Invalid is when a WRI transaction (full-line DMA write) hits in the L2. The WRI will invalidate the L1 and L2, but write directly to memory. This is rare enough that it is not worthwhile to take Valid into account in the replacement algorithm.

D.5.2 Directory Coherence

The L2 cache has a directory of all L1 lines, both I-cache and D-cache, implemented as duplicate tags. Thus, the L2 always knows exactly which lines are in which L1 caches, and in which "way" of each cache. When the L1 requests a line from the L2, the virtual processor specifies whether the line will be allocated (put into the cache), and which "way" it will go into.

The L2 to virtual processor (CPX) protocol allows the L2 to issue invalidates to any/all of the cores simultaneously, but only a single invalidation to each core. For this reason, for a given virtual processor, an L1 line is only allowed to be in either I-cache or the D-cache, but not both. The invalidate transaction includes only index, way, and L1-cache (I or D); it does not include the address.

Since the L2 tracks which lines are in which L1 ways, just invalidating an L1 line via `ASI_ICACHE_TAG` or `ASI_DCACHE_TAG` is not safe and can lead to stale data problems and data corruption. The problem occurs if a line is marked invalid, and a subsequent access to the L1-cache refetches the line, but into a different "way." At this time, the L2 directory has the same line in two places in its directory. Later, when the L2 wants to invalidate that address, it gets a double hit on its CAM access, which the logic does not support. (If an L1 line needs to be invalidated, it can be done by injecting an error into its tag, then accessing it. The hardware error handling will invalidate the line, and inform the L2 directory.)

D.5.3 Direct-Mapped Mode

The L2-cache direct-mapped mode (see Section 28.14.1, *L2 Control Register*, on page 438) works by changing the replacement algorithm from NRU to instead use four bits of index (address{21:18}) to select the "way." Since lines already present are not affected but only new lines brought into the cache are affected, it is safe to turn on (or off) the direct-mapped mode at any time.

D.5.4 L2 Cache Disable

The L2 cache disable (see Section 28.14.1, *L2 Control Register*, on page 438) actually disables an L2 bank. Thus, it is recommended that the L2 be flushed first so that modified lines are written back to memory. While an L2 bank is disabled, the cache effectively has only a single line, which is invalidated or written back at the end of the access. Thus, a store will miss to memory, perform the write into the one-line cache, then flush. Then, the next cache access can be started.

The L2 directory is not used while the L2 cache is disabled. Thus, all L1 caches must be disabled and emptied before disabling any (or all) L2 banks.

D.6 I/O Ordering Rules

OpenSPARC T2 supports the PCI ordering rules to a much higher degree than recent SPARC processors. In particular, OpenSPARC T2 supports maintaining the order of DMA completion notification relative to previous DMA writes, where DMA completion notification can be achieved by (1) receiving an interrupt, (2) reading the device's control register (PIO read return), or (3) reading updated device status in memory (DMA write).

OpenSPARC T2 does not support the following PCI ordering rules:

1. DMA reads may pass previous DMA writes, unless they are to the same address. This rule is not needed for Producer/Consumer, and no need has ever been seen for this.
2. DMA read returns may pass previous PIO writes. This means that if a processor issues a PIO store, followed by a memory store, and wants to guarantee that the I/O device sees the PIO store before seeing the updated memory data, the processor must issue an intervening PIO load to the same device.

The complete rules are described in the following two tables.

TABLE D-7 Outbound (Memory-to-I/O) Transaction Ordering

Row pass Column?	PIO Write	PIO Read Request	DMA Read Completion
PIO Write	No	No	Yes
PIO Read Request	No	No	Yes
DMA Read Compl	Yes	Yes	Yes

Notes:

1. PIO requests are ordered relative to each other, as long as they are to the same "device."
2. DMA read completions follow a different path than PIO requests and are thus unordered relative to PIO requests. So if a processor issues a PIO store followed by a memory store, other processors see them in that order, but an I/O device may see the memory store first.

TABLE D-8 Inbound (I/O-to-Memory) Transaction Ordering

Row pass Column?	DMA Write (Col. 2)	JBUS Mondo Interrupt (Col. 3)	DMA Read Request (Col. 4)	PIO Read Completion (Col. 5)
DMA Write (row A)	No	Yes	Yes	Yes
JBUS Mondo Intr (row B)	No	Yes	Yes	Yes
DMA Read Req (row C)	Yes	Yes	Yes	Yes
PIO Read Compl (row D)	No	Yes	Yes	No

APPENDIX **E**

ECC Codes

E.1 ECC Summary

TABLE E-1 lists the arrays that are protected by ECC.

TABLE E-1 Error Handling

Array	ECC	Size	Instances	Total
L2 Cache Data	32+7 SEC/DED	936 KB	4	3744 KB
L2 Writeback Buffer	32+7 SEC/DED	624 B	4	2496 B
L2 Fill Buffer	32+7 SEC/DED	624 B	4	2496 B
L2 DMA Input Buffer	32+7 SEC/DED	312 B	4	1248 B
L2 Cache Tag	22+6 SEC	42 KB	4	168 KB
FP Register File	32+7 SEC/DED	2560 B	8	20.8 KB
Integer Register File	64+8 SEC/DED	9 KB	8	72 KB
Trap Stack Array (TSA)	67+8 SEC/DED	320 B	16	5120 B
Tick Compare Array (TCA)	67+8 SEC/DED	288 B	8	2304 B
Scratchpad Array (SCA)	67+8 SEC/DED	288 B	16	4608 B
Store Buffer Data (SBD)	32+7 SEC/DED	640 B	8	5120 B

The L2 Writeback Buffer, L2 Fill Buffer, and L2 DMA Buffer contain data that is in the processor being moved to or from the cache. The ECC generation and check blocks were placed to include these buffers, but errors in these buffers are indistinguishable from L2 cache errors.

E.2 IRF ECC Code

TABLE E-2 IRF Check Bit Generation

Check {7:0}	Data{31:0}																															
	31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
C0	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	1	0	1	1
C1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1
C2	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	0
C3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
C4	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
C5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	0	0	1	0	1	1	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	1	1
	Data{63:32}																															
	63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
C0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
C1	1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
C2	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1
C3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
C4	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
C5	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	1

1 – Data bit is **xor**ed into calculation of that check bit.

0 – Data bit is not part of the check bit calculation.

The syndrome calculation is the inverse of the checkbit calculation, for synd{6:0}. synd{7}, however, is simply the **xor** of data{63:0} and check{7:0}.

TABLE E-3 Syndrome Table for IRF ECC Code

SYND {7:4} Value	SYND [3:0} Value															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	ne	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
1	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
2	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
3	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
4	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
5	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
6	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
7	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
8	C7	C0	C1	0	C2	1	2	3	C3	4	5	6	7	8	9	10
9	C4	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	C5	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
B	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
C	C6	57	58	59	60	61	62	63	M	M	M	M	M	M	M	M
D	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
F	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

ne — No error

C0–C7 — Single bit error on syndrome/check bit of that number.

0–63 — Single bit error on data bit of that number.

U — Uncorrectable double (or 2*n*) bit error.

M — Triple or worse (2*n* + 1) bit error.

E.3 FRF ECC CODE

The floating-point register files use the same 32+7 SEC/DED codes as the L2 Data, except that the FRF is never intentionally marked with a poison indication, and triple or worse ($2n + 1$) bit errors are reported as correctable errors (FRFC). Software will need to examine the syndrome to determine if a FRFC is a triple or worse error.

E.4 TSA, TCA, and SCA ECC Code

Each of these arrays uses the same ECC code. The TCA and SCA wire the four most-significant bits to 0 as they are only 64 data bits wide.

TABLE E-4 TSA, TCA, SCA 68/8 ECC Check Bit Generation

	Data{31:0}																															
Check {7:0}	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
C0	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	1	0	1	1
C1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1
C2	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	0
C3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
C4	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
C5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	0	0	1	0	1	1	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	1	1

TABLE E-4 TSA, TCA, SCA 68/8 ECC Check Bit Generation

	Data{31:0}																													
Check {7:0}	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
	Data{63:32}																													
	63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
C0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
C1	1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
C2	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1
C3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
C4	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	1

	Data[67:64]			
Check {7:0}	**67**	**66**	**65**	**64**
C0	0	0	1	0
C1	1	1	0	0
C2	0	0	0	0
C3	1	1	1	1
C4	0	0	0	0
C5	0	0	0	0
C6	1	1	1	1
C7	1	0	0	1

TABLE E-5 Syndrome Table for TSA, TCA, and SCA Data ECC Code

synd {7:4} Value	synd {3:0} Value															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ne	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
1	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
2	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
3	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
4	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
5	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
6	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
7	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
8	M	C0	C1	0	C2	1	2	3	C3	4	5	6	7	8	9	10
9	C4	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	C5	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
B	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
C	C6	57	58	59	60	61	62	63	64	65	66	67	M	M	M	M
D	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
F	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

ne — No error

C0–C6 — Single bit error on syndrome/check bit of that number.

0–67 — Single bit error on data bit of that number.

U — Uncorrectable double (or 2*n*) bit error.

M — Triple or worse (2*n* + 1) bit error

E.5 Store Buffer Data Array (SBD), L2 UA, L2 VD, and L2 Data ECC Code

TABLE E-6 L2 Data Check Bit Generation

	data{31:0}																															
check {6:0}	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
C0	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	1	0	1	1
C1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1
C2	1	1	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	0
C3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
C4	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
C5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C6	0	0	1	0	1	1	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	1	1

1 – Data bit is **xor**ed into calculation of that check bit.

0 – Data bit is not part of the check bit calculation.

The syndrome calculation is the inverse of the checkbit calculation, for synd{5:0}. synd{6}, however, is simply the **xor** of data{31:0} and check{6:0}.

TABLE E-7 Syndrome Table for L2 Data ECC Code

synd {6:4} Value	synd {3:0} Value															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	ne	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
1	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
2	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
3	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
4	C6	C0	C1	0	C2	1	2	3	C3	4	5	6	7	8	9	10
5	C4	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
6	C5	26	27	28	29	30	31	M	M	M	M	M	M	M	M	M
7	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N/M

ne — No Error.

C0–C6 — Single bit error on syndrome/check bit of that number.

0–31 — Single bit error on data bit of that number.

N/M — NotData, or triple or worse ($2n + 1$) bit error.

U — Uncorrectable double (or $2n$) bit error.

M — Triple or worse ($2n + 1$) bit error.

E.6 L2 Tag ECC Code

The syndrome is not captured on L2 Tag ECC errors (LTC), so we only document checkbit calculation for L2 tag.

TABLE E-8 L2 Tag Check Bit Generation

	Tag{21:0}																						
Check {5:0}	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
C0	1	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	1	1
C1	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	1	0	1	1
C2	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1
C3	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	0
C4	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0
C5	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

1 – Data bit is **xor**ed into calculation of that check bit.

0 – Data bit is not part of the check bit calculation.

E.7 Memory Extended ECC Support

OpenSPARC T2 supports Extended ECC error correction (QEC/OED) for main memory, where we can correct any error contained within a single memory nibble (4 bits), and detect as uncorrectable any error that is contained within any two nibbles. The ECC coding scheme uses 4-bit words (16 symbols), 128-bit data (16 words), and a 16-bit syndrome (4 words), and uses Galois Field of (2^4) to implement Add/Multiply Operation that is completely inclusive within its field (Definition of Galois Field). It uses 3×4 bit correction code + 1×4 bit detection code (16 bits total) to correct single-nibble errors and detect double-nibble errors. While the addition is a trivial bitwise **xor**, the multiplication is not as straightforward and involves a Modulo multiplication using its field Primitive Polynomial of value 10011. Also, the syndrome or Parity generated is **xor**ed bitwise with the parity of the address ((**xor_bit_vector**(PA{39:9})) **xor** PA{6}) to that location.

E.7.1 Nomenclature and Nibble Order

The data nibbles and check/syndrome nibbles are numbered in a little-endian fashion, so the order as seen by software is:

C3 C2 C1 C0 N31 N30 … N5 N4 N3 N2 N1 N0

so N0 is made up of data{3:0}, N1 is data{7:4}, etc.

The data for the ECC code is referred to either as check nibbles or as syndrome (nibbles), depending on where it is relative to the ECC generation calculation and to the ECC check calculation. After the ECC generation calculation, it is called "check nibbles," and this is what is stored in physical DRAM. When memory is accessed, the data nibbles and check nibbles are read out of DRAM and run through an ECC check calculation, which produces the "syndrome nibbles," synd{15:0}.

The check nibbles are not directly accessible by software.

E.7.1.1 External Hardware Bit Order

External to the chip, the data nibbles are numbered in a little-endian fashion, but the check nibbles are numbered big-endian, so the order as seen by an external logic analyzer is:

TABLE 0-1

C0 C1 C2 C3	N3 1 N3 0 N2 9 N2 8 . . . . . N3 N2 N1 N0
← dramn_cb{15:0} →	←——— dramn_dq{127:0} ———→

So N0 is made up of dq{3:0}, N1 is dq{7:4}, etc. However, the check nibbles are wired in big-endian order by nibble, but little-endian within each check nibble, so c0{3:0} is made up of cb{15:12}, c1{3:0} is cb{11:8}, c2{3:0} is cb{7:4}, and c3{3:0} is cb{3:0}.

E.7.2 Memory ECC Code Description

The calculation for the check nibbles is as follows:

Check Nibble0 (4 bits) ←
(N0 + 2 × N1 + 3 × N2 + 4 × N3 + 5 × N4 + 6 × N5 + 7 × N6 + 8 × N7 +
9 × N8 + A × N9 + B × N10 + C × N11 + D × N12 + E × N13 + F × N14 +
N15 + 2 × N16 + 3 × N17 + 4 × N18 + 5 × N19 + 6 × N20 + 7 × N21 + 8 × N22 +
9 × N23 + A × N24 + B × N25 + C × N26 + D × N27 + E × N28 + F × N29 + N31)
xor (addr_parity **::** addr_parity **::** addr_parity **::** addr_parity)

Check Nibble1 (4 bits) ←
(N0 + N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 +
N11 + N12 + N13 + N14 + N30 + N31)
xor (addr_parity **::** addr_parity **::** addr_parity **::** addr_parity)

Check Nibble2 (4 bits) ←
(N15 + N16 + N17 + N18 + N19 + N20 + N21 + N22 + N23 + N24 +
N25 + N26 + N27 + N28 + N29 + N30 + N31)

xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

Check Nibble3 (4 bits) ←
(N0 + 9 × N1+ E × N2 + D × N3 + B × N4 + 7 × N5 + 6 × *N6 + F × N7 + 2 × N8 +
C × N9 + 5 × N10+ A × N11 + 4 × N12 + 3 × N13 + 8 × N14 + N15 + 9 × N16 +
E × N17 + D × N18 + B × N19 + 7 × N20 + 6 × N21 + F × N22 + 2 × N23 + C × N24 +
5 × N25 + A × N26+ 4 × N27 + 3 × N28 + 8 × N29 + N30)
xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

The calculation for the syndrome nibbles is similar, but includes the corresponding check nibble:

Syndrome Nibble0 (4 bits) ←
(C0 + N0 + 2 × N1 + 3 × N2 + 4 × N3 + 5 × N4 + 6 × N5 + 7 × N6 + 8 × N7 +
9 × N8 + A × N9 + B × N10 + C × N11 + D × N12 + E × N13 + F × N14 +
N15 + 2 × N16 + 3 × N17 + 4 × N18 + 5 × N19 + 6 × N20 + 7 × N21 + 8 × N22 +
9 × N23 + A × N24 + B × N25 + C × N26 + D × N27 + E × N28 + F × N29 + N31)
xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

Syndrome Nibble1 (4 bits) ←
(C1 + N0 + N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10 +
N11 + N12 + N13 + N14 + N30 + N31)
xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

Syndrome Nibble2 (4 bits) ←
(C2 + N15 + N16 + N17 + N18 + N19 + N20 + N21 + N22 + N23 + N24 +
N25 + N26 + N27 + N28 + N29 + N30 + N31)
xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

Syndrome Nibble3 (4 bits) ←
(C3 + N0 + 9 × N1+ E × N2 + D × N3 + B × N4 + 7 × N5 + 6 × N6 + F × N7 + 2 ×
N8 + C × N9 + 5 × N10+ A × N11 + 4 × N12 + 3 × N13 + 8 × N14 + N15 + 9 × N16 +
E × N17 + D × N18 + B × N19 + 7 × N20 + 6 × N21 + F × N22 + 2 × N23 + C × N24 +
5 × N25 + A × N26+ 4 × N27 + 3 × N28 + 8 × N29 + N30)
xor (addr_parity :: addr_parity :: addr_parity :: addr_parity)

Error correction is accomplished by following equations. If S0, S1, S2, and S3 are the 4 Syndrome nibbles,

Position 0 – 14 (nibble position) ("/" indicates Galois field division, the inverse of Galois field multiplication operation in TABLE E-9 on page 537)

if (S2 = 0 **and** (S1 ≠ 0) **and** (S0 ≠ 0))
then
nibble_to_correct ← ((S0 / S1) – 1);
corrected_data ← S1 + erred_nibble;
endif

Position 15 – 29 (nibble position)

if (S1 = 0 **and** S0 ≠ 0 **and** S2 ≠ 0), then {
    nibble_to_correct ← ((S0 / S2) + 14);
    corrected_data ← S2 + erred_nibble;
endif

Position 30 (nibble position)

if (S0 = 0 **and** S1 ≠ 0 **and** S2 ≠ 0 **and** S1 = S2)
then
    nibble_to_correct ← N30
    corrected_data ← S1+ erred_nibble;
endif

Position 31 (nibble position)

If (S0 ≠ 0 **and** S1 ≠ 0 **and** S2 ≠ 0 **and** S0 = S1 = S2)
then
    nibble_to_correct ← N31;
    corrected_data ← S1+ erred_nibble;
endif

Notes Nibble S3 is not used in correction but only for multiple error detection. Double errors are detected if (1) exactly two of the check-nibbles are non-zero, or (2) all four of the check-nibbles are non-zero, or (3) the nibble position as indicated by S0/S1 or S0/S2 does not match the nibble position as indicated by S3/S1 or S3/S2, or (4) S1 and S2 are non-zero and the non-zero check-nibbles are not all equal.

This memory ECC scheme assumes that memory is implemented with x4 DRAMs. If x8 parts are used, this is effectively a SEC/DED scheme, with some multibit correction, but no Extended ECC survival capability.

E.7.3 Memory Address Parity Protection

Note that address parity is added (**xor**ed) into all of the check bits. Any normal address parity error will be detected as a multiple-nibble uncorrectable error, since all four syndrome nibbles will be all 1's ($FFFF_{16}$) when the data is read back.

Address parity is defined as the **xor** of all the address bits that specify the bank-specific line address, which is (**xor_bit_vector**(PA{39:9}) **xor** PA{6}) for a four-MCU memory system, (**xor_bit_vector**(PA{39:8}) **xor** PA{6}) for a two-MCU memory system, or (**xor**PA{39:6}) for a one-MCU memory system.

E.7.4 Galois Field Multiplication Table

TABLE E-9 Galois Field Multiplication Table, Polynomial 10011

	Multiplicand															
Multiplier	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	c	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	c	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	A

E.7.5 DRAM Syndrome Interpretation

When examining a OpenSPARC T2 DRAM syndrome, first look at TABLE E-10, to find that pattern of zeros and non-zero nibbles in the 16-bit syndrome. If the pattern of the syndrome nibbles is “a0bc” or “ab0c” (only the second or third nibble is zero), two more tables need to be checked to see which nibble is in error (TABLE E-11 or TABLE E-13), and (2) whether there is a multi-nibble error (TABLE E-12 or TABLE E-14).

The other syndrome nibble patterns are fully described in TABLE E-10.

TABLE E-10 Memory Syndrome Summary

S3	S2	S1	S0	Description
0	0	0	0	No error
a	0	b	c	Possible correctable error in N0-N14. See TABLE E-11 and TABLE E-12.
a	b	0	c	Possible correctable error in N15-N29. See TABLE E-13 on page 541 and TABLE E-14 on page 542.
0	d	d	0	Correctable error in N30. Bits to correct are the value “d” in data nibble N30.
0	d	d	d	Correctable error in N31. Bits to correct are the value “d” in data nibble N31.
0	0	0	e	Error in check nibble C0. Bits to correct are the value “e” in check nibble C0.
0	0	e	0	Error in check nibble C1. Bits to correct are the value “e” in check nibble C1.
0	e	0	0	Error in check nibble C2. Bits to correct are the value “e” in check nibble C2.
e	0	0	0	Error in check nibble C3. Bits to correct are the value “e” in check nibble C3.
8_{16}	2_{16}	2_{16}	1_{16}	Poison Indication, or possibly uncorrectable multiple nibble error
F_{16}	F_{16}	F_{16}	F_{16}	Address Parity Error, or possibly uncorrectable multiple nibble error
Other				Uncorrectable multiple nibble error

a, b, c — Non-zero values for syndrome nibbles, potentially different values, or may be same.

d — Non-zero identical values in these three syndrome nibbles.

e — Non-zero value in a single syndrome nibble.

0 — Syndrome value is zero as part of this pattern.

TABLE E-11 Memory Syndrome, Case a0bc, Contents = Nibble in error, Bits in nibble to correct

	synd{3:0}															
synd{7:4}	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
1	*	N0 1_{16}	N1 1_{16}	N2 1_{16}	N3 1_{16}	N4 1_{16}	N5 1_{16}	N6 1_{16}	N7 1_{16}	N8 1_{16}	N9 1_{16}	N10 1_{16}	N11 1_{16}	N12 1_{16}	N13 1_{16}	N14 1_{16}
2	*	N8 2_{16}	N0 2_{16}	N7 2_{16}	N1 2_{16}	N10 2_{16}	N2 2_{16}	N9 2_{16}	N3 2_{16}	N12 2_{16}	N4 2_{16}	N11 2_{16}	N5 2_{16}	N14 2_{16}	N6 2_{16}	N13 2_{16}
3	*	N13 3_{16}	N14 3_{16}	N0 3_{16}	N12 3_{16}	N2 3_{16}	N1 3_{16}	N11 3_{16}	N8 3_{16}	N6 3_{16}	N5 3_{16}	N7 3_{16}	N3 3_{16}	N9 3_{16}	N10 3_{16}	N4 3_{16}
4	*	N12 4_{16}	N8 4_{16}	N3 4_{16}	N0 4_{16}	N11 4_{16}	N7 4_{16}	N4 4_{16}	N1 4_{16}	N14 4_{16}	N10 4_{16}	N5 4_{16}	N2 4_{16}	N13 4_{16}	N9 4_{16}	N6 4_{16}
5	*	N10 5_{16}	N4 5_{16}	N13 5_{16}	N9 5_{16}	N0 5_{16}	N14 5_{16}	N3 5_{16}	N6 5_{16}	N11 5_{16}	N1 5_{16}	N8 5_{16}	N12 5_{16}	N5 5_{16}	N7 5_{16}	N2 5_{16}
6	*	N6 6_{16}	N13 6_{16}	N8 6_{16}	N14 6_{16}	N7 6_{16}	N0 6_{16}	N5 6_{16}	N12 6_{16}	N9 6_{16}	N2 6_{16}	N3 6_{16}	N1 6_{16}	N4 6_{16}	N11 6_{16}	N10 6_{16}
7	*	N5 7_{16}	N11 7_{16}	N9 7_{16}	N10 7_{16}	N12 7_{16}	N6 7_{16}	N0 7_{16}	N4 7_{16}	N2 7_{16}	N8 7_{16}	N14 7_{16}	N13 7_{16}	N7 7_{16}	N1 7_{16}	N3 7_{16}
8	*	N14 8_{16}	N12 8_{16}	N1 8_{16}	N8 8_{16}	N5 8_{16}	N3 8_{16}	N10 8_{16}	N0 8_{16}	N13 8_{16}	N11 8_{16}	N2 8_{16}	N7 8_{16}	N6 8_{16}	N4 8_{16}	N9 8_{16}
9	*	N1 9_{16}	N3 9_{16}	N5 9_{16}	N7 9_{16}	N9 9_{16}	N11 9_{16}	N13 9_{16}	N2 9_{16}	N0 9_{16}	N6 9_{16}	N4 9_{16}	N10 9_{16}	N8 9_{16}	N14 9_{16}	N12 9_{16}
A	*	N11 A_{16}	N10 A_{16}	N6 A_{16}	N4 A_{16}	N8 A_{16}	N13 A_{16}	N1 A_{16}	N9 A_{16}	N5 A_{16}	N0 A_{16}	N12 A_{16}	N14 A_{16}	N2 A_{16}	N3 A_{16}	N7 A_{16}
B	*	N4 B_{16}	N9 B_{16}	N14 B_{16}	N6 B_{16}	N1 B_{16}	N12 B_{16}	N7 B_{16}	N13 B_{16}	N10 B_{16}	N3 B_{16}	N0 B_{16}	N8 B_{16}	N11 B_{16}	N2 B_{16}	N5 B_{16}
C	*	N9 C_{16}	N6 C_{16}	N12 C_{16}	N13 C_{16}	N3 C_{16}	N8 C_{16}	N2 C_{16}	N14 C_{16}	N4 C_{16}	N7 C_{16}	N1 C_{16}	N0 C_{16}	N10 C_{16}	N5 C_{16}	N11 C_{16}
D	*	N3 D_{16}	N7 D_{16}	N11 D_{16}	N2 D_{16}	N6 D_{16}	N10 D_{16}	N14 D_{16}	N5 D_{16}	N1 D_{16}	N13 D_{16}	N9 D_{16}	N4 D_{16}	N0 D_{16}	N12 D_{16}	N8 D_{16}
E	*	N2 E_{16}	N5 E_{16}	N4 E_{16}	N11 E_{16}	N14 E_{16}	N9 E_{16}	N8 E_{16}	N10 E_{16}	N7 E_{16}	N12 E_{16}	N13 E_{16}	N6 E_{16}	N3 E_{16}	N0 E_{16}	N1 E_{16}
F	*	N7 F_{16}	N2 F_{16}	N10 F_{16}	N5 F_{16}	N13 F_{16}	N4 F_{16}	N12 F_{16}	N11 F_{16}	N3 F_{16}	N14 F_{16}	N6 F_{16}	N9 F_{16}	N1 F_{16}	N8 F_{16}	N0 F_{16}

* This table doesn't apply. Look up on Table E-10 on page 538.

Ndd / h_{16} — Top identifies which nibble is in error (in decimal format, dd). Bottom identifies error bits within nibble (in hexadecimal format, h).

TABLE E-12 Memory Syndrome, Case a0bc, Contents = synd{15:12} value

	synd{3:0}															
synd{7:4}	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
1	*	1_{16}	9_{16}	E_{16}	D_{16}	B_{16}	7_{16}	6_{16}	F_{16}	2_{16}	C_{16}	5_{16}	A_{16}	4_{16}	3_{16}	8_{16}
2	*	4_{16}	2_{16}	D_{16}	1_{16}	A_{16}	F_{16}	B_{16}	9_{16}	8_{16}	5_{16}	7_{16}	E_{16}	3_{16}	C_{16}	6_{16}
3	*	5_{16}	B_{16}	3_{16}	C_{16}	1_{16}	8_{16}	D_{16}	6_{16}	A_{16}	9_{16}	2_{16}	4_{16}	7_{16}	F_{16}	E_{16}
4	*	3_{16}	8_{16}	1_{16}	4_{16}	E_{16}	9_{16}	A_{16}	2_{16}	6_{16}	7_{16}	F_{16}	D_{16}	C_{16}	5_{16}	B_{16}
5	*	2_{16}	1_{16}	F_{16}	9_{16}	5_{16}	E_{16}	C_{16}	D_{16}	4_{16}	B_{16}	A_{16}	7_{16}	8_{16}	6_{16}	3_{16}
6	*	7_{16}	A_{16}	C_{16}	5_{16}	4_{16}	6_{16}	1_{16}	B_{16}	E_{16}	2_{16}	8_{16}	3_{16}	F_{16}	9_{16}	D_{16}
7	*	6_{16}	3_{16}	2_{16}	8_{16}	F_{16}	1_{16}	7_{16}	4_{16}	C_{16}	E_{16}	D_{16}	9_{16}	B_{16}	A_{16}	5_{16}
8	*	C_{16}	6_{16}	4_{16}	3_{16}	D_{16}	2_{16}	E_{16}	8_{16}	B_{16}	F_{16}	9_{16}	1_{16}	5_{16}	7_{16}	A_{16}
9	*	D_{16}	F_{16}	A_{16}	E_{16}	6_{16}	5_{16}	8_{16}	7_{16}	9_{16}	3_{16}	C_{16}	B_{16}	1_{16}	4_{16}	2_{16}
A	*	8_{16}	4_{16}	9_{16}	2_{16}	7_{16}	D_{16}	5_{16}	1_{16}	3_{16}	A_{16}	E_{16}	F_{16}	6_{16}	B_{16}	C_{16}
B	*	9_{16}	D_{16}	7_{16}	F_{16}	C_{16}	A_{16}	3_{16}	E_{16}	1_{16}	6_{16}	B_{16}	5_{16}	2_{16}	8_{16}	4_{16}
C	*	F_{16}	E_{16}	5_{16}	7_{16}	3_{16}	B_{16}	4_{16}	A_{16}	D_{16}	8_{16}	6_{16}	C_{16}	9_{16}	2_{16}	1_{16}
D	*	E_{16}	7_{16}	B_{16}	A_{16}	8_{16}	C_{16}	2_{16}	5_{16}	F_{16}	4_{16}	3_{16}	6_{16}	D_{16}	1_{16}	9_{16}
E	*	B_{16}	C_{16}	8_{16}	6_{16}	9_{16}	4_{16}	F_{16}	3_{16}	5_{16}	D_{16}	1_{16}	2_{16}	A_{16}	E_{16}	7_{16}
F	*	A_{16}	5_{16}	6_{16}	B_{16}	2_{16}	3_{16}	9_{16}	C_{16}	7_{16}	1_{16}	4_{16}	8_{16}	E_{16}	D_{16}	F_{16}

* This table doesn't apply. Look up on Table E-10 on page 538.

h_{16} — Specifies the value synd{15:12} must have in hexadecimal; otherwise, this is a multi-nibble error.

TABLE E-13 Memory Syndrome, Case ab0c, Contents = Nibble in error, Bits in nibble to correct

	synd{3:0}															
synd{11:8}	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
1	*	N15 1_{16}	N16 1_{16}	N17 1_{16}	N18 1_{16}	N19 1_{16}	N20 1_{16}	N21 1_{16}	N22 1_{16}	N23 1_{16}	N24 1_{16}	N25 1_{16}	N26 1_{16}	N27 1_{16}	N28 1_{16}	N29 1_{16}
2	*	N23 2_{16}	N15 2_{16}	N22 2_{16}	N16 2_{16}	N25 2_{16}	N17 2_{16}	N24 2_{16}	N18 2_{16}	N27 2_{16}	N19 2_{16}	N26 2_{16}	N20 2_{16}	N29 2_{16}	N21 2_{16}	N28 2_{16}
3	*	N28 3_{16}	N29 3_{16}	N15 3_{16}	N27 3_{16}	N17 3_{16}	N16 3_{16}	N26 3_{16}	N23 3_{16}	N21 3_{16}	N20 3_{16}	N22 3_{16}	N18 3_{16}	N24 3_{16}	N25 3_{16}	N19 3_{16}
4	*	N27 4_{16}	N23 4_{16}	N18 4_{16}	N15 4_{16}	N26 4_{16}	N22 4_{16}	N19 4_{16}	N16 4_{16}	N29 4_{16}	N25 4_{16}	N20 4_{16}	N17 4_{16}	N28 4_{16}	N24 4_{16}	N21 4_{16}
5	*	N25 5_{16}	N19 5_{16}	N28 5_{16}	N24 5_{16}	N15 5_{16}	N29 5_{16}	N18 5_{16}	N21 5_{16}	N26 5_{16}	N16 5_{16}	N23 5_{16}	N27 5_{16}	N20 5_{16}	N22 5_{16}	N17 5_{16}
6	*	N21 6_{16}	N28 6_{16}	N23 6_{16}	N29 6_{16}	N22 6_{16}	N15 6_{16}	N20 6_{16}	N27 6_{16}	N24 6_{16}	N17 6_{16}	N18 6_{16}	N16 6_{16}	N19 6_{16}	N26 6_{16}	N25 6_{16}
7	*	N20 7_{16}	N26 7_{16}	N24 7_{16}	N25 7_{16}	N27 7_{16}	N21 7_{16}	N15 7_{16}	N19 7_{16}	N17 7_{16}	N23 7_{16}	N29 7_{16}	N28 7_{16}	N22 7_{16}	N16 7_{16}	N18 7_{16}
8	*	N29 8_{16}	N27 8_{16}	N16 8_{16}	N23 8_{16}	N20 8_{16}	N18 8_{16}	N25 8_{16}	N15 8_{16}	N28 8_{16}	N26 8_{16}	N17 8_{16}	N22 8_{16}	N21 8_{16}	N19 8_{16}	N24 8_{16}
9	*	N16 9_{16}	N18 9_{16}	N20 9_{16}	N22 9_{16}	N24 9_{16}	N26 9_{16}	N28 9_{16}	N17 9_{16}	N15 9_{16}	N21 9_{16}	N19 9_{16}	N25 9_{16}	N23 9_{16}	N29 9_{16}	N27 9_{16}
A	*	N26 A_{16}	N25 A_{16}	N21 A_{16}	N19 A_{16}	N23 A_{16}	N28 A_{16}	N16 A_{16}	N24 A_{16}	N20 A_{16}	N15 A_{16}	N27 A_{16}	N29 A_{16}	N17 A_{16}	N18 A_{16}	N22 A_{16}
B	*	N19 B_{16}	N24 B_{16}	N29 B_{16}	N21 B_{16}	N16 B_{16}	N27 B_{16}	N22 B_{16}	N28 B_{16}	N25 B_{16}	N18 B_{16}	N15 B_{16}	N23 B_{16}	N26 B_{16}	N17 B_{16}	N20 B_{16}
C	*	N24 C_{16}	N21 C_{16}	N27 C_{16}	N28 C_{16}	N18 C_{16}	N23 C_{16}	N17 C_{16}	N29 C_{16}	N19 C_{16}	N22 C_{16}	N16 C_{16}	N15 C_{16}	N25 C_{16}	N20 C_{16}	N26 C_{16}
D	*	N18 D_{16}	N22 D_{16}	N26 D_{16}	N17 D_{16}	N21 D_{16}	N25 D_{16}	N29 D_{16}	N20 D_{16}	N16 D_{16}	N28 D_{16}	N24 D_{16}	N19 D_{16}	N15 D_{16}	N27 D_{16}	N23 D_{16}
E	*	N17 E_{16}	N20 E_{16}	N19 E_{16}	N26 E_{16}	N29 E_{16}	N24 E_{16}	N23 E_{16}	N25 E_{16}	N22 E_{16}	N27 E_{16}	N28 E_{16}	N21 E_{16}	N18 E_{16}	N15 E_{16}	N16 E_{16}
F	*	N22 F_{16}	N17 F_{16}	N25 F_{16}	N20 F_{16}	N28 F_{16}	N19 F_{16}	N27 F_{16}	N26 F_{16}	N18 F_{16}	N29 F_{16}	N21 F_{16}	N24 F_{16}	N16 F_{16}	N23 F_{16}	N15 F_{16}

* This table doesn't apply. Look up on Table E-10 on page 538.

Ndd / h_{16} — Top identifies which nibble is in error (in decimal format, dd). Bottom identifies error bits within nibble (in hexadecimal format, h).

TABLE E-14 Memory Syndrome, Case ab0c, Contents = synd{15:12} value

	synd{3:0}															
synd{11:8}	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
1	*	1_{16}	9_{16}	E_{16}	D_{16}	B_{16}	7_{16}	6_{16}	F_{16}	2_{16}	C_{16}	5_{16}	A_{16}	4_{16}	3_{16}	8_{16}
2	*	4_{16}	2_{16}	D_{16}	1_{16}	A_{16}	F_{16}	B_{16}	9_{16}	8_{16}	5_{16}	7_{16}	E_{16}	3_{16}	C_{16}	6_{16}
3	*	5_{16}	B_{16}	3_{16}	C_{16}	1_{16}	8_{16}	D_{16}	6_{16}	A_{16}	9_{16}	2_{16}	4_{16}	7_{16}	F_{16}	E_{16}
4	*	3_{16}	8_{16}	1_{16}	4_{16}	E_{16}	9_{16}	A_{16}	2_{16}	6_{16}	7_{16}	F_{16}	D_{16}	C_{16}	5_{16}	B_{16}
5	*	2_{16}	1_{16}	F_{16}	9_{16}	5_{16}	E_{16}	C_{16}	D_{16}	4_{16}	B_{16}	A_{16}	7_{16}	8_{16}	6_{16}	3_{16}
6	*	7_{16}	A_{16}	C_{16}	5_{16}	4_{16}	6_{16}	1_{16}	B_{16}	E_{16}	2_{16}	8_{16}	3_{16}	F_{16}	9_{16}	D_{16}
7	*	6_{16}	3_{16}	2_{16}	8_{16}	F_{16}	1_{16}	7_{16}	4_{16}	C_{16}	E_{16}	D_{16}	9_{16}	B_{16}	A_{16}	5_{16}
8	*	C_{16}	6_{16}	4_{16}	3_{16}	D_{16}	2_{16}	E_{16}	8_{16}	B_{16}	F_{16}	9_{16}	1_{16}	5_{16}	7_{16}	A_{16}
9	*	D_{16}	F_{16}	A_{16}	E_{16}	6_{16}	5_{16}	8_{16}	7_{16}	9_{16}	3_{16}	C_{16}	B_{16}	1_{16}	4_{16}	2_{16}
A	*	8_{16}	4_{16}	9_{16}	2_{16}	7_{16}	D_{16}	5_{16}	1_{16}	3_{16}	A_{16}	E_{16}	F_{16}	6_{16}	B_{16}	C_{16}
B	*	9_{16}	D_{16}	7_{16}	F_{16}	C_{16}	A_{16}	3_{16}	E_{16}	1_{16}	6_{16}	B_{16}	5_{16}	2_{16}	8_{16}	4_{16}
C	*	F_{16}	E_{16}	5_{16}	7_{16}	3_{16}	B_{16}	4_{16}	A_{16}	D_{16}	8_{16}	6_{16}	C_{16}	9_{16}	2_{16}	1_{16}
D	*	E_{16}	7_{16}	B_{16}	A_{16}	8_{16}	C_{16}	2_{16}	5_{16}	F_{16}	4_{16}	3_{16}	6_{16}	D_{16}	1_{16}	9_{16}
E	*	B_{16}	C_{16}	8_{16}	6_{16}	9_{16}	4_{16}	F_{16}	3_{16}	5_{16}	D_{16}	1_{16}	2_{16}	A_{16}	E_{16}	7_{16}
F	*	A_{16}	5_{16}	6_{16}	B_{16}	2_{16}	3_{16}	9_{16}	C_{16}	7_{16}	1_{16}	4_{16}	8_{16}	E_{16}	D_{16}	F_{16}

* This table doesn't apply. Look up on Table E-10 on page 538.

h_{16} — Specifies the value synd{15:12} must have in hexadecimal; otherwise, this is a multi-nibble error.

E.8 Data Poisoning

Data poisoning is the practice of marking known corrupt data with bad ECC so that any later access will get an ECC error. This is normally done when data is transferred from one cache or memory to another, to keep track that the data is

corrupted. If data is not marked with bad ECC, this will lead to silent data corruption when an unsuspecting virtual processor or DMA accesses the old stale copy of the data that still has “good” ECC.

E.8.1 ECC Conversion of UEs as Poison Source

Another source of poison is uncorrectable ECC errors that occur when data is being transferred to a structure that has a different ECC (or parity) encoding. Thus, whenever data is transferred between memory, L2, and the L1 caches, and the source data has an uncorrectable error, the destination will be marked as poison, since the ECC code cannot be transferred intact.

E.8.2 Poisoning L1

The L1 caches (I and D) have only parity protection, so poisoning is implemented by marking the corrupt data (not tag) with a parity error (which is indistinguishable from other parity errors, except that there is also an L2 error with the same address). Since data is always transferred into the L1 in 16-byte chunks, poison in the L1 will always be in aligned 16-byte chunks.

Even though the L1 caches never have dirty data, it is still necessary to receive the corrupt data and install it into the cache, in order to maintain consistency between the L1 tags and the L2 directory.

If the L1 just dropped the corrupt data without installing into the cache, there are scenarios where the L1 tags and L2 directory get out of sync and lose coherency, thus causing silent data corruption (because an invalidate didn’t work).

E.8.3 Poisoning L2

L2 poisoning is implemented by flipping "all" of the 7 checkbits foreach 32bit data word that is corrupt, which means a syndrome of $7F_{16}$ is most likely a poison indication.

L2 ECC (and thus poison) has a 4-byte granularity, so every 4-byte word of uncorrectable data that is written into the cache will have a poison indication. However, all transfers out of the cache are done in 16-byte chunks, so DMA reads, L1 misses, and writebacks to memory will all increase poison/error indications to 16-byte granularity to the recipient, but the L2 is unmodified if it keeps the data.

E.8.3.1 Partial Write Details

Subword writes are read-modify-write, with any ECC correction after the read. If the word read has an uncorrectable error, however, the write is aborted, which leaves the original uncorrectable error intact. Thus, subline writes will not convert uncorrectable errors into poison.

E.8.4 Poisoning Memory

Memory poisoning is implemented by flipping four specific checkbits, specifically C{15, 9, 5, 0} which means that a syndrome of 8221_{16} is most likely a poison indication. This syndrome was chosen because no single nibble error can convert a poison syndrome into a correctable error. A side effect of this is that a minimum of a triple nibble error is needed to "accidentally" generate a failing syndrome of 8221_{16}, so the possibility that a syndrome of 8221_{16} was generated by anything but poison is infinitesimal.

Memory ECC (and thus poison) has a 16-byte granularity, so every 16-byte chunk of uncorrectable data that is written to memory will have a poison indication.

E.8.5 Erasing Poison

Poison can be erased by overwriting it with good data, in such a way that no subword L2 writes occur and no writebacks occur to main memory while the poison is partially erased.

Alternatively, `ASI_BLK_INIT_ST` stores can be used to force poison to be overwritten by causing the entire line to be zeroed out if the line was faulted back to memory.

APPENDIX **F**

JTAG (IEEE 1149.1) Scan Interface

F.1 System JTAG Commands

OpenSPARC T2 supports the following JTAG commands. Note that many of the capabilities that are available through JTAG cannot be used in a running system (for example, accessing normal scan registers).

TABLE F-1 Available System JTAG Commands *(1 of 4)*

Command	Encoding	Description
TAP_EXTEST	00_{16}	Selects BOUNDARY_SCAN_REGISTER.
TAP_IDCODE	01_{16}	Selects IDCODE DR.
TAP_SAMPLE_PRELOAD	02_{16}	Selects BOUNDARY_SCAN_REGISTER.
TAP_HIGHZ	03_{16}	Used for MIO I/O cells.
TAP_CLAMP	04_{16}	Used for MIO I/O cells.
TAP_EXTEST_PULSE	05_{16}	IEEE 1149.6 compliant.
TAP_EXTEST_TRAIN	06_{16}	IEEE 1149.6 compliant.
TAP_CREG_ADDR	08_{16}	Stores address to be used for system access to control register.
TAP_CREG_WDATA	09_{16}	Stores data to be used for system access to control register.
TAP_CREG_RDATA	$0A_{16}$	Captures data from system access.
TAP_NCU_WRITE	$0C_{16}$	Initiates write to system control register.
TAP_NCU_READ	$0D_{16}$	Initiates read from system control register.
TAP_NCU_WADDR	$0E_{16}$	Combination of TAP_CREG_ADDR and TAP_NCU_WRITE.
TAP_NCU_WDATA	$0F_{16}$	Combination of TAP_CREG_WDATA and TAP_NCU_WRITE.
TAP_NCU_RADDR	10_{16}	Combination of TAP_CREG_ADDR and TAP_NCU_READ.
TAP_MBIST_CLKSTPEN	13_{16}	Enables clock stop for MBIST via cycle counter.

TABLE F-1 Available System JTAG Commands *(2 of 4)*

Command	Encoding	Description
TAP_MBIST_BYPASS	14_{16}	Selects engines to be excluded from MBIST operation via mbist_bypass data register.
TAP_MBIST_MODE	15_{16}	Specify serial/parallel, diagnostic mode, or bist/bisi modes via mbist_mode data register.
TAP_MBIST_START	16_{16}	Initiate MBIST.
TAP_MBIST_RESULT	18_{16}	Query 2-bit done/fail register: and/or of all 48 JTAG visible MBIST engines.
TAP_MBIST_DIAG	19_{16}	Run MBIST on one array; MBIST engine and arrays are data register.
TAP_MBIST_GETDONE	$1A_{16}$	Query 48-bit done data register, one bit per JTAG visible MBIST engine.
TAP_MBIST_GETFAIL	$1B_{16}$	Query 48-bit fail data register, one bit per JTAG visible MBIST engine.
TAP_DMO_ACCESS	$1C_{16}$	Set DMO mode; enables DMO logic and package pins.
TAP_DMO_CLEAR	$1D_{16}$	Clears DMO mode.
TAP_DMO_CONFIG	$1E_{16}$	Access 32-bit DMO configuration register.
TAP_MBIST_ABORT	$1F_{16}$	Stop any MBIST activity and reset MBIST controls.
TAP_FUSE_READ	28_{16}	Shift out 32 bits selected by ROW_ADDR; selects eFuse DR.
TAP_FUSE_BYPASS_DATA	29_{16}	Provides user data directly to EFU; selects eFuse DR.
TAP_FUSE_BYPASS	$2A_{16}$	Starts EFU control using bypass data provided by user.
TAP_FUSE_ROW_ADDR	$2B_{16}$	Shift in 7-bit row address for EFU access; selects eFuse ROW ADDRESS DR.
TAP_FUSE_COL_ADDR	$2C_{16}$	Shift in 5-bit column address for EFU programming; selects eFuse COLUMN ADDRESS DR.
TAP_FUSE_READ_MODE	$2D_{16}$	Configures EFU with 3 bits for EFU access; selects eFuse READ MODE DR.
TAP_FUSE_DEST_SAMPLE	$2E_{16}$	Samples EFU destination redundancy value from the destination specified.
TAP_FUSE_RVCLR	$2F_{16}$	Access 6-bit redundancy value clear register.
TAP_SPCTHR0_SHSCAN	30_{16}	Samples thread 0 for all available strands.
TAP_SPCTHR1_SHSCAN	31_{16}	Samples thread 1 for all available strands.
TAP_SPCTHR2_SHSCAN	32_{16}	Samples thread 2 for all available strands.
TAP_SPCTHR3_SHSCAN	33_{16}	Samples thread 3 for all available strands
TAP_SPCTHR4_SHSCAN	34_{16}	Samples thread 4 for all available strands.
TAP_SPCTHR5_SHSCAN	35_{16}	Samples thread 5 for all available strands.

TABLE F-1 Available System JTAG Commands *(3 of 4)*

Command	Encoding	Description
TAP_SPCTHR6_SHSCAN	36_{16}	Samples thread 6 for all available strands.
TAP_SPCTHR7_SHSCAN	37_{16}	Samples thread 7 for all available strands.
TAP_L2T_SHSCAN	38_{16}	Samples specified error registers in the 8 L2 Tags.
TAP_CLOCK_SSTOP	40_{16}	Soft stop of clocks; strands only.
TAP_CLOCK_HSTOP	41_{16}	Hard stop of clocks.
TAP_CLOCK_START	42_{16}	Start clocks.
TAP_CLOCK_DOMAIN	43_{16}	Specify entry clock domain for stopping/starting clocks.
TAP_CLOCK_STATUS	44_{16}	2-bit status indicating if clock stop/start routine finished.
TAP_CLOCK_DELAY	45_{16}	7-bits specifying up to 128 cycle delay between successive clk_stop signals.
TAP_CORE_SEL	46_{16}	8-bit register to specify target SPC cores for clock operations.
TAP_DE_COUNT	48_{16}	Access 32-bit debug event counter.
TAP_CYCLE_COUNT	49_{16}	Access 64-bit reset/cycle counter.
TAP_TCU_DCR	$4A_{16}$	Access 3-bit TCU Debug Control register.
TAP_CORE_RUN_STATUS	$4C_{16}$	Access 64-bit CMP Strand Running Status register.
TAP_DOSS_ENABLE	$4D_{16}$	Access 64-bit disable overlap or single step completion.
TAP_DOSS_MODE	$4E_{16}$	Specify either disable overlap or single step mode; 1 = Enable, 0 = Single step if set to 1, disable overlap if set to 0.
TAP_SS_REQUEST	$4F_{16}$	Pulse single-step request signal.
TAP_DOSS_STATUS	50_{16}	1-bit status for disable overlap or single-step completion.
TAP_CS_MODE	51_{16}	Specify cycle-step mode. 1-bit register set to 1 to enable, uses cycle counter for cycle-step operation.
TAP_CS_STATUS	52_{16}	Read 1-bit status indicating cycle stepping has completed.
TAP_L2_ADDR	58_{16}	Load L2 address (to be written to or read from).
TAP_L2_WRDATA	59_{16}	Load L2 write data.
TAP_L2_WR	$5A_{16}$	Initiate write to L2; wrdata to addr.
TAP_L2_RD	$5B_{16}$	Initiate read from L2 at addr and receive L2 data.
TAP_LBIST_START	60_{16}	Initiate Logic BIST.
TAP_LBIST_BYPASS	61_{16}	Bypass Logic BIST for specified strands; 1-bit per strand.
TAP_LBIST_MODE	62_{16}	Control program mode: parallel/serial modes.
TAP_LBIST_ACCESS	63_{16}	Place one Logic BIST controller between TDI-TDO.

TABLE F-1 Available System JTAG Commands *(4 of 4)*

Command	Encoding	Description
TAP_LBIST_GETDONE	64_{16}	Determine if Logic BIST is done across all selected strands.
TAP_LBIST_ABORT	65_{16}	Abort any Logic BIST currently in progress.
TAP_SERSCAN	80_{16}	Access all or 31 internal scan chains (excluding SerDes scan chain); selects internal scan flops as data register.
TAP_CHAINSEL	81_{16}	Select all or one of the 32 chains for serial scan mode using CHAIN SELECT DR.
TAP_MT_ACCESS	82_{16}	Enables Macro Test mode for JTAG scan.
TAP_MT_CLEAR	83_{16}	Clears Macro Test mode.
TAP_MT_SCAN	84_{16}	Similar to TAP_SERSCAN but drive TCK onto clock tree for pulse capture during RTI state.
TAP_STCI_ACCESS	90_{16}	Enables STCI mode.
TAP_STCI_CLEAR	91_{16}	Clears STCI mode.
TAP_JTPOR_ACCESS	$A0_{16}$	Enables JTAG access window during POR sequence.
TAP_JTPOR_CLEAR	$A1_{16}$	Clears JTAG access window during POR sequence.
TAP_JTPOR_STATUS	$A1_{16}$	JTAG access window status. 1_2 if window is active.
TAP_SCKBYP_ACCESS	$A3_{16}$	Enables bypass for SSI lock time counter in NCU for tester.
TAP_SCKBYP_CLEAR	$A4_{16}$	Clears bypass for SSI lock time counter in NCU for system.
TAP_BYPASS	FF_{16}	Selects Bypass register.

The detailed description of these commands and the exact protocol of how to communicate to the JTAG interface (TAP) plus all the other JTAG commands are documented in the OpenSPARC T2 TCU Microarchitecture Specification.

F.2 JTAG CREG Interface

The CREG interface in the TAP allows access to much of the internal state of OpenSPARC T2 with minimal invasiveness. At a high level, the JTAG CREG interface, represented in TAP_CREG or TAP_NCU type JTAG instructions, allows read and write access to any I/O-addressable location in OpenSPARC T2 except 85_{16} TCU registers. This includes 80_{16} NCU; 83_{16} CCU; 84_{16} MCU; 86_{16} DBG; 88_{16} DMU; 89_{16} RST; 90_{16} ASI; $A0_{16}$–BF_{16} L2; and FF_{16} SSI. All 85_{16} TCU registers can be accessed from NCU via UCB. Note that SPC shscan and L2 accesses are done with different JTAG instructions.

F.2.1 I/O Mapped Register Accesses

The CREG interface can be used to read or write any I/O-addressable location in OpenSPARC T2's address space. All accesses are 8-byte accesses.

The Unit Control Bus interface is a protocol for transmission of packets via the NCU between units. It is implemented inside the TCU and allows access via JTAG to I/O-mapped registers. A register's address and data in the case of writes are loaded via JTAG into holding registers in the TCU. The TCU then uses its UCB interface to communicate to the NCU, which puts the new transaction (packet) into the data flow. The interface allows both reading and writing. On N2, UCB access through the crossbar to the l2 and strands is not available, so access to the L2 is done via a separate interface between the TCU and the SIU.

For a write, a 40-bit address and 64 bits of data must be provided by JTAG to the UCB. For a read, a 40-bit address is needed, with the data received from the NCU captured into a register in the TCU. To implement a read, a sentinel bit is used since the exact timing of the read return is not deterministic. The system is only allowed to have one read outstanding at one time. There is no protection built in against this; adherence is left to the user.

F.2.1.1 JTAG Instructions Used to Access the UCB

The following descriptions are excerpts from the OpenSPARC T1 DFT specification and the OpenSPARC T1 DFT User's Guide but have been ported to OpenSPARC T2.

TAP_CREG_ADDR

Load System Address: Causes a 40-bit address register to become accessible from TDI. The target system address is loaded during shift-DR. On update-DR a transfer occurs from the TCK domain to a 40-bit holding register in the IO CLK domain.In shift-DR, LSB of CSR `addr` is shifted in first and MSB is shifted in last.

TAP_CREG_WDATA

Load System Write Data: Causes a 64-bit data register to become accessible from TDI, into which the data for the specified system address is loaded during shift-DR. On update-DR a transfer occurs from the TCK domain to a 64-bit holding register in the I/O CLK domain. In shift-DR, LSB of write data is shifted in first and MSB is shifted in last.

TAP_CREG_RDATA

Load System Read Data: Causes a 65-bit data register to become accessible from TDO. The 65th bit is used as a sentinel to allow driver software to synchronize with the read operation. While the read is outstanding, the sentinel bit remains zero. Once the NCU has returned valid data, then the read is complete and the sentinel bit is set to 1. To use this, the JTAG is kept in shift-DR and TCK is clocked until the TDO

reads a 1; this indicates the sentinel bit has been set. When the sentinel bit becomes 1, the next 64 bits shifted out are the valid read data.In shift-DR, LSB of CSR data (that is, CSR content) is shifted out first and MSB is shifted out last.

The TCU can only issue a single access at a given time to the NCU. The user is responsible for ensuring that this is the case. Note too that the TCU does not report erroneous reads made to the NCU. Therefore, the driver software should time out on a read, assuming an error if this occurs.

TAP_NCU_WRITE

Initiate Write Transaction: Causes a write transaction to be initiated on update-IR.

TAP_NCU_READ

Initiate Read Transaction: Causes a read transaction to be initiated on update-IR.

TAP_NCU_WADDR

Load System Address and Initiate Write Transaction: Causes a 40-bit address register to become accessible from TDI. The target system address is loaded during shift-DR. On update-DR a transfer occurs from the TCK domain to a 40-bit holding register in the IO_CLK domain. In the cycle after the transfer is complete, the contents of the address register are forwarded to the UCB interface and a write transaction is initiated. This instruction is a combination of TAP_CREG_ADDR and TAP_NCU_WRITE.

TAP_NCU_WDATA

Load Write Data and Initiate Write Transaction: Causes a 64-bit data register to become accessible from TDI, into which the data for the specified system address is loaded during shift-DR. On update-DR a transfer occurs from the TCK domain to a 64-bit holding register in the IO_CLK domain. In the cycle after the transfer is complete, the contents of the address register and data register are forwarded to the UCB interface to initiate a write transaction. This instruction is a combination of TAP_CREG_WDATA and TAP_NCU_WRITE.

TAP_NCU_RADDR

Load System Address and Initiate Read Transaction: Causes a 40-bit address register to become accessible from TDI. The target system address is loaded during shift-DR. On update-DR a transfer occurs from the TCK domain to a 40-bit holding register in the IO_CLK domain. In the cycle after the transfer is complete, the contents of the address register are forwarded to the UCB interface and a read transaction is initiated. This instruction is a combination of TAP_CREG_ADDR and TAP_NCU_READ.

F.2.1.2 Expected Data and Address Format

The data to be written is 64 bits in length. A 40-bit address is also loaded into the UCB address register.

F.2.1.3 Accesses to Unsupported I/O Addresses

It is possible to specify an illegal or unsupported I/O address in the CREG. Most locations will silently drop writes to unsupported addresses and will return NACK on reads to unsupported addresses, which is ignored by the TAP.

F.2.2 TAP Access to CPU ASI Registers

On OpenSPARC T2, JTAG UCB access to the ASI registers mapped to both CPU and L2 are not available. However, access for all CMP data ASI registers mapped to NCU is available via UCB (TAP_CREG_ and TAP_NCU_ instructions).

Threads in each strand are virtual processors; for those CMP registers specifying physical cores, each physical core is assigned 8 bits in a 64-bit register; allowed values are $1111\ 1111_2$ and $0000\ 0000_2$. The assigned 8 bits are 63:56 = strand 7; 55:48 = strand 6; 47:40 = strand 5; 39:32 = strand 4; 31:24 = strand 3; 23:16 = strand 2; 15:8 = strand 1; 7:0 = strand 0.

Refer to section of 24.4 of the PRM for the ASI registers mapped to NCU and the algorithm for mapping ASI accesses to I/O accesses.

F.3 JTAG Access to Memory

On OpenSPARC T2, JTAG access to L2 is done differently from UCB access through crossbar. Access to the L2 is done via a separate interface between the TCU and the SIU, using TAP_L2 JTAG instructions.

F.3.1 JTAG L2 Access Registers

It is possible to write and read the L2 addresses while the chip is running using JTAG. The L2_ADDR register is accessed via TAP_L2_ADDR; the L2_WRITE_DATA register is accessed via TAP_L2_WRDATA; and the L2_READ_DATA register is accessed via TAP_L2_RD as described below. The L2_WRITE_DATA and L2_READ_DATA registers are the same physical register.

JTAG user should provide physical address that is 8 byte aligned. Irrespective of the original content of bit 2, SIU will force bit 2 = 0.

TABLE F-2 L2 Access Registers

Register	JTAG Instruction	Bits 64:1	Bit 0
L2_ADDR{64:0}	TAP_L2_ADDR	bit 64 =1: JTAG access bits 63:57 = 000 0001 for read request bits 63:57 = 000 0010 for write request bits 56:41 = Unused bits 40:1 = Physical address (8-byte boundary)	Ignored
L2_WRITE_DATA{64:0}	TAP_L2_WRDATA	bits 64:1 = 8-bytes of data to write to L2.	Ignored
L2_READ_DATA{64:0}	TAP_L2_RD	bits 64:1 = 8-bytes of data returned from L2.	1 = Data Valid

F.3.1.1 Memory Write

To write the L2, an address and data must be loaded via JTAG using TAP_L2_ADDR and TAP_L2_WRDATA, followed by TAP_L2_WR. When the TAP_L2_WR instruction is active, the `run-test-idle` state (C_{16}) of the TAP state machine is used to transfer the address and data to the L2 and at least 128 TCK clocks must be cycled while in RTI state for the transfer to complete.

F.3.1.2 Memory Read

A read is accomplished by loading an address using TAP_L2_ADDR followed by a TAP_L2_RD. When the TAP_L2_RD instruction is active, only 64 TCK clocks need be cycled while in RTI to transfer the address to the L2. Then, repeated passes through capture-DR and shift-DR should be used to retrieve the data returned by the L2. Valid data is indicated during TAP_L2_RD at TDO in the shift-DR state by the presence of a leading 1 (bit 0 of the 65-bit L2_READ_DATA register); otherwise, another pass through capture-DR should be implemented without intervening visits to `run-test-idle`. Memory read is nondestructive and has no functional side effect.

Further details on the Addr (Header) and Data (Payload) can be found in the SOC RAS specification. Only one write or read may be outstanding at any time. Also, since non-JTAG logic is used, the POR reset sequence should be performed before using this feature (or at least the POR1 section of the reset sequence).

F.4 JTAG Private Instruction Accessible and Software Accessible Registers

This section describes the JTAG private access registers that are also accessible by software through the NCU. Only the minimal subset of the control registers required for chip debug and testing is accessible by software through the NCU. For joint access between JTAG private and software, JTAG private access will have priority. These registers are local TCU CSRs that the NCU UCB can access.These registers cannot be accessed using JTAG UCB protocol (like TAP_CREG or TAP_NCU type JTAG instructions), but can be accessed using JTAG instructions TAP_DE_COUNT, TAP_CYCLE_COUNT, TAP_TCU_DCR. Note that in PRM section 29.4.4, the trigout_reg bit is set when DCR = 100, 101, 110, or 111, so there is not a separate JTAG instruction to set this bit.

Note | The TCU Debug Control register, TCU Cycle Counter register, TCU Debug Event Counter register, and TCU Trigger Output register also fall in this category of registers and are described in section 29.4 of the PRM.

TABLE F-3 MBIST Mode Register (85 0000 0000_{16})

Bit	Field	Initial Value	R/W	Description
63:4	—	X	RO	*Reserved*
3	loop	0	RW	0 = No loop; 1 = Loop.
2	diagnostic	0	RW	Diagnostic mode if set.
1	bisi	0(1 - See Note below)	RW	BISI if 1; BIST if 0.
0	parallel	0	RW	Parallel mode if 1.

TABLE F-4 MBIST Bypass Register (85 0000 0008_{16}

Bit	Field	Initial Value	R/W	Description
63:48	—	X	RO	*Reserved*
47:0	bypass	0_{16} ($FFFFFFFF FFFF_{16}$ - See Note below)	RW	MBIST bypass.

Note | In the case of the Mbist Mode and Bypass Registers, the default value is overwritten during the power on reset sequence.BISI Enable bit of Mbist Mode Register is written as 1_2 by logic during powere on reset sequence and this bit stays as 1_2 until it is programmed otherwise.The value of the MBIST Bypass register will depend on the core and bank available fuse values after POR1; if there is no partial mode, then this register will be all 1's in bits 47:0

TABLE F-5 MBIST Start Register (85 0000 0010_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	X	W	*Reserved*
0	mbist_ start	0	W	Starts MBIST sequence when written to 1.

TABLE F-6 MBIST Abort Register (85 0000 0018_{16})

Bit	Field	Initial Value	R/W	Description
63:1	—	X	W	*Reserved*
0	mbist_abort	0	W	Aborts MBIST sequence when written to 1.

TABLE F-7 MBIST Result Register (85 0000 0020_{16})

Bit	Field	Initial Value	R/W	Description
63:2	—	X	RO	*Reserved*
1	mbist_done	0	RO	MBIST done.
0	mbist_fail	0	RO	MBIST fail.

TABLE F-8 MBIST Done Register (85 0000 0028_{16})

Bit	Field	Initial Value	R/W	Description
63:48	—	X	RO	*Reserved*
47:0	mbist_done	0_{16}	RO	MBIST Done bits.

TABLE F-9 MBIST Fail Register (85 0000 0030_{16}))

Bit	Field	Initial Value	R/W	Description
63:48	—	X	RO	*Reserved*
47:0	mbist_fail	0_{16}	RO	MBIST Fail bits.

TABLE F-10 MBIST Start WMR Register ($85\ 0000\ 0038_{16}$)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	W	*Reserved*
0	wrm_start	0_{16}	W	Starts MBIST sequence when written to 1, but delayed until after the next warm reset occurs.

TABLE F-11 LBIST Mode Register ($85\ 0000\ 0040_{16}$)

Bit	Field	Initial Value	R/W	Description
63:2	—	X	RO	*Reserved*
1	program	0	RW	Program mode if 1.
0	parallel	0	RW	Parallel mode if 1.

TABLE F-12 LBIST Bypass Register ($85\ 0000\ 0048_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	—	X	RO	*Reserved*
7:0	lbist_bypass	0_{16}	RW	LBIST bypass.

TABLE F-13 LBIST Start Register ($85\ 0000\ 0050_{16}$)

Bit	Field	Initial Value	R/W	Description
63:1	—	X	RO	*Reserved*
0	lbist_start	0	RW	Starts LBIST sequence when written to 1.

TABLE F-14 LBIST Done Register ($85\ 0000\ 0058_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	—	X	RO	*Reserved*
7:0	lbist_done	0_{16}	RW	LBIST done status.

TABLE F-15 CLKSTOP_DELAY Register ($85\ 0000\ 0120_{16}$)

Bit	Field	Initial Value	R/W	Description
63:8	—	X	RO	*Reserved*
7:0	clkstp_delay	0_{16}	RW	Clock stop delay counter.

F.5 Shadow Scan Chains

Shadow scan access is provided in OpenSPARC T2 on SPARC physical core and L2 Error registers. During a shadow-scan operation, JTAG is used to capture the desired values into the shadow scan register. The contents are then scanned-out via TDO. Both the core and L2 tag shadow scan registers can only be read; any value scanned into them will be overwritten. Shadow scan is nondestructive and can happen even when the chip is running in functional mode.

F.5.1 SPARC Shadow Scan

Shadow scan for the strands is controlled via JTAG. The contents to be captured in the shadow scan are shown in TABLE F-1.

Each physical OpenSPARC T2 SPARC virtual processor supports the ability to capture a subset of each strand's state for inspection via a shadow scan facility. The shadow scan is invoked by JTAG commands shown in TABLE F-1.

The TCU continually specifies a strand ID to each physical OpenSPARC T2 SPARC core. In response, the physical core atomically captures the state as described in TABLE 29-1 in a scan string. The TCU then accesses the scan string and captures it in a JTAG-visible register for presentation over the JTAG interface.

VA, HPSTATE, PSTATE, and TL are updated dynamically. Thus, they correctly reflect updates after a trap, a DONE, a RETRY, or a software write to `%tl`.

TPC, TT, and TL_FOR_TT, however, are updated only when the strand takes a trap. They are not updated for DONE, RETRY, or software writes to `%tl`. For example, if the processor traps from TL = 0 to TL = 1 to TL = 2 and then uses DONE and/or RETRY to get back to TL = 0, shadow scan will still reflect TT[2], TPC[2], and TL_FOR_TT will still be 2. Similarly, if the processor traps out to TL = 2 and then software writes TL to 1 or 0, shadow scan will still show TT[2], TPC[2], and TL_FOR_TT will still be 2.

On the JTAG bus, bit 117 appears first, followed by bit 116, sequentially down to and including bit 0.

If multiple traps occur after the state was atomically captured into the shadow scan string but while the shadow scan string is being scanned, only the state belonging to the last trap remains to be (potentially) captured on the next capture point. Due to the length of time to perform a shadow scan relative to the time between traps, sampling via shadow scan can miss several traps.

All eight strand shadow scans are scanned serially as one chain, with strand 0 closest to TDI and strand 7 closest to TDO. Any strand marked unavailable in the CMP STRAND_AVAILABLE register will not be included when scanned via TDI to TDO. The shadow scan chain for a given strand is placed in that strand's second scan chain during ATPG test mode; they are accessible otherwise only via JTAG shadow scan instructions (that is, not during JTAG serial scan).

F.5.2 L2 Shadow Scan

Shadow scan for L2 Error registers is controlled via JTAG. The contents to be captured in the shadow scan are listed in TABLE 29-9 on page 458.

All eight L2 shadow scan contents are captured at the same time and are available at TDO with L2T0 first and L2T7 last (closest to TDO). JTAG instructions to support L2 tag shadow scan are shown in TABLE F-1 on page 545.

On the JTAG bus, bit 141 appears first, followed by bit 140, sequentially down to and including bit 0.

F.6 JTAG Memory BIST

The memory BIST or MBIST engines for OpenSPARC T2 are based on the engine used in OpenSPARC T1. In OpenSPARC T2 there are 80 MBIST engines: 3 per virtual processor (24 total) and 56 distributed throughout the SOC logic. Each MBIST engine will therefore test several arrays. However, these 80 MBIST engines are mapped to 48 JTAG-visible engines. Please refer to TCU Microarchitecture Specification for the mapping.

The MBIST operation may be controlled by the TCU during reset sequencing via the JTAG interface or as invoked via software. The MBIST engines can be operated in a serial mode, a parallel mode, or a diagnostic mode for memory bit-fail mapping. Both the serial and parallel modes run MBIST in a pass/fail mode, where the only information available is whether MBIST passed all of its arrays or failed at least one of them.

For more details on OpenSPARC T2's memory BIST architecture and operation, refer to the TCU Microarchitecture Specification.

F.6.1 MBIST Modes

F.6.1.1 Serial Mode

JTAG will typically be used to run MBIST in the serial mode. When activated in serial mode, the MBIST engines will be started sequentially in the order specified in the TCU Microarchitecture Specification.

To enable the serial MBIST mode via JTAG, the instruction TAP_MBIST_BYPASS must be used to specify which of the 48 JTAG Visible MBIST engines to bypass, if any, via the MBIST_BYPASS register. Next, the TAP_MBIST_MODE is used to clear the parallel mode bit in the MBIST_MODE register. The TAP_MBIST_START instruction is then programmed into JTAG; when JTAG enters the `run-test-idle` state, the MBIST operation will be started; it is not necessary to remain in the `run-test-idle` state. It is up to the user to wait a predetermined number of cycles for the MBIST operation for all arrays to finish. Status can be checked using the TAP_MBIST_RESULT instruction and capturing the MBIST_RESULT register (2 bits) in the capture-DR state and examining them; this can be done repeatedly for polling (via capture-DR, without staying in `run-test-idle`). This allows early truncation of the test (via the TAP_MBIST_ABORT instruction) if the fail bit becomes active before the MBIST operation is done. The done bit must be set to validate a fail bit of 0 indicating a passing condition. A done bit set to 1 and a fail bit set to 0 indicates all arrays for the selected MBIST engines passed MBIST.

The default operation is to run BISI instead of MBIST. To run BISI, the instruction TAP_MBIST_DIAG may be used to program a given MBIST engine's config bits. Selection of BISI or MBIST is done by setting the corresponding bit in the MBIST config register of the MBIST engines via the MBIST_BYPASS register or by setting the bisi bit the TAP_MBIST_MODE register.

F.6.1.2 Parallel Mode

JTAG must be used to run MBIST in the parallel mode. When activated in parallel mode, the MBIST engines will be started in parallel, whereas the arrays controlled by each individual MBIST engine will test their arrays sequentially. Operation of MBIST parallel mode via JTAG is similar to the serial mode, except that the parallel

mode bit of the MBIST_MODE register must instead be set, using the TAP_MBIST_MODE instruction. There is no non-JTAG default method of running MBIST in parallel mode.

Note When the serial or parallel MBIST is determined to be finished (via polling/examination of the done/fail register or a timeout), all that is known is that either all arrays passed or at least one of them failed. To get information on which MBIST engine failed, the TAP_MBIST_GETDONE instruction must be used. This allows capture of all 48 JTAG visible done bits which may then be observed at TDO; the TAP_MBIST_GETFAIL similarly captures all 48 JTAG visible fail bits. If detailed information as to which array failed within a given MBIST engine is needed, then the TAP_MBIST_DIAG instruction must be used to retrieve the contents of the specific MBIST engine that indicated a fail.

F.6.1.3 Diagnostic Mode

In one method to perform bit-fail mapping, the TAP_MBIST_DIAG instruction is used to access the MBIST engine as the target JTAG data register. In this diagnostic mode only one MBIST engine should be selected, by setting the appropriate bits in the MBIST_BYPASS register via the TAP_MBIST_BYPASS instruction; it is up to the user to bypass all but one MBIST engine. Only one array controlled by the selected MBIST engine may be active; this is specified by scanning in (loading) the target MBIST engine registers. After both the MBIST engine and array are specified, the TAP_MBIST_START is programmed, and entering `run-test-idle` will start the MBIST operation on the selected array. After an appropriate wait time, the test should finish. Polling via TAP_MBIST_RESULT can be used to inspect the done/fail JTAG data register, or the TAP_MBIST_GETDONE and TAP_MBIST_GETFAIL can be used to determine the MBIST test results.

To get the detailed information on the target array, the TAP_MBIST_DIAG instruction must be used. This allows the contents of the targeted MBIST engine to be scanned as the MBIST_DIAG register via TDO.

F.6.1.4 Abort Mode

To abort any MBIST activity the TAP_MBIST_ABORT instruction should be used. This will cause all MBIST start signals to be deasserted and any internal JTAG states to be reset. A separate instruction is useful since the JTAG MBIST instructions have memory. Use of TAP_MBIST_ABORT does not clear any of the JTAG data registers used for or during MBIST—only the control states and signals—and does not clear the MBIST engine flops; this allows the TAP_MBIST_DIAG to be used to get data on the failing arrays. Entering `test-logic-reset` state will also stop MBIST.

F.6.2 JTAG MBIST Registers

In OpenSPARC T2, JTAG accessible registers for MBIST are as follows.

TABLE F-16 JTAG MBIST Registers

Register	JTAG Instruction	Fields
RESULT{1:0}	TAP_MBIST_RESULT	Bit 1 – 1 when all 48 Jtag visible mbist engines are done. Bit 0 – 1 if any of 48 Jtag visible mbist engines reports a fail.
BYPASS{47:0}	TAP_MBIST_BYPASS	One bit per JTAG visible mbist engine; to bypass an engine during MBIST testing set its bit to 1.
DONE{47:0}	TAP_MBIST_GETDONE	One bit per JTAG visible mbist engine; a 1 indicates the corresponding engine is done; same order as MBIST_BYPASS register.
FAIL{47:0}	TAP_MBIST_GETFAIL	One bit per JTAG visible mbist engine; a 1 indicates the corresponding engine failed MBIST for one of its arrays.
DIAG{k:0}	TAP_MBIST_DIAG	Includes targeted MBIST engines in a cluster; variable length.
MODE{2:0}	TAP_MBIST_MODE	Bit 2 – user mode if set. Bit 1 – bisi mode if set, else bist mode. Bit 0 – parallel mode if 1, serial mode if 0.
None	TAP_MBIST_CLKSTPEN	Enables mbist controller to begin cycle counter; reset with TLR or TAP_CLOCK_START.

F.6.3 MBIST Clock Stop and Scan Dump

The cycle counter may be used in conjunction with MBIST to stop clocks and perform a scan dump. The instruction TAP_MBIST_CLKSTPEN must be programmed to enable the cycle counter for MBIST. If enabled, the cycle counter will begin decrementing when the MBIST controller begins operation. When the cycle counter reaches zero, a hard clock stop will be issued to the clock sequencer.

All relevant registers—clock domain, clock stop delay—will be recognized in this mode to allow control of the clock stop sequence. The clock stop status may be checked with TAP_CLOCK_STATUS, and when stopped the scan chains can be dumped via TAP_SERSCAN.

Using this feature and repeatedly running MBIST with successively greater cycle count values allows another method of bit-fail mapping arrays. This is sometimes referred to as MBIST Plus. Since the start of MBIST and when the cycle counter begins, decrementing is coordinated and synchronized to the same CMP clock cycle the entire process should be repeatable and cycle accurate

F.6.4 MBIST DMO: Direct Memory Observe

For a complete description of DMO, refer to the OpenSPARC T2 DMO specification. There are three JTAG instructions: TAP_DMO_ACCESS, TAP_DMO_CLEAR, and TAP_DMO_CONFIG, as described in TABLE F-1 on page 545. The TAP_DMO_ACCESS puts the chip in DMO mode, so data from BIST run on L2 Tags or certain SPC arrays are observable at package pins. TAP_DMO_CLEAR clears this mode. To access and program the DMO control logic inside TCU the TAP_DMO_CONFIG instruction should be used to set the 32-bits as desired.

TABLE F-17 JTAG DMO Configuration Register

Register	JTAG Instruction	Bit	Description
DMO_CONFIG {31:0}	TAP_DMO_ CONFIG	31:16	16-bit shift register.
		15	1 selects CMP clock domain, 0 selects I/O clock domain
		14:13	00 selects DMO path to strands 4, 5, 1, or 0. 01 selects DMO path to strands 6, 7, 3, or 2; 10 selects DMO path to L2 tags 4, 5, 1, or 0; 11 selects DMO path to L2 tags 6, 7, 3, or 2.
		12:11	00 selects RTX of NIU; 11 is not allowed.
		10:8	000 – rtx_txc_txe0_dmo_dout 001 – rtx_txc_txe1_dmo_dout 010 – rtx_rxc_ipp0_mb3_dmo_dout 011 – rtx_rxc_ipp1_mb3_dmo_dout 100 – rtx_rxc_zcp0_mb7_dmo_dout 101 – rtx_rxc_zcp1_mb7_dmo_dout 110 – rtx_rxc_vlan_mb6_dmo_dout default – rtx_txc_txe0_dmo_dout 111 – Not allowed.
		7	Selects data cache.
		6	Selects instruction cache.

TABLE F-17 JTAG DMO Configuration Register *(Continued)*

Register	JTAG Instruction	Bit	Description
		14:13	14:135:32:0
		5:3	00 xx0xxx → CORE4
		2:0	10xx0xxx → L2T4
			00x01xxx → CORE5
			10x01xxx → L2T5
			00011xxx → CORE1
			10011xxx → L2T1
			00111xxx → CORE0
			10111xxx → L2T0
			01xxxxx0 → CORE6
			11xxxxx0 → L2T6
			01xxxx01 → CORE7
			11xxxx01 → L2T7
			01xxx011 → CORE3
			11xxx011 → L2T3
			01xxx111 → CORE2
			11xxx111 → L2T2

F.7 JTAG Logic BIST

The Logic BIST test function is only applied to the SPC cores in OpenSPARC T2, and one engine is instantiated per core. The control of the Logic BIST engines comes from the TCU via either software or JTAG.

The control logic allows the Logic BIST engines to be run in parallel or in series and gathers the done signals for JTAG to query. There is no pass/fail indication that comes from the Logic BIST engines, so the engine must be scanned to determine the result.

F.7.1 JTAG Logic BIST Registers

In OpenSPARC T2, JTAG accessible registers for Logic BIST are listed in TABLE F-18.

TABLE F-18 JTAG Logic BIST Registers

Register	Jtag Instruction	Fields
BYPASS{7:0}	TAP_LBIST_BYPASS	One bit per Logic BIST engine; to bypass an engine during testing, set its bit to 1.
MODE{1:0}	TAP_LBIST_MODE	Bit 1 – Program access mode selected. Bit 0 – Parallel mode if 1, serial mode if 0.
LBIST{k:0}	TAP_LBIST_ACCESS	Includes targeted Logic BIST engines across strands.
DONE{7:0}	TAP_LBIST_GETDONE	One bit per MBIST engine; a 1 indicates the corresponding engine is done; same order as Logic BIST bypass register.

F.7.2 Accessing Pass/Fail Signature

To determine if the Logic BIST engine passed or failed, the signature must be scanned out via JTAG using TAP_LBIST_ACCESS. The signature must be compared to a known-good value.

APPENDIX **G**

(Placeholder chapter)

APPENDIX **H**

Glossary

This chapter defines concepts and terminology unique to the OpenSPARC T2 implementation. Definitions of terms common to all UltraSPARC Architecture implementations may be found in the Definitions chapter of *UltraSPARC Architecture 2007*.

ALU Arithmetic Logical Unit

architectural state Software-visible registers and memory (including caches).

ARF Architectural register file.

ASI ring A daisy-chained bus connected in a loop fashion that goes through all of the blocks that have structures with diagnostic path or control registers for ASI access.

blocking ASI An ASI access that accesses its ASI register or array location once all older instructions in that strand have retired, no instructions in the other strand can issue, and the store queue, TSW, and LMB are all empty.

Additionally, the snoop pipeline is stalled before the ASI register/array location is accessed.

branch outcome A reference as to whether or not a branch instruction will alter the flow of execution from the sequential path. A taken branch outcome results in execution proceeding with the instruction at the branch target; a not-taken branch outcome results in execution proceeding with the instruction along the sequential path after the branch.

branch resolution A branch is said to be resolved when the result (that is, the branch outcome and branch target address) has been computed and is known for certain. Branch resolution can take place late in the pipeline.

branch target address The address of the instruction to be executed if the branch is taken.

CAM Content addressable memory.

commit An instruction commits when it modifies architectural state.

complex instruction A complex instruction is an instruction that requires the creation of secondary "helper" instructions for normal operation, excluding trap conditions such as spill/fill traps (which use helpers). Refer to*Instruction Latency* on page 476 for a complete list of all complex instructions and their helper sequences.

consistency *See* **coherence**.

CPU Central Processing Unit. A synonym for **virtual processor**.

CSR Control Status register.

DFT Designed for test.

DTLB Data Cache Translation lookaside buffer.

ECC Error correction code.

EXU Execution Unit.

FP Floating point.

helper A helper instruction is generated by the IRU in response to a complex instruction. Helper instructions are not visible to software. Refer to *Instruction Latency* on page 476 [xref OK?] for a complete list of all complex instructions and their helper sequences.

IFU Instruction Fetch Unit.

ITLB Instruction Cache Translation lookaside buffer.

L2C (or L2$) Level 2 cache.

leaf procedure A procedure that is a leaf in the program's call graph; that is, one that does not call (by using CALL or JMPL) any other procedures.

nonblocking ASI A nonblocking ASI access will access its ASI register/array location once all older instructions in that strand have retired, and there are no instructions in the other strand which can issue.

older instruction Refers to the relative fetch order of instructions. Instruction *i* is older than instruction *j* if instruction *i* was fetched before instruction *j*. Data dependencies flow from older instructions to younger instructions, and an instruction can only be dependent upon older instructions.

one hot An *n*-bit binary signal is one hot if and only if $n - 1$ of the bits are each zero and a single bit is a 1.

Page Table Entry (PTE) Describes the virtual-to-physical translation and page attributes for a specific page. A PTE generally means an entry in the page table or in the TLB, but it is sometimes used as an entry in the TSB (translation storage buffer). In general, a PTE contains fewer fields than does a TTE. *See also* TLB and TSB.

PIO Programmed I/O.

PPN Physical Page Number

pr Processor reset.

PTE Page Table Entry.

quadlet

SFAR Synchronous Fault Address register.

SFSR Synchronous Fault Status register.

SIAM Set interval arithmetic mode instruction.

strand identifier (SID) In a processor implementing 2^n strands, the strand identifier is an n-bit value used to uniquely identify each strand. The strand identifier in OpenSPARC T2 is six bits wide.

VPA Virtual page array.

VPN Virtual page number.

younger instruction *See* **older instruction**.

writeback The process of writing a dirty cache line back to memory before it is refilled.

APPENDIX I

Bibliography

[contents of this appendix are TBD]

Index

SYMBOLS

A

B

C

D

E

F

J

L

M

N

O

P

Q

R

S

T

U

V

W

X