SPARC® JPS2 Extensions: Sun UltraSPARC IV

Sun Microsystems, Inc.

Revision 1.0

March, 2004

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 1-800-555-9SUN

Part No. 806-____-0.2
Revision 1.0 March 2004

Please
Recycle

Adobe PostScript

Contents

S.CHAPTER **1**

Overview

UltraSPARC IV is a high performance microprocessor that compies with the *SPARC Joint Programming Specification 2 (JPS2) Common Specification* and the *SPARC Architecture Manual-Version 9*. This document describes the architecture extension of the UltraSPARC IV to the JPS2 common specification.

S.CHAPTER **2**

Definitions

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.

S.CHAPTER **3**

Architectural Overview

Please refer to Chapter 3 of the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.

S.CHAPTER **4**

Data Formats

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Implementation Supplement*.

S.CHAPTER **5**

Registers

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* for mor details.

5.1 Data Cache Unit Control Register (DCUCR)

FIGURE 5-1 illustrates the DCUCR implemented in UltraSPARC IV. TABLE 5-1 describes those bits that are defined as implementation dependent or as reserved bits in the JPS2 Common Specification.

Name: `ASI_DCU_CONTROL_REGISTER`
ASI 45_{16}, VA<63:0>==00_{16},
Read-Write

—	cp	cv	me	re	pe	hpe	spe	sl	we	pm	vm	pr	pw	vr	vw	—	wih	dm	im	dc	ic
63 50	49	48	47	46	45	44	43	42	41	40 33	32 25	24	23	22	21	20 5	4	3	2	1	0

FIGURE 5-1 Data Cache Unit Control Register

TABLE 5-1 DCU Control Register Description

Bits	Field	Use - Description
4	wih	Write Cache index function control
41	we	Write Cache enable
42	sl	Second load steering enable
43	spe	Software prefetch enable
44	hpe	Prefetch Cache HW prefetch enable
45	pe	Prefetch Cache enable
46	re	RAW bypass enable
47	me	Noncacheable store merging enable

S.CHAPTER **6**

Instructions

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.CHAPTER **7**

Traps

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* for more details.

UltraSPARC IV implements two implementation dependent traps: TT= 071_{16} and TT = 072_{16}.TABLE 7-1 lists their definitions, global register set used, and the priority.

TABLE 7-1 UltraSPARCIV implementation-dependent traps.

Exception or Interrupt Requests	TT	Global Register Set	Priority
dcache_parity_error	071_{16}	AG	2
icache_parity_error	072_{16}	AG	2

S.CHAPTER **8**

Memory Models

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.CHAPTER **9**

Multithreaded Processing (MTP)

The UltraSPARC IV processor complies with the MTP specification described in Chapter 9 of the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*. It does not implment ASI_INTR_DISPATCH_EXT_W (ASI 0x77, VA<13:0>=0x78), and it uses one reserved ASI range, ASI 0x63, VA = 0x40, for interconnect support. Please refer to the JPS2 Common Specification, and the Implemenation Dependent and ASI chapters of this document for more details.

S.APPENDIX **A**

Instruction Definitions

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

9.1 Prefetch Data

UltraSPARC IV implements 11 fcn codes: 0, 1, 2, 3, 4, 16, 17, 20, 21, 22, and 23. Among them, 16 and 17 are implemenation dependent. TABLE 9-1 describes the prefetch function for fcn=16 and fcn=17.

TABLE 9-1 Implementation Dependent Prefetch Function Description

fcn	Description
16 (10_{16})	Prefetch Cache line invalidation
17 (11_{16})	Prefetch for read to nearest unified cache

S.APPENDIX **B**

IEEE Std 754-1985 Requirements for SPARC V9

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **C**

Implementation Dependencies

This appendix summarizes implementation dependencies in the SPARC V9 standard. In SPARC V9, the notation "**IMPL. DEP. #*nn*:**" identifies the definition of an implementation dependency; the notation "(impl. dep. #*nn*)" identifies a reference to an implementation dependency. These dependencies are described by their number *nn* in TABLE C-1 on page 26.

The appendix contains these sections:

Note – SPARC International maintains a document, *Implementation Characteristics of Current SPARC-V9-based Products, Revision 9.x*, that describes the implementation-dependent design features of all SPARC V9-compliant implementations. Contact SPARC International for this document at:
home page: `www.sparc.org`
email: info@sparc.org 535 Middlefield Rd, Suite 210
Menlo Park, CA 94025
(415) 321-8692

C.1 Definition of an Implementation Dependency

The SPARC V9 architecture is a *model* that specifies unambiguously the behavior observed by *software* on SPARC V9 systems. Therefore, it does not necessarily describe the operation of the *hardware* of any actual implementation.

An implementation is *not* required to execute every instruction in hardware. An attempt to execute a SPARC V9 instruction that is not implemented in hardware generates a trap. Whether an instruction is implemented directly by hardware, simulated by software, or emulated by firmware is implementation dependent.

The two levels of SPARC V9 compliance are described in *SPARC V9 Compliance* in *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.

Some elements of the architecture are defined to be implementation dependent. These elements include certain registers and operations that may vary from implementation to implementation; they are explicitly identified as such in this appendix.

Implementation elements (such as instructions or registers) that appear in an implementation but are not defined in this document (or its updates) are not considered to be SPARC V9 elements of that implementation.

C.2 Hardware Characteristics

Hardware characteristics that do not affect the behavior observed by software on SPARC V9 systems are not considered architectural implementation dependencies. A hardware characteristic may be relevant to the user system design (for example, the speed of execution of an instruction) or may be transparent to the user (for example, the method used for achieving cache consistency). The SPARC International document, *Implementation Characteristics of Current SPARC-V9-based Products, Revision 9.x*, provides a useful list of these hardware characteristics, along with the list of implementation-dependent design features of SPARC V9-compliant implementations.

In general, hardware characteristics deal with

- Instruction execution speed
- Whether instructions are implemented in hardware

- The nature and degree of concurrency of the various hardware units constituting a SPARC V9 implementation

C.3 Implementation Dependency Categories

Many of the implementation dependencies can be grouped into four categories, abbreviated by their first letters throughout this appendix:

- **Value (v)**
  The semantics of an architectural feature are well defined, except that a value associated with the feature may differ across implementations. A typical example is the number of implemented register windows (impl. dep. #2).
- **Assigned Value (a)**
  The semantics of an architectural feature are well defined, except that a value associated with the feature may differ across implementations and the actual value is assigned by SPARC International. Typical examples are the `impl` field of Version register (VER) (impl. dep. #13) and the `FSR.ver` field (impl. dep. #19).
- **Functional Choice (f)**
  The SPARC V9 architecture allows implementors to choose among several possible semantics related to an architectural function. A typical example is the treatment of a catastrophic error exception, which may cause either a deferred or a disrupting trap (impl. dep. #31).
- **Total Unit (t)**
  The existence of the architectural unit or function is recognized, but details are left to each implementation. Examples include the handling of I/O registers (impl. dep. #7) and some alternate address spaces (impl. dep. #29).

C.4 List of Implementation Dependencies

TABLE C-1 provides a complete list of the UltraSPARC IV implementation of the SPARC V9 implementation dependencies. The Page column lists the page for the context in which the dependency is defined.

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(1 of 6)*

Nbr	Category	Description
1	f	**Software emulation of instructions** The UltraSPARC IV processor meets Level 2 SPARC V9 compliance as specified in *The SPARC Architecture Manual-Version 9*. Refer Section 6.3.10 of the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.
2	v	**Number of IU registers** The UltraSPARC IV processor implements 160 64-bit general purpose registers: 4 sets of global registers (regular, alternate, interrupt, and MMU), 8 each, plus 8 (NWINDOWS = 8) register windows.
3	f	**Incorrect IEEE Std 754-1985 results** The UltraSPARC IV processor implements precise floating-point exception handling. All quad-precision floating-point instructions cause an *fp_exception_other* (with `FSR.ftt` = 3, *unimplemented_FPop*) trap. These operations are emulated in system software.
4-5		.
6	f	**I/O registers privileged status**
7	t	**I/O register definitions**
8	t	**RDASR/WRASR target registers** Software can use read / write ancillary state register instructions to read / write implementation-dependent processor registers (ASRs 18–31).
9	f	**RDASR/WRASR privileged status** The UltraSPARC IV processor implements RDASR / WRASR privileged status as described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* (Refer section A.51)
10-12		*Reserved.*
13	a	**`VER.impl`** The `impl` field of the Version Register is a 16-bit implementation code, 0018_{16}, that uniquely identifies the UltraSPARC IV processor class CPU.
14-15		*Reserved.*
16		*Reserved.*
17		*Reserved.*

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(2 of 6)*

Nbr	Category	Description
18	f	**Nonstandard IEEE 754-1985 results** When bit 22 (NS) of the FSR is set, the UltraSPARC IV processor can deliver a non-IEEE-754-compatible result. In particular, subnormal operands and results can be flushed to 0.
19	a	**FPU version, FSR.ver** The version bit of the FSR identifies a particular implementation of the UltraSPARC IV Floating Point and Graphics Unit architecture.
20-21		*Reserved.*
22	f	**FPU tem, cexc, and aexc** tem is a 5-bit trap enable mask for the IEEE-754 floating-point exceptions. If a floating-point operate instruction produces one or more exceptions, then the corresponding cexc/aexc bits are set and an *fp_exception_ieee_754* (with FSR.ftt = 1, *IEEE_754_exception*) exception is generated.
23		*Reserved.*
24		*Reserved.*
25	f	*Reserved.*
26-28		*Reserved.*
29	t	**Address space identifier (ASI) definitions** See Appendix L for the full list of ASIs implemented in the UltraSPARC IV processor.
30	f	**ASI address decoding** The UltraSPARC IV processor decodes all eight bits of ASIs. See the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification.*
31	f	**Catastrophic error exceptions** The UltraSPARC IV processor does not implement the *internal_processor_error* exception.
32	t	**Deferred traps** The UltraSPARC IV processor supports precise trap handling for all operations except for deferred or disrupting traps from hardware failures encountered during memory accesses.
33	f	**Trap precision** The UltraSPARC IV processor supports precise trap handling for all operations except for deferred or disrupting traps from hardware failures encountered during memory accesses.
34	f	**Interrupt clearing**
35	t	**Implementation-dependent traps** The UltraSPARC IV processor implements two implementation-specific traps.: TT=0x71 and TT=0x72, described in Chapter & on page 13.

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(3 of 6)*

Nbr	Category	Description
36	f	**Trap priorities** The priorities of particular traps are relative and are implementation dependent because a future version of the architecture may define new traps, and implementations may define implementation-dependent traps that establish new relative priorities. The UltraSPARC IV processor implements the priority 2 *fast_ECC_error* exception.
37	f	**Reset trap** On a reset trap, TPC may be inaccurate. The value saved in TPC may differ from the correct trap PC by 63 bytes; this value is not necessarily equal to the PC of any instruction that has been or is scheduled to be executed. The value saved in TnPC will always be 4 more than the value saved in TPC.
38	f	**Effect of reset trap on implementation-dependent registers** The effect of reset traps on all UltraSPARC IV registers is enumerated in TABLE N-1 on page 71.
39	f	**Entering `error_state` on implementation-dependent errors** Upon `error_state` entry, the processor automatically recovers through watchdog reset (WDR) into RED_state.
40	f	**`Error_state` processor state** CWP updates for window traps that enter `error_state` are the same as when `error_state` is not entered.
41		*Reserved.*
42	t, f, v	**FLUSH instruction** Whether FLUSH traps is implementation dependent. On the UltraSPARC IV processor, the FLUSH effective address is ignored. FLUSH does not access the data MMU and cannot generate a data MMU miss or exception.
43		*Reserved.*
44	f	**Data access FPU trap** If a load floating-point instruction causes a trap due to any type of data access error, the contents of the destination register are undefined.
45 - 46		*Reserved.*
47	t	**RDASR** UltraSPARC IV implements RDASR instruction as described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*
48	t	**WRASR** UltraSPARC IV implements WRASR instruction as described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*
49-54		*Reserved.*
55	f	**Floating-point underflow (tininess) detection** UltraSPARC IV implements tininess detection as described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*, i.e., before rounding
56-100		*Reserved.*

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(4 of 6)*

Nbr	Category	Description
101	v	**Maximum trap level** The UltraSPARC IV processor supports five trap levels; that is, MAXTL = 5.
102	f	**Clean windows trap** The UltraSPARC IV processor generates a *clean_window* trap when a SAVE instruction requests a window and there are no more clean windows. System software must then initialize all registers in the next available window(s) to 0 before returning to the requesting context.
103	f	**Prefetch instructions** PREFETCH(A) instructions with fcn = 0–4 are implemented. In accordance with SPARC V9, PREFETCH(A) instructions with fcn = 5–15 cause an *illegal_instruction* trap.
104	a	**VER.manuf** The manuf bit of the Version Register is a 16-bit manufacturer code, $003E_{16}$ (Sun's JEDEC number), that identifies the manufacture of the processor.
105	f	**TICK register** The UltraSPARC IV processor implements a 63-bit TICK counter.
106	f	**IMPDEP2A instructions** The UltraSPARC IV processor extends the standard SPARC V9 instruction set. Unimplemented IMPDEP1 and IMPDEP2 opcodes encountered during execution cause an *illegal_instruction* trap.
107	f	**Unimplemented LDD trap** LDD instructions are directly executed in hardware, so the *unimplemented_LDD* exception does not exist in the UltraSPARC IV processor.
108	f	**Unimplemented STD trap** STD instructions are directly executed in hardware, so the *unimplemented_STD* exception does not exist in the UltraSPARC IV processor.
109	f	***LDDF_mem_address_not_aligned*** LDDF(A) causes an *LDDF_mem_address_not_aligned* trap if the effective address is 32-bit aligned but not 64-bit (doubleword) aligned.
110	f	***STDF_mem_address_not_aligned*** STQF(A) causes an *STDF_mem_address_not_aligned* trap if the effective address is 32-bit aligned but not 64-bit (doubleword) aligned.
111	f	***LDQF_mem_address_not_aligned*** This trap is not used in SPARC JPS2. See the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.
112	f	***STQF_mem_address_not_aligned*** This trap is not used in SPARC JPS2. See the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.
113	f	**Implemented memory models** The UltraSPARC IV processor supports only the TSO memory model.

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(5 of 6)*

Nbr	Category	Description
114	f	**RED_state trap vector address (RSTVaddr)** The RED_state trap vector is located at an implementation-dependent address referred to as RSTVaddr. In the UltraSPARC IV processor, the RED_state trap vector address (RSTVaddr) is 256 MB below the top of the virtual address space. Virtual address FFFF FFFF F000 0000_{16} is passed through to physical address 7FF F000 0000_{16} in RED_state.
115	f	**RED_state processor state** (What occurs after the processor enters RED_state is implementation dependent.) In the UltraSPARC IV processor, a reset or trap that sets PSTATE.red (including a trap in RED_state) will clear the DCU Control Register, including enable bits for I-cache, D-cache, IMMU, DMMU, and virtual and physical watchpoints.
116	f	**SIR_enable control flag** As for all SPARC V9 JPS2 processors, in the UltraSPARC IV processor, SIR is permanently enabled. A software-initiated reset (SIR) is initiated by execution of a SIR instruction while in privileged mode. In nonprivileged mode, a SIR instruction behaves as a NOP.
117	f	**MMU disabled prefetch behavior** When the data MMU is disabled, accesses default to the settings in the Data Cache Unit Control Register CP and CV bits. **Note:** E is the inverse of CP.
118	f	**Identifying I/O locations** (The manner in which I/O locations are identified is implementation dependent.)
119	f	**Unimplemented values for PSTATE.mm** Because the UltraSPARC IV processor implements only the TSO memory model, PSTATE.mm always reads as 00_2 and writes to it are ignored.
120	f	**Coherence and atomicity of memory operations** Cacheable accesses observe supported cache coherency protocols; the unit of coherence is 64 bytes. Noncacheable and side-effect accesses do not observe supported cache coherency protocols; the smaller unit in each transaction is a single byte.
121	f	**Implementation-dependent memory model** The UltraSPARC IV processor only implements the TSO memory model.
122	f	**FLUSH latency** When a FLUSH operation is performed, the UltraSPARC IV processor guarantees that earlier code modifications will be visible across the whole system.
123	f	**Input/output (I/O) semantics** The UltraSPARC IV MMU includes an attribute bit e in each page translation, which when set signifies that this page has side effects.
124	v	**Implicit ASI when TL > 0** (See the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.)

TABLE C-1 UltraSPARC IV Implementation of the SPARC V9 Implementation Dependencies *(6 of 6)*

Nbr	Category	Description
125	f	**Address masking** When PSTATE.am = 1, the UltraSPARC IV processor writes the full 64-bit program counter value to the destination register of a CALL, JMPL, or RDPC instruction. When PSTATE.am = 1 and a trap occurs, the UltraSPARC IV processor writes the full 64-bit program counter value to TPC[TL].
126		*Reserved.*
127-199	—	*Reserved.*

TABLE C-2 provides a list of implementation dependencies that, in addition to those in TABLE C-1, apply to SPARC JPS2 processors.

TABLE C-2 SPARC JPS2 Implementation Dependencies *(1 of 8)*

Nbr	Description
200-201	*Reserved.*
202	***fast_ECC_error* trap** The UltraSPARC IV processor implements *fast_ECC_error* trap. It indicates that an ECC error was detected in an external cache and its trap type is 070_{16}.
203	**Dispatch Control Register bits 13:6 and 1** In the UltraSPARC IV processor, bits 11:6 of the Dispatch Control Register (DCR) can be programmed to select the set of signals to be observed at obsdata<9:0>. DCR bit 1 is the Interrupt Floating-Point Operation Enable (ifpoe) bit.
204	**DCR bits 5:3 and 0** UltraSPARC IV processor implements all of these bits. Their definitions are described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*
205	**Instruction Trap Register** The UltraSPARC IV processor implements the Instruction Trap Register.
206	**SHUTDOWN instruction** In privileged mode, the SHUTDOWN instruction executes as a NOP in the UltraSPARC IV processor.
207	**PCR register bits 47:32, 26:17, and 3** Bits 47:32, 26:17, and bit 3 of PCR are unused in the UltraSPARC IV processor. They read as zero and should be written only to zero or to a value previously read from those bits.
208	**Ordering of errors captured in instruction execution** The order in which errors are captured in instruction execution is in order of detection.

TABLE C-2 SPARC JPS2 Implementation Dependencies *(2 of 8)*

Nbr	Description
209	**Software intervention after instruction-induced error** In UltraSPARC IV processor, precision of the trap to signal an instruction-induced error of which recovery requires software intervention can be either precise trap or deferred trap.
210	**ERROR output signal** In UltraSPARC IV processor, the causes of ERROR output signal are as follows: • Bus transaction time out • Internal unrecoverable error • Incoming system address parity error • Uncorrectable system bus MTag ECC error • Uncorrectable E-Cache Tag ECC error When the ERROR signal is asserted, the processor expects a system reset. Hence, its behavior after the ERROR is asserted is undefined.
211	**Error logging registers' information** (The information that the error logging registers preserves beyond the reset induced by an ERROR signal is implementation dependent.)
212	**Trap with fatal error** In UltraSPARC IV processor, the following two situations will generate a trap along with ERROR signal assertion: • Uncorrectable system bus MTag ECC error • Uncorrectable E-Cache Tag ECC error
213	**`AFSR.priv`** AFSR.priv accumulates the state of the `PSTATE.priv` bit at the time the event is detected, rather than the `PSTATE.priv` value associated with the instruction that caused the access which returns the error.
214	**Enable/disable control for deferred traps** The UltraSPARC IV processor implements an enable/disable control feature for deferred traps.
215	**Error barrier** On the UltraSPARC IV processor, `DONE` and `RETRY` instructions implicitly provide an error barrier function as `MEMBAR #Sync`.
216	***data_access_error* trap precision** On the UltraSPARC IV processor, *data_access_error* causes a deferred trap.
217	***instruction_access_error* trap precision** On the UltraSPARC IV processor, *instruction_access_error* causes a deferred trap.
218	***async_data_error*** The UltraSPARC IV processor does not implement this trap.
219	**Asynchronous Fault Address Register (AFAR) allocation** AFAR1 VA=0_{16}, and AFAR2 VA=8_{16}.

TABLE C-2 SPARC JPS2 Implementation Dependencies *(3 of 8)*

Nbr	Description
220	**Addition of logging and control registers for error handling** The UltraSPARC IV processor implements additional logging/control registers for error handling.)
221	**Special/signalling ECCs** The UltraSPARC IV processor generates "special" or "signalling" ECCs. The method does not include the processor-ID.
222	**TLB organization** The UltraSPARC IV implements separate instruction and data memory management units (I-MMU and D-MMU). The I-MMU contains one 128-entry 2-way associative TLB and one 16-entry fully associative TLB. The D-MMU contains two 512-entry 2-way associative TLBs and one 16-entry fully associative TLB.
223	**TLB multiple-hit detection** The UltraSPARC IV does not detect TLB multiple-hit. Hence, this scenario may cause undefined results.
224	**MMU physical address width** The physical address width supported by the UltraSPARC IV processor is 43 bits.
225	**TLB locking of entries** **(**The mechanism by which entries in TLB are locked is implementation dependent in JPS2.)
226	**TTE support for `cv` bit** The `cv` bit is fully implemented in the UltraSPARC IV processor.
227	**TSB number of entries** In the UltraSPARC IV processor, the maximum number of TSB entries is 512 x 2^7, or 64K entries.
228	**`tsb_hash` supplied from `TSB` or context-ID register** In the UltraSPARC IV processor, `tsb_hash` is supplied from a `TSB` extension register.
229	**`TSB_Base` address generation** The UltraSPARC IV processor generates the `TSB_Base` address by exclusive-ORing the `TSB_Base` register and a TSB register.
230	***data_access_exception* trap** The UltraSPARC IV processor implements mandatory causes of *data_access_exception* trap specified in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.
231	**MMU physical address variability** The physical address width supported by the UltraSPARC IV processor is fixed to 43 bits.
232	**DCU Control Register bits** The UltraSPARC IV processor fully implements the `cp` and `cv` "cacheability" bits in the DCU Control Register.

TABLE C-2 SPARC JPS2 Implementation Dependencies *(4 of 8)*

Nbr	Description
233	**`tsb_hash` field** The UltraSPARC IV processor implements `tsb_hash` field in the I/D Primary/Secondary/Nucleus TSB Extension Register.
234	**TLB replacement algorithm** The UltraSPARC IV processor implements random replacement strategy.
235	**TLB data access address assignment** The TLB data access address assignment in the UltraSPARC IV processor is as follows: • bits 2-0: always 0 • bits 11-3: TLB entry • bits 15-12: reserved • bits 17-16: TLB number (select which TLB to access) • bits 18: always 0 • bits 63-19: reserved
236	**`tsb_size` field width** In the UltraSPARC IV processor, `tsb_size` is 3 bits wide, occupying bits 2:0 of the `TSB` register. The maximum number of `TSB` entries is, therefore, 512×2^7 (64K entries).
237	**`JMPL`/`RETURN` *mem_address_not_aligned*** On a *mem_address_not_aligned* trap that occurs during a `JMPL` or `RETURN` instruction, the UltraSPARC IV processor updates the `D-SFAR` and `D-SFSR` registers with the fault address and status, respectively.
238	**TLB page offset for large page sizes** The UltraSPARC IV stores page offset bits for larger page sizes (`pa`<15:13>, `pa`<18:13>, and `pa`<21:13> for 64 KB, 512 KB, and 4 MB pages, respectively) in the TLB. The data returned from those fields by a Data Access read are the data previously written to them.
239	**Register access by ASIs 55_{16} and $5D_{16}$ at virtual address 40000_{16} - $60FF8_{16}$** On the UltraSPARC IV processor, loads and stores to ASIs 55_{16} and $5D_{16}$ at virtual address 40000_{16} - $60FF8_{16}$ access the IMMU and DMMU (respectively) TLB CAM Diagnostic Registers.
240	**DCU Control Register bits 47:41** The UltraSPARC IV processor implements all seven bits, for store queue and prefetch control.
241	**Address Masking and `DSFAR`** When `PSTATE.am` = 1 and an exception occurs, the UltraSPARC IV processor writes the full 64-bit address to the Data Synchronous Fault Address Register (`DSFAR`).

TABLE C-2 SPARC JPS2 Implementation Dependencies *(5 of 8)*

Nbr	Description
242	**TLB lock bit** In the UltraSPARC IV processor, the TLB lock bit is only implemented in the D-MMU 16-entry, fully associative TLB, and the I-MMU 16-entry, fully associative TLB. In other TLBs, D-MMU 512-entry, 2-way associative TLBs and I-MMU 128-entry, 2-way associative TLB, reading the lock bit returns 0 and writing to it is ignored.
243	**Interrupt Vector Dispatch Status Register `busy`/`nack` pairs** The UltraSPARC IV processor implements 32 `busy`/`nack` pairs in the Interrupt Vector Dispatch Status Register.
244	**Data Watchpoint** On the UltraSPARC IV processor, watchpoint comparison is only done on the MS (memory) pipeline of the processor; any second-issued Ax pipe FP loads will not trigger a watchpoint. For reliable use of the watchpoint mechanism, the second FP load feature must be disabled (`DCUCR.sl` set to 0).
245	**Call/Branch Displacement Encoding in I-Cache** In the UltraSPARC IV processor, the least significant 11 bits (bits 10:0) of a `CALL` or branch (`BPcc`, `FBPcc`, `Bicc`, `BPr`) instruction in the instruction cache contain the sum of the least significant 11 bits of the architectural instruction encoding (as appears in main memory) and the least significant 11 bits of the virtual address of the `CALL`/branch instructions.
246	**`va`<38:29> for Interrupt Vector Dispatch Register Access** The UltraSPARC IV processor interprets all 10 bits of `va`<38:39> when the Interrupt Vector Dispatch Register is written.
247	**Interrupt Vector Receive Register `sid` Fields** The UltraSPARC IV processor sets all 10 physical module ID (`mid`) bits in the `sid_u` and `sid_l` fields of the Interrupt Vector Receive Register. The UltraSPARC IV processor obtains `sid_u` from `va`<38:34> of the interrupt source and `sid_l` from `va`<33:29> of the interrupt source.
248	**Conditions for *fp_exception_other* with *unfinished_FPop*** The UltraSPARC IV processor triggers *fp_exception_other* with trap type *unfinished_FPop* under the conditions described in *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*. These conditions differ from the JPS2 "standard" set, described in the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification*.
249	**Data Watchpoint for Partial Store Instruction** Watchpoint exceptions on Partial Store instructions occur conservatively on the UltraSPARC IV processor. The `DCUCR` Data Watchpoint masks are only checked for nonzero value (watchpoint enabled). The byte store mask (`r[rs2]`) in the Partial Store instruction is ignored, and a watchpoint exception can occur even if the mask is zero (that is, no store will take place).

TABLE C-2 SPARC JPS2 Implementation Dependencies *(6 of 8)*

Nbr	Description
250	**PCR accessibility when PSTATE.priv = 0** In the UltraSPARC IV processor, access to PCR is strictly privileged; when PSTATE.priv = 0, an attempt to execute either an RDPCR or a WRPCR instruction causes a *privileged_opcode* exception.
251	*Reserved.*
252	**DCUCR.dc (Data Cache Enable)** The UltraSPARC IV processor implements DCUCR.dc. When DCUCR.dc = 0 (D-cache is disabled), the data cache is not updated. When the D-cache is reenabled, software must flush any inconsistent lines from the D-cache.
253	**DCUCR.ic (Instruction Cache Enable)** The UltraSPARC IV processor implements DCUCR.ic. When DCUCR.ic = 0 (I-cache is disabled), the instruction cache is not updated. When the I-cache is reenabled, software must invalidate any inconsistent lines from the instruction cache.
254	**Means of exiting error_state** Upon entry into error_state, the UltraSPARC IV processor generates an immediate *watchdog_reset* (WDR).
255	**LDDFA with ASI $E0_{16}$ or $E1_{16}$ and misaligned destination register number** For LDDF with ASI $E0_{16}$ or $E1_{16}$, if a misaligned (not multiple of 8) destination register number is specified, the UltraSPARC IV processor generates an *illegal_instruction* exception.
256	**LDDFA with ASI $E0_{16}$ or $E1_{16}$ and misaligned memory address** For LDDF with ASI $E0_{16}$ or $E1_{16}$, if a misaligned (not 64-byte aligned) memory address is specified, the UltraSPARC IV processor generates a *mem_address_not_aligned* exception.
257	**LDDFA with ASI $C0_{16}$–$C5_{16}$ or $C8_{16}$–CD_{16} and misaligned memory address** For LDDF with $C0_{16}$–$C5_{16}$ or $C8_{16}$–CD_{16}, if a misaligned (not 8-byte aligned) memory address is specified, the UltraSPARC IV processor generates a *data_access_expection* exception, with fault type 08_{16} recorded in DSFSR.ftype.
258	**ASI_SERIAL_ID** The UltraSPARC IV processor does not implement the ASI_SERIAL_ID register.
259-299	*Reserved.*
300	**Attempted access to ASI registers with LDDA** Upon attempted access to ASI registers with LDDA, the UltraSPARC IV processor will generate a *data_access_exception.*
301	**Attempted access to ASI registers with STDA** Upon attempted access to ASI registers with STDA, the UltraSPARC IV processor will generate a *data_access_exception.*

TABLE C-2 SPARC JPS2 Implementation Dependencies *(7 of 8)*

Nbr	Description
302	**Scratchpad Registers** The UltraSPARC IV processor does not implement the scratchpad registers.
303	**GSR.gcc bits** The UltraSPARC IV processor does not implement the gcc bits. The gcc bits are reserved.
304	**XIR** The XIR on UltraSPARC IV processor affects only one virtual processor.
305	**Updating AFSR/AFAR** The UltraSPARC IV processor implements two pairs of AFSR/AFAR registers: AFSR1/AFAR1 and AFSR2/AFAR2. The secondary pair is provided to capture the first event of a series of errors. The AFSR1 bits are either read-only or read-write-1-clear (RW1C), and the AFSR2 are read-only. Writes to read-only fields have no effect. Writes to AFSR1 RW1C bits with particular bits set will clear the corresponding bits in both AFSR1 and AFSR2. Writes to AFSR1 RW1C bits with particular bits clear have no effect to those bits in either AFSR. The AFAR1 address field is read-write, however, the AFAR2 is read-only. A write the AFAR1 address field will also update the AFAR2's address field.
306	**Trap Type Generated Upon Attempted Access to Noncacheable Page with LDDA** The UltraSPARC IV processor generates TT = 30 when LDDA is issued to an address which is not mapped to a cacheable memory space.
307	**Global Register Set Enabled During Normal Trap @ TL = MAXTL – 1** In an UltraSPARC IV processor, if normal trap while TL = MAXTL – 1, the trap globals are selected the same as for any other normal trap: if the trap was caused by a *fast_instruction_access_MMU_miss, fast_data_access_MMU_miss*, *fast_data_access_protection*, *instruction_access_exception,* or *data_access_exception* exception, the MMU globals are enabled; if the trap was caused by an *interrupt_vector_trap* exception, the interrupt globals are enabled; otherwise, the standard alternate globals are enabled.
1100	**Core Interrupt ID Register** In the UltraSPARC IV processor, bits[9:0] of the Core Interrupt ID register contain the ID of the core issuing the interrupt. Bits[15:10] of the Int ID field always read as 0.
1101	**Interrupt Vector Dispatch Extension Register** The UltraSPARC IV processor does not implements the Interrupt Vector Dispatch Extension register.
1102	**Power used by CMP** The UltraSPARC IV processor does not guarantee that disabling a virtual processor will reduce power used by a CMP.

TABLE C-2 SPARC JPS2 Implementation Dependencies *(8 of 8)*

Nbr	Description
1103	**Updating Core Enable Register** The UltraSPARC IV processor does not implement the restriction to protect against all available virtual processors being disabled.
1104	**Parking a Virtual Processor** Parking a virtual processor stops the execution of new instructions and a virtual processor can be parked or unparked without a reset. Parking a virtual processor keeps the caches coherent.
1105	**XIR Steering Register (XIR Reset)** The UltraSPARC IV processor implements this register as a read / write register.
1106	**CMP Error Steering Register** The UltraSPARC IV processor implements this register with a one bit subset of the `target_id` field. Bits[5:1] of the `target_id` field are set to 0.

S.APPENDIX **D**

Formal Specification of the Memory Models

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **E**

Opcode Maps

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **F**

Memory Management Unit

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **G**

Assembly Language Syntax

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **H**

Software Considerations

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

Extending the SPARC V9 Architecture

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **J**

Programming with the Memory Models

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **K**

Changes from SPARC V8 to SPARC V9

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX L

Address Space Identifiers

This appendix describes address space identifiers (ASIs) in the following sections:

L.1 Address Space Identifiers and Address Spaces

A SPARC V9 processor provides an address space identifier (ASI) with every address sent to memory. The ASI does the following:

- Distinguishes between different address spaces
- Provides an attribute that is unique to an address space
- Maps internal control and diagnostics registers within a processor

The memory management hardware uses a 64-bit virtual address and an 8-bit ASI to generate a physical address. This physical address space can be accessed through virtual-to-physical address mapping or through the MMU bypass mode.

SPARC V9 also extended the limit of virtual addresses from 32 bits to 64 bits for each address space. SPARC V9 supports 32-bit addressing through masking of the upper 32 bits to 0 when the address mask (am) bit in the PSTATE register is set.

L.2 ASI Values

The SPARC V9 address space identifier (ASI) is evenly divided into restricted and unrestricted halves. ASIs in the range 00_{16}–$7F_{16}$ are restricted. ASIs in the range 80_{16}–FF_{16} are unrestricted. An attempt by nonprivileged software to access a restricted ASI causes a *privileged_action* trap.

Normal or *translating* ASIs are translated by the MMU.

Nontranslating ASIs are not translated by the MMU and instead pass through their virtual addresses as physical addresses.

Bypass ASIs, like nontranslating ASIs, are not translated by the MMU and instead pass through their virtual addresses as physical addresses. However, unlike a nontranslating ASI, access to a bypass ASI can cause a *PA_watchpoint* trap.

Implementation-dependent ASIs may or may not be translated by the MMU. See Appendix L in the Implementation Extension Supplements for a given implementation for detailed information about specific implementation-dependent ASIs.

L.3 ASI Assignments

Every load or store address in a SPARC V9 processor has an 8-bit Address Space Identifier (ASI) appended to the virtual address (VA). The VA plus the ASI fully specify the address. For instruction fetches and for data loads or stores that do not use the load or store alternate instructions, the ASI is an implicit ASI generated by the hardware. If a load alternate or store alternate instruction is used, the value of the ASI can be specified in the `%asi` register or as an immediate value in the instruction. In practice, ASIs are not only used to differentiate address spaces but are also used for other functions like referencing registers in the MMU unit.

L.3.1 Supported ASIs

TABLE L-1 lists both the SPARC V9 architecture-defined ASIs and ASIs that were not defined in SPARC V9 but are required for JPS2 processors.

ASIs marked with a closed bullet (●) are SPARC V9 architecture-defined ASIs. All operand sizes are supported when accessing one of these ASIs.

ASIs marked with an open bullet (❍) were not defined in SPARC V9 but were defined in JPS1 and are required to be implemented in all JPS1 and JPS2 processors. ASIs marked with an open square bullet (❑) were not required by SPARC V9 or JPS1 but are required to be implemented in all JPS2 processors. ASIs marked with an closed square bullet (■) were not required by SPARC V9, JPS1, or JPS2 but are implemented in the UltraSPARC IV processor. These ASIs can be used only with LDXA, STXA, LDDFA, or STDFA instructions, unless otherwise noted. An attempt to access any of these ASIs with other load or store alternate instructions (for example, using a byte-, halfword-, or word-length access) causes a *data_access_exception* trap, unless otherwise noted. Whether access with LDDA/STDA causes a *data_access_exception* trap is implementation dependent (impl. dep. # 300, 301).

The word "decoded" in the Virtual Address column of TABLE L-1 indicates that the the supplied virtual address is decoded by the processor.

Attempting to access an address space described as "Implementation dependent" in TABLE L-1 produces implementation-dependent results.

TABLE L-1 JPS2 ASIs *(1 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
00_{16}–03_{16}	●	—	—	—	—	—	Implementation dependent[1]
04_{16}	●	ASI_NUCLEUS (ASI_N)	RW	(decoded)	T	—	Implicit address space, nucleus privilege, TL > 0
05_{16}–$0B_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$0C_{16}$	●	ASI_NUCLEUS_LITTLE (ASI_NL)	RW	(decoded)	T	—	Implicit address space, nucleus privilege, TL > 0, little-endian
$0D_{16}$–$0F_{16}$	●	—	—	—	—	—	Implementation dependent[1]
10_{16}	●	ASI_AS_IF_USER_PRIMARY (ASI_AIUP)	RW[2]	(decoded)	T	—	Primary address space, user privilege
11_{16}	●	ASI_AS_IF_USER_SECONDARY (ASI_AIUS)	RW[2]	(decoded)	T	—	Secondary address space, user privilege
12_{16}–13_{16}	●	—	—	—	—	—	Implementation dependent[1]
14_{16}	❍	ASI_PHYS_USE_EC	RW[3,4]	(decoded)	B	—	Physical address external cacheable only
15_{16}	❍	ASI_PHYS_BYPASS_EC_WITH_EBIT	RW[3]	(decoded)	B	—	Physical address, noncacheable, with side effect
16_{16}–17_{16}	●	—	—	—	—	—	Implementation dependent[1]

TABLE L-1 JPS2 ASIs *(2 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
18_{16}	●	ASI_AS_IF_USER_PRIMARY_LITTLE (ASI_AIUPL)	RW[2]	(decoded)	T	—	Primary address space, user privilege, little-endian
19_{16}	●	ASI_AS_IF_USER_SECONDARY_LITTLE (ASI_AIUSL)	RW[2]	(decoded)	T	—	Secondary address space, user privilege, little-endian
$1A_{16}$–$1B_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$1C_{16}$	❍	ASI_PHYS_USE_EC_LITTLE (ASI_PHYS_USE_EC_L)	RW [3,4]	(decoded)	B	—	Physical address, external cacheable only, little-endian
$1D_{16}$	❍	ASI_PHYS_BYPASS_EC_WITH_EBIT_LITTLE (ASI_PHYS_BYPASS_EC_WITH_EBIT_L)	RW [3]	(decoded)	B	—	Physical address, noncacheable, with side effect, little-endian
$1E_{16}$–23_{16}	●	—	—	—	—	—	Implementation dependent[1]
24_{16}	❍	ASI_NUCLEUS_QUAD_LDD	R[5,9]	(decoded)	T	—	Cacheable, 128-bit (quad) atomic load
25_{16}–$2B_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$2C_{16}$	❍	ASI_NUCLEUS_QUAD_LDD_LITTLE, (ASI_NUCLEUS_QUAD_LDD_L)	R[5,9]	(decoded)	T	—	Cacheable, 128-bit (quad) atomic load, little-endian
$2D_{16}$–$2F_{16}$	●	—	—	—	—	—	Implementation dependent[1]
30_{16}–33_{16}	■		—	—	—	—	Diagnostic Support
34_{16}	❑	ASI_QUAD_LDD_PHYS (ASI_ATOMIC_QUAD_LDD_PHYS†)	R[5,9]	(decoded)	B	—	128-bit atomic load
35_{16}–37_{16}	●	—	—	—	—	—	Implementation dependent[1]
38_{16}–$3B_{16}$	■		—	—	—	—	Diagnostic Support
$3C_{16}$	❑	ASI_QUAD_LDD_PHYS_LITTLE (ASI_QUAD_LDD_PHYS_L, ASI_ATOMIC_QUAD_LDD_PHYS_LITTLE†, ASI_ATOMIC_QUAD_LDD_PHYS_L†)	R[5,9]	(decoded)	B	—	128-bit atomic load, little-endian

TABLE L-1 JPS2 ASIs *(3 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$3D_{16}$–40_{16}	●	—	—	—	—	—	Implementation dependent[1]
40_{16}	■		—	—	—	—	Initialization Support
41_{16}	❑	MTP control/status (shared)					
41_{16}	❑	ASI_CORE_AVAILABLE	R	00_{16}	N	shared	Core Available Register
41_{16}	❑	ASI_CORE_ENABLE_STATUS	R	10_{16}	N	shared	Core Enabled Register
41_{16}	❑	ASI_CORE_ENABLE	RW	20_{16}	N	shared	Core Enable Register
41_{16}	❑	ASI_XIR_STEERING	R/RW	30_{16}	N	shared	XIR Steering Register Implementation dependent (impl. dep. #1105)
41_{16}	❑	ASI_ERROR_STEERING	RW	40_{16}	N	shared	MTP Error Steering Register
41_{16}	❑	ASI_CORE_RUNNING_RW	RW	50_{16}	N	shared	Core Running Register, general access.
41_{16}	❑	ASI_CORE_RUNNING_STATUS	R	58_{16}	N	shared	Core Running Status Register
41_{16}	❑	ASI_CORE_RUNNING_W1S	W	60_{16}	N	shared	Core Running Register, general access. Write one to set bits
41_{16}	❑	ASI_CORE_RUNNING_W1C	W	68_{16}	N	shared	Core Running Register, general access. Write one to set bits
42_{16}–44_{16}	■		—	—	—	—	Diagnostic Support
45_{16}	❍	ASI_DCU_CONTROL_REGISTER (ASI_DCUCR)	RW	0_{16}	N	/core	Data Cache Unit Control Register
46_{16}–47_{16}	■		—	—	—	—	Diagnostic Support
48_{16}	❍	ASI_INTR_DISPATCH_STATUS (ASI_MONDO_SEND_CTRL)	R[5]	0_{16}	N	/core	Interrupt vector dispatch status
49_{16}	❍	ASI_INTR_RECEIVE (ASI_MONDO_RECEIVE_CTRL)	RW	0_{16}	N	/core	Interrupt vector receive status
$4A_{16}$	❍	(reserved for interconnect configuration)	—	—	—	—	Implementation dependent[1]
$4B_{16}$	■		—	—	—	—	Diagnostic Support
$4C_{16}$	❍	ASI_ASYNC_FAULT_STATUS (ASI_AFSR)	RW[14]	0_{16}	N	/core	Async fault status register

TABLE L-1 JPS2 ASIs *(4 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
	■	ASI_ASYNC_FAULT_STATUS2 (ASI_AFSR2)	RW[14]	8_{16}	N	/core	Async fault status register 2
$4D_{16}$	❍	ASI_ASYNC_FAULT_ADDR (ASI_AFAR)	RW[14]	0_{16}	N	/core	Async fault address register
	■	ASI_ASYNC_FAULT_ADDR2 (ASI_AFAR2)	RW[14]	8_{16}	N	/core	Async fault address register 2
$4E_{16}$	■		—	—	—	—	Diagnostic Support
$4F_{16}$	❑	Scratchpad registers					
	❑	ASI_SCRATCHPAD_0_REG	RW	0_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_1_REG	RW	8_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_2_REG	RW	10_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_3_REG	RW	18_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_4_REG	RW	20_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_5_REG	RW	28_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_6_REG	RW	30_{16}	N	/core	Implementation dependent[13]
	❑	ASI_SCRATCHPAD_7_REG	RW	38_{16}	N	/core	Implementation dependent[13]
50_{16}	❍	ASI_IMMU_...					
	❍	ASI_IMMU_TAG_TARGET_REG	R[5]	0_{16}	N	/core	IMMU tag target register
	❍	ASI_IMMU_SFSR	RW	18_{16}	N	/core	IMMU sync fault status register
	❍	ASI_IMMU_TSB_BASE	RW	28_{16}	N	/core	IMMU TSB base register
	❍	ASI_IMMU_TAG_ACCESS	RW	30_{16}	N	/core	IMMU TLB tag access register
	❍	ASI_IMMU_TSB_PEXT_REG	RW	48_{16}	N	/core	IMMU TSB primary extension register
	❍	ASI_IMMU_TSB_NEXT_REG	RW	58_{16}	N	/core	IMMU TSB nucleus extension register
51_{16}	❍	ASI_IMMU_TSB_8KB_PTR_REG	R[5]	0_{16}	N	/core	IMMU TSB 8-Kbyte pointer register
52_{16}	❍	ASI_IMMU_TSB_64KB_PTR_REG	R[5]	0_{16}	N	/core	IMMU TSB 64-Kbyte pointer register

TABLE L-1 JPS2 ASIs *(5 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
53_{16}	❍	ASI_SERIAL_ID‡ (*semantics and encoding are implementation dependent)*	R[5]	0_{16}[12]	N	/core	*Implementation dependent* (impl. dep. #258) [7]
54_{16}	❍	ASI_ITLB_DATA_IN_REG	W[10]	0_{16}	N	/core	IMMU TLB data in register
55_{16}	❍	ASI_ITLB_DATA_ACCESS_REG	RW	0_{16}–$20FF8_{16}$	N	/core	IMMU TLB data access register
55_{16}	●	*Implementation dependent* (impl. dep. #239)		40000_{16}–$60FF8_{16}$	N	—	Implementation dependent [1]
56_{16}	❍	ASI_ITLB_TAG_READ_REG	R[5]	<17:0>	N	/core	IMMU TLB tag read register
57_{16}	❍	ASI_IMMU_DEMAP	W[10]	(decoded)	N		IMMU TLB demap
58_{16}	❍	ASI_DMMU_...					
	❍	ASI_DMMU_TAG_TARGET_REG	R[5]	0_{16}	N	/core	DMMU tag target register
	❍	ASI_PRIMARY_CONTEXT_REG	RW	8_{16}	N	/core	I/D MMU primary context register
	❍	ASI_SECONDARY_CONTEXT_REG	RW	10_{16}	N	/core	DMMU secondary context register
	❍	ASI_DMMU_SFSR	RW	18_{16}	N	/core	DMMU sync fault status register
	❍	ASI_DMMU_SFAR	RW	20_{16}	N	/core	DMMU sync fault address register
	❍	ASI_DMMU_TSB_BASE	RW	28_{16}	N	/core	DMMU TSB register
	❍	ASI_DMMU_TAG_ACCESS	RW	30_{16}	N	/core	DMMU TLB tag access register
	❍	ASI_DMMU_VA_WATCHPOINT_REG	RW	38_{16}	N	/core	DMMU VA data watchpoint register
	❍	ASI_DMMU_PA_WATCHPOINT_REG	RW	40_{16}	N	/core	DMMU PA data watchpoint register
	❍	ASI_DMMU_TSB_PEXT_REG	RW	48_{16}	N	/core	DMMU TSB primary ext register
	❍	ASI_DMMU_TSB_SEXT_REG	RW	50_{16}	N	/core	DMMU TSB secondary ext register
	❍	ASI_DMMU_TSB_NEXT_REG	RW	58_{16}	N	/core	DMMU TSB nucleus ext register
59_{16}	❍	ASI_DMMU_TSB_8KB_PTR_REG	R[5]	0_{16}	N	/core	DMMU TSB 8-K pointer register
$5A_{16}$	❍	ASI_DMMU_TSB_64KB_PTR_REG	R[5]	0_{16}	N	/core	DMMU TSB 64-K pointer register

TABLE L-1 JPS2 ASIs *(6 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$5B_{16}$	❍	ASI_DMMU_TSB_DIRECT_PTR_REG	R[5]	0_{16}	N	/core	DMMU TSB direct pointer register
$5C_{16}$	❍	ASI_DTLB_DATA_IN_REG	W[10]	0_{16}	N	/core	DMMU TLB data in register
$5D_{16}$	❍	ASI_DTLB_DATA_ACCESS_REG	RW	VA<18>=0	N	/core	DMMU TLB data access register
$5D_{16}$	●	*Implementation dependent* (impl. dep. #239)		40000_{16}–$60FF8_{16}$	—	—	Implementation dependent [1]
$5E_{16}$	❍	ASI_DTLB_TAG_READ_REG	R[5]	<17:0>	N	/core	DMMU TLB tag read register
$5F_{16}$	❍	ASI_DMMU_DEMAP	W[10]	(decoded)	N	/core	DMMU TLB demap
60_{16}	❍	ASI_IIU_INST_TRAP	RW	0_{16}	N	/core	Instruction breakpoint register
61_{16}–62_{16}	●	—	—	—	—	—	Implementation dependent[1]
63_{16}	❑	MTP control/status (per-core)					
63_{16}	❑	ASI_INTR_ID	RW	00_{16}	N	/core	Core Interrupt ID
63_{16}	❑	ASI_CORE_ID	R	10_{16}	N	/core	Core ID
63_{16}	■			40_{16}	N	/core	Interconnect Support
63_{16}	❑	*Implementation dependent*		48_{16}	N	/core	Virtual addresses 0x40 and above are reserved for Implementation specific registers [1]
64_{16}–65_{16}	●	—	—	—	—	—	Implementation dependent[1]
66_{16}–68_{16}	■		—	—	—	—	Diagnostic Support
69_{16}–$6E_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$6F_{16}$	❍	(reserved for ASI_BARRIER_SYNCH_P)	—	—	—	—	Implementation dependent[1]
70_{16}	❍	ASI_BLOCK_AS_IF_USER_PRIMARY (ASI_BLK_AIUP)	RW[2,8]	(decoded)	T	/core	Primary address space, block load/store, user privilege
71_{16}	❍	ASI_BLOCK_AS_IF_USER_SECONDARY (ASI_BLK_AIUS)	RW[2,8]	(decoded)	T	/core	Secondary address space, block load/store, user priv
72_{16}	■			—	—	—	Memory Control
73_{16}	■			—	—	—	E-Cache Control

TABLE L-1 JPS2 ASIs *(7 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
74_{16}	■	–		—	—	—	Diagnostic Support
75_{16}	■	–		—	—	—	E-Cache Control
76_{16}	■	–		—	—	—	Diagnostic Support
77_{16}	❍	ASI_INTR_DATA0_W	W[10]	40_{16}	N	/core	Outgoing interrupt vector data Register 0 H
77_{16}	❍	ASI_INTR_DATA1_W	W[10]	48_{16}	N	/core	Outgoing interrupt vector data Register 0 L
77_{16}	❍	ASI_INTR_DATA2_W	W[10]	50_{16}	N	/core	Outgoing interrupt vector data Register 1 H
77_{16}	❍	ASI_INTR_DATA3_W	W[10]	58_{16}	N	/core	Outgoing interrupt vector data Register 1 L
77_{16}	❍	ASI_INTR_DATA4_W	W[10]	60_{16}	N	/core	Outgoing interrupt vector data Register 2 H
77_{16}	❍	ASI_INTR_DATA5_W	W[10]	68_{16}	N	/core	Outgoing interrupt vector data Register 2 L
77_{16}	❍	ASI_INTR_DISPATCH_W	W[10]	70_{16}	N	/core	Interrupt vector dispatch
77_{16}	❑	ASI_INTR_DISPATCH_EXT_W	W[10]	78_{16}	N	/core	Interrupt vector dispatch extension register. VA<13:0>=0x78 VA<63:14>= Target and source information. Implementation Dependent (impl. dep. #1101)
77_{16}	❍	ASI_INTR_DATA6_W	W[10]	80_{16}	N	/core	Outgoing interrupt vector data Register 3 H
77_{16}	❍	ASI_INTR_DATA7_W	W[10]	88_{16}	N	/core	Outgoing interrupt vector data Register 3 L
77_{16}	❍	ASI_INTR_DISPATCH_W	W[10]	0100000070 $_{16}$– 8FFFFFC07 0_{16}	N	/core	Interrupt vector dispatch
78_{16}	❍	ASI_BLOCK_AS_IF_USER _PRIMARY_LITTLE (ASI_BLK_AIUPL)	RW [2,8]	0_{16}	T	/core	Primary address space, block load/store, user privilege, little-endian
79_{16}	❍	ASI_BLOCK_AS_IF_USER _SECONDARY_LITTLE (ASI_BLK_AIUSL)	RW[2,8]	0_{16}	T	/core	Secondary address space, block load/store, user privilege, little-endian

TABLE L-1 JPS2 ASIs *(8 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$7A_{16}$-$7D_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$7E_{16}$	■						Diagnostic Support
$7F_{16}$	❍	ASI_INTR_DATA0_R	R[5]	40_{16}	N	/core	Incoming interrupt vector data Register 0 H
$7F_{16}$	❍	ASI_INTR_DATA1_R	R[5]	48_{16}	N	/core	Incoming interrupt vector data Register 0 L
$7F_{16}$	❍	ASI_INTR_DATA2_R	R[5]	50_{16}	N	/core	Incoming interrupt vector data Register 1 H
$7F_{16}$	❍	ASI_INTR_DATA3_R	R[5]	58_{16}	N	/core	Incoming interrupt vector data Register 1 L
$7F_{16}$	❍	ASI_INTR_DATA4_R	R[5]	60_{16}	N	/core	Incoming interrupt vector data Register 2 H
$7F_{16}$	❍	ASI_INTR_DATA5_R	R[5]	68_{16}	N	/core	Incoming interrupt vector data Register 2 L
$7F_{16}$	❍	ASI_INTR_DATA6_R	R[5]	80_{16}	N	/core	Incoming interrupt vector data Register 3 H
$7F_{16}$	❍	ASI_INTR_DATA7_R	R[5]	88_{16}	N	/core	Incoming interrupt vector data Register 3 L
80_{16}	●	ASI_PRIMARY (ASI_P)	RW	—	T	—	Implicit primary address space
81_{16}	●	ASI_SECONDARY (ASI_S)	RW	—	T	—	Secondary address space
82_{16}	●	ASI_PRIMARY_NO_FAULT (ASI_PNF)	R[5]	—	T	—	Primary address space, no fault
83_{16}	●	ASI_SECONDARY_NO_FAULT (ASI_SNF)	R[5]	—	T	—	Secondary address space, no fault
84_{16}–87_{16}	●	—	—	—	—	—	Reserved
88_{16}	●	ASI_PRIMARY_LITTLE (ASI_PL)	RW	—	T	—	Implicit primary address space, little-endian
89_{16}	●	ASI_SECONDARY_LITTLE (ASI_SL)	RW	—	T	—	Secondary address space, little-endian
$8A_{16}$	●	ASI_PRIMARY_NO_FAULT _LITTLE (ASI_PNFL)	R[5]	—	T	—	Primary address space, no fault, little-endian
$8B_{16}$	●	ASI_SECONDARY_NO_FAULT _LITTLE (ASI_SNFL)	R[5]	—	T	—	Physical address, noncacheable, with side effect, little-endian
$8C_{16}$–BF_{16}	●	—	—	—	—	—	*Reserved*

TABLE L-1 JPS2 ASIs *(9 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$C0_{16}$	❍	ASI_PST8_PRIMARY (ASI_PST8_P)	W[11]	(decoded)	T	—	Primary address space, 8×8-bit partial store
$C1_{16}$	❍	ASI_PST8_SECONDARY (ASI_PST8_S)	W[11]	(decoded)	T	—	Secondary address space, 8x8-bit partial store
$C2_{16}$	❍	ASI_PST16_PRIMARY (ASI_PST16_P)	W[11]	(decoded)	T	—	Primary address space, 4×16-bit partial store
$C3_{16}$	❍	ASI_PST16_SECONDARY (ASI_PST16_S)	W[11]	(decoded)	T	—	Secondary address space, 4×16-bit partial store
$C4_{16}$	❍	ASI_PST32_PRIMARY (ASI_PST32_P)	W[11]	(decoded)	T	—	Primary address space, 2x32-bit partial store
$C5_{16}$	❍	ASI_PST32_SECONDARY (ASI_PST32_S)	W[11]	(decoded)	T	—	Secondary address space, 2×32-bit partial store
$C6_{16}$–$C7_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$C8_{16}$	❍	ASI_PST8_PRIMARY_LITTLE (ASI_PST8_PL)	W[11]	(decoded)	T	—	Primary address space, 8x8-bit partial store, little-endian
$C9_{16}$	❍	ASI_PST8_SECONDARY_LITTLE (ASI_PST8_SL)	W[11]	(decoded)	T	—	Secondary address space, 8×8-bit partial store, little-endian
CA_{16}	❍	ASI_PST16_PRIMARY_LITTLE (ASI_PST16_PL)	W[11]	(decoded)	T	—	Primary address space, 4x16-bit partial store, little-endian
CB_{16}	❍	ASI_PST16_SECONDARY_LITTLE (ASI_PST16_SL)	W[11]	(decoded)	T	—	Secondary address space, 4×16-bit partial store, little-endian
CC_{16}	❍	ASI_PST32_PRIMARY_LITTLE (ASI_PST32_PL)	W[11]	(decoded)	T	—	Primary address space, 2×32-bit partial store, little-endian
CD_{16}	❍	ASI_PST32_SECONDARY_LITTLE (ASI_PST32_SL)	W[11]	(decoded)	T	—	Second address space, 2×32-bit partial store, little-endian
CE_{16}–CF_{16}	●	—	—	—	—	—	Implementation dependent[1]
$D0_{16}$	❍	ASI_FL8_PRIMARY (ASI_FL8_P)	RW[8]	(decoded)	T	—	Primary address space, one 8-bit floating-point load / store

TABLE L-1 JPS2 ASIs *(10 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$D1_{16}$	❍	ASI_FL8_SECONDARY (ASI_FL8_S)	RW[8]	(decoded)	T	—	Second address space, one 8-bit floating-point load / store
$D2_{16}$	❍	ASI_FL16_PRIMARY (ASI_FL16_P)	RW[8]	(decoded)	T	—	Primary address space, one 16-bit floating-point load / store
$D3_{16}$	❍	ASI_FL16_SECONDARY (ASI_FL16_S)	RW[8]	(decoded)	T	—	Second address space, one 16-bit floating-point load / store
$D4_{16}$–$D7_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$D8_{16}$	❍	ASI_FL8_PRIMARY_LITTLE (ASI_FL8_PL)	RW[8]	(decoded)	T	—	Primary address space, one 8-bit floating point load / store, little-endian
$D9_{16}$	❍	ASI_FL8_SECONDARY_LITTLE (ASI_FL8_SL)	RW[8]	(decoded)	T	—	Second address space, one 8-bit floating point load / store, little-endian
DA_{16}	❍	ASI_FL16_PRIMARY_LITTLE (ASI_FL16_PL)	RW[8]	(decoded)	T	—	Primary address space, one 16-bit floating-point load / store, little-endian
DB_{16}	❍	ASI_FL16_SECONDARY_LITTLE (ASI_FL16_SL)	RW[8]	(decoded)	T	—	Second address space, one 16-bit floating point load / store, little-endian
DC_{16}–DF_{16}	●	—	—	—	—	—	Implementation dependent[1]
$E0_{16}$	❍	ASI_BLOCK_COMMIT_PRIMARY (ASI_BLK_COMMIT_P)	W[6,11]	(decoded)	T	—	Primary address space, 8x8- byte block store commit operation
$E1_{16}$	❍	ASI_BLOCK_COMMIT_SECONDARY (ASI_BLK_COMMIT_S)	W[6,11]	(decoded)	T	—	Secondary address space, 8x8-byte block store commit operation
$E2_{16}$–EE_{16}	●	—	—	—	—	—	Implementation dependent[1]
EF_{16}	❍	(reserved for ASI_BARRIER_SYNCH)	—	—	—	—	Implementation dependent[1]
$F0_{16}$	❍	ASI_BLOCK_PRIMARY (ASI_BLK_P)	RW[8]	(decoded)	T	—	Primary address space, 8x8- byte block load / store
$F1_{16}$	❍	ASI_BLOCK_SECONDARY (ASI_BLK_S)	RW[8]	(decoded)	T	—	Secondary address space, block load / store

TABLE L-1 JPS2 ASIs *(11 of 11)*

ASI Value	V9 (●); JPS1 (❍); JPS2 (❑) USIV (■)	ASI Name (and Abbreviation)	Access Type(s)	Virtual Address (VA)	T/ Non-T/ Bypass	Shared /Per-Core	Description
$F2_{16}$–$F7_{16}$	●	—	—	—	—	—	Implementation dependent[1]
$F8_{16}$	❍	ASI_BLOCK_PRIMARY_LITTLE (ASI_BLK_PL)	RW[8]	(decoded)	T	—	Primary address space, block load/store, little endian
$F9_{16}$	❍	ASI_BLOCK_SECONDARY_LITTLE (ASI_BLK_SL)	RW[8]	(decoded)	T	—	Secondary address space, block load/store, little endian
FA_{16}–FF_{16}	●	—	—	—	—	—	Implementation dependent[1]

† This ASI name has been changed, for consistency; although use of this name is deprecated and software should use the new name, the old name is listed here for compatibility

‡ This ASI was named ASI_DEVICE_ID+SERIAL_ID in older documents

1 Implementation dependent ASI (impl. dep. #29); available for use by implementors. Software that references this ASI may not be portable.

2 Causes a *data_access_exception* trap if the page being accessed is privileged.

3 8-bit, 16-bit, 32-bit, and 64-bit accesses are allowed.

4 Can be used with LDSTUBA, SWAPA, CAS(X)A.

5 Read(load)-only ASI; an attempted store or atomic load-store to this ASI causes a *data_access_exception* exception.

6 May only be used in an STDDA instruction. Use of this ASI in any other store instruction causes a *data_access_exception*.

7 Implementation dependent ASI (impl. dep. #258), intended for use for a part identification number that is unique to each processor. Software that references this ASI may not be portable.

8 May only be used in a LDDFA or STDFA instruction. Use in any other load or store instruction causes a *data_access_exception* exception.

9 May only be used in an LDDA instruction. Use of this ASI in any other load instruction causes a *data_access_exception*.

10 Write(store)-only ASI; a load from this ASI causes a *data_access_exception*.

11 Write(store)-only ASI; a load from this ASI causes an exception

12 An implementation may or may not decode the virtual address when this ASI is accessed, but for future compatibility software should always supply VA = 0.

13 Implementation-dependent ASI (impl. dep. #302). The ASI is reserved exclusively for this use, but may not be present in all implementations. Software that references this ASI may not be portable.

14 **IMPL. DEP. #305:** The effect of write is implementation dependent.

L.3.2 Special Memory Access ASIs

See *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* for description of the special memory access ASIs.

S.APPENDIX **M**

Interrupt Handling

Please refer to the *SPARC® Joint Programming Specification 2 (JPS2): Common Specification* and *JPS1 UltraSPARC III Cu Supplement*.

S.APPENDIX **N**

Reset, RED_state, and Error_state

N.1 Machine States After Reset

TABLE N-1 and TABLE N-2 list the states of the newly added registers and fields at Hard POR and System reset (Soft POR). These new added registers or fields are unchanged after Watchdog Reset (WDR), External Initiated Reset (XIR), Software-Initiated Reset (SIR), or after entering RED_state.

TABLE N-1 UltraSPARC IV New Defined Per-Core Register/Field Reset Machine State *(1 of 2)*

No.	New Register	Field	Hard_POR State	System Reset (Soft POR)	Comments
1.	ASI_ECACHE_CTRL (75_{16}, VA = 0x00) in both cores	All	0	Unchanged	Default to direct-mapped L2-cache
2.	ASI_ECACHE_CTRL2 (75_{16}, VA==0x08) in both cores	All	Undefined	Unchanged	Unused in the UltraSPARC IV processor
3.	ASI_CORE_ID – Core 0	max_core_id	000001	Unchanged	2 cores per UltraSPARC IV chip
		Core ID	000000	Unchanged	Core ID
	ASI_CORE_ID – Core 1	max_core_id	000001	Unchanged	2 cores per UltraSPARC IV chip
		Core ID	000001	Unchanged	Core ID
4.	ASI_INTR_ID	All	Undefined	Unchanged	Undefined for both cores
5.	ASI_ESTATE_ERROR_EN_REG	<22:19>	0	0	

TABLE N-1 UltraSPARC IV New Defined Per-Core Register / Field Reset Machine State *(2 of 2)*

No.	New Register	Field	Hard_POR State	System Reset (Soft POR)	Comments
6.	ASI_CESR_ID	<7:0>	0	Unchanged	
7.	ASI_DCU_CONTROL_REGISTER	wih <4>	0	0	Default to use pa<8:6> to index W-cache
8	Dispatch Control Register (ASR 18)	obs <11:6>	0	Unchanged	

TABLE N-2 UltraSPARC IV New Defined Shared Register / Field Reset Machine State *(1 of 3)*

No.	New Register	Field	Hard POR	System Reset (Soft POR)	Comments
1.	ASI_ECACHE_CFG_TIMING_CTRL (73_{16}, VA = 00_{16})	ec_assoc	0	Unchanged	Default to direct-mapped L2-cache
		trace_out	11	Unchanged	Default to 6 cycles (6-6-5)
		trace_in	010	Unchanged	Default to 5 cycles (6-6-5)
		ec_clock	11	Unchanged	Default to 6:1 L2-cache clock ratio
		ec_size	10	Unchanged	Default to 8 MB L2-cache
		ec_turn_rw	1	Unchanged	Default to 2 cycles
		Others	0	Unchanged	
2.	New Sun Fireplane Interconnect Clock Ratio in SAFARI_CONFIG and SAFARI_CONFIG_2[1]	clk<2>, <1:0>	0,10	Unchanged	Default to 6:1 system clock ratio
3.	SAFARI_CONFIG[2]	<26:17>	= ASI_INTR_ID <9:0> of core 0	= ASI_INTR_ID <9:0> of core 0	Default to reflect core 0's intr_id
4.	Mem_Timing5_CTL	All	Undefined	Unchanged	
5.	Mem_Address_CTL	<63>	Undefined	Unchanged	Default to disable internal banking
6	ASI_CORE_AVAILABLE (41_{16}, VA = 00_{16})	<63:0>	3 (decimal)	3 (decimal)	UltraSPARC IV hardware always sets 3 (decimal) to this register

TABLE N-2 UltraSPARC IV New Defined Shared Register/Field Reset Machine State *(2 of 3)*

No.	New Register	Field	Hard POR	System Reset (Soft POR)	Comments
7.	ASI_CORE_ENABLE_STATUS (41_{16}, VA =10_{16})	<63:2>	0	0	Cores 63-2 are not implemented
		<1:0>	Value of ASI_CORE_ENABLE<1:0> at the time of reset deassertion	Value of ASI_CORE_ENABLE<1:0> at the time of reset deassertion	
8.	ASI_XIR_STEERING (41, VA = 30_{16})$_{16}$	<63:2>	0	0	Cores 63-2 are not implemented
		<1:0>	Value of ASI_CORE_ENABLE_STATUS <1:0> at the time of reset deassertion	Value of ASI_CORE_ENABLE_STATUS<1:0> at the time of reset deassertion	
9.	ASI_CORE_ENABLE (41_{16}, VA = 20_{16})	<63:2>	0	0	Cores 63-2 are not implemented
		<1:0>	11	Unchanged	Both cores are enabled by default. During reset, this register could be overwritten by the JTAG controller.
10.	ASI_CORE_RUNNING (41_{16}, VA = 50_{16}, 60_{16}, 68_{16})	<63:2>	0	0	Cores 63-2 are not implemented
		<1:0>	Deassertion: = 01, if core 0 is enabled; = 10, otherwise	Deassertion: = 01, if core 0 is enabled; = 10, otherwise	By default, only the lowest enabled core will be running after reset. The JTAG controller can overwrite this default setting. However, only enabled cores can become running.

TABLE N-2 UltraSPARC IV New Defined Shared Register / Field Reset Machine State *(3 of 3)*

No.	New Register	Field	Hard POR	System Reset (Soft POR)	Comments
11.	ASI_CORE_RUNNING_STATUS (41_{16}, VA = 58_{16})	<63:2>	0	0	Cores 63-2 are not implemented
		<1:0>	= ASI_CORE_RUNNING <1:0>	= ASI_CORE_RUNNING<1:0>	0 when the corresponding core is successfully parked
12.	ASI_ERROR_STEERING (41_{16}, VA = 40_{16})	<63:1>	0	0	Cores 63-2 are not implemented
		<0>	Deassertion: = 0, if core 0 is running; = 1, otherwise	Deassertion: = 0, if core 0 is running; = 1, otherwise	By default, this register encodes the lowest running core after reset. However, the JTAG controller can overwrite the default value.

1. Except for the Sun Fireplane Interconnect Clock Ratio, SAFARI_CONFIG_2 has the same reset values as the SAFARI_CONFIG in the UltraSPARC III Cu processor.
2. Except for the int_id field, the SAFARI_CONFIG has the same reset values as the SAFARI_CONFIG_2 register.

Note – AFAR2 (ASI 0x4C, VA 0x8) has an unknown state after Hard POR, and is unchanged after all other types of resets.

S.APPENDIX **O**

Error Handling

The UltraSPARC IV implements a secondary Asynchronous Fault Status Register (AFSR2) and a secondary Asynchronous Fault Address Register (AFAR2). They are designed to capture the first event that sets the primary AFSR bit. They have the same layout as the primary AFSR/AFAR. Clearing the bits in the AFSR1 automatically clear the corresponding bits in the AFSR2. These two registers have the following accessed addresses:

- AFSR2: ASI = 0x4C, VA<63:0> = 0x8
- AFAR2: ASI = 0x4D, VA<63:0> = 0x8

S.APPENDIX **P**

Performance Instrumentation

Performance instrumentation consists of processor event counters that can be used to gather statistics during program execution. Approximately seventy events can be monitored, two at a time, to gain information about the performance of the processor. Cache miss counts and stall times, for example, can be measured using two, 32-bit Performance Instrumentation Counters (PICs). Some event counting can be synthesized from the event counters available to provide additional program execution statistics.

The counters can be monitored during program execution to gather on-going statistics or reconfigure during steady-state program execution to gather statistics for more than two events.

The Performance Control Register (PCR) is used to select the events to monitor and provide control for counting in privileged and/or non-privileged modes.

Each of the two 32-bit PICs, PICL and PICU, can accumulate over four billion events before wrapping. Event logging counts can be extended by periodically reading contents of the PICs to detect and avoid an overflow. An interrupt can be enabled on a counter overflow. Additional event or stall cycle statistics can be collected by reading the PIC counts between repeated program executions.

This appendix describes the performance instrumentation features in the following sections:

- Section P.1, *Performance Instrumentation Operation*
- Section P.2, *Performance Control Register (PCR)*
- Section P.3, *Performance Instrumentation Counter (PIC) Register*
- Section P.4, *Pipeline Counters*
- Section P.5, *Cache Access Counters*
- Section P.6, *Memory Controller Counters*
- Section P.7, *Data Locality Counters for Scalable Shared Memory Systems*
- Section P.8, *Miscellaneous Counters*
- Section P.9, *PCR.SL and PCR.SU Encodings*

Supervisor/User Mode

Access to the PCR is restricted to Supervisor software. User software accessing the PCR causes a privileged_opcode trap.

Supervisor software controls user accessibility to the PIC counters by setting the PCR.PRIV field. When PCR.PRIV = 1 (supervisor access only), an attempt by User software to access the PIC register causes a privileged_action trap.

Also related to Supervisor / User mode, the mode in which the counters are enabled to count is controlled by setting the PCR.UT (User Trace) and PCR.ST (System Trace) bits.

P.1 Performance Instrumentation Operation

FIGURE P-1 shows how an operating system might use the performance instrumentation features to provide event monitoring services.

Setup the PCR register as desired to select two events and in which modes data should be collected. The monitoring must consider the real effects of the computer that includes calls to the system and interrupts. When used, the PCR register is considered part of a process state and must be saved and restored when switching process contexts.

Multiple data collection times can be done while the program executes to show on-going statistics.

P.1.1 Gathering Data for More Than Two Events

When more than two events need to be monitored, the program, program sequence, or program loop need to be run again with the new events enabled. It is not possible to monitor more than two events at any given time.

P.1.2 Gathering Data in Privileged and Non-Privileged Modes

The PCR has mode bits to enable the counters in privileged mode, non-privileged mode, or to count when in either mode. The mode setting affects both counters.

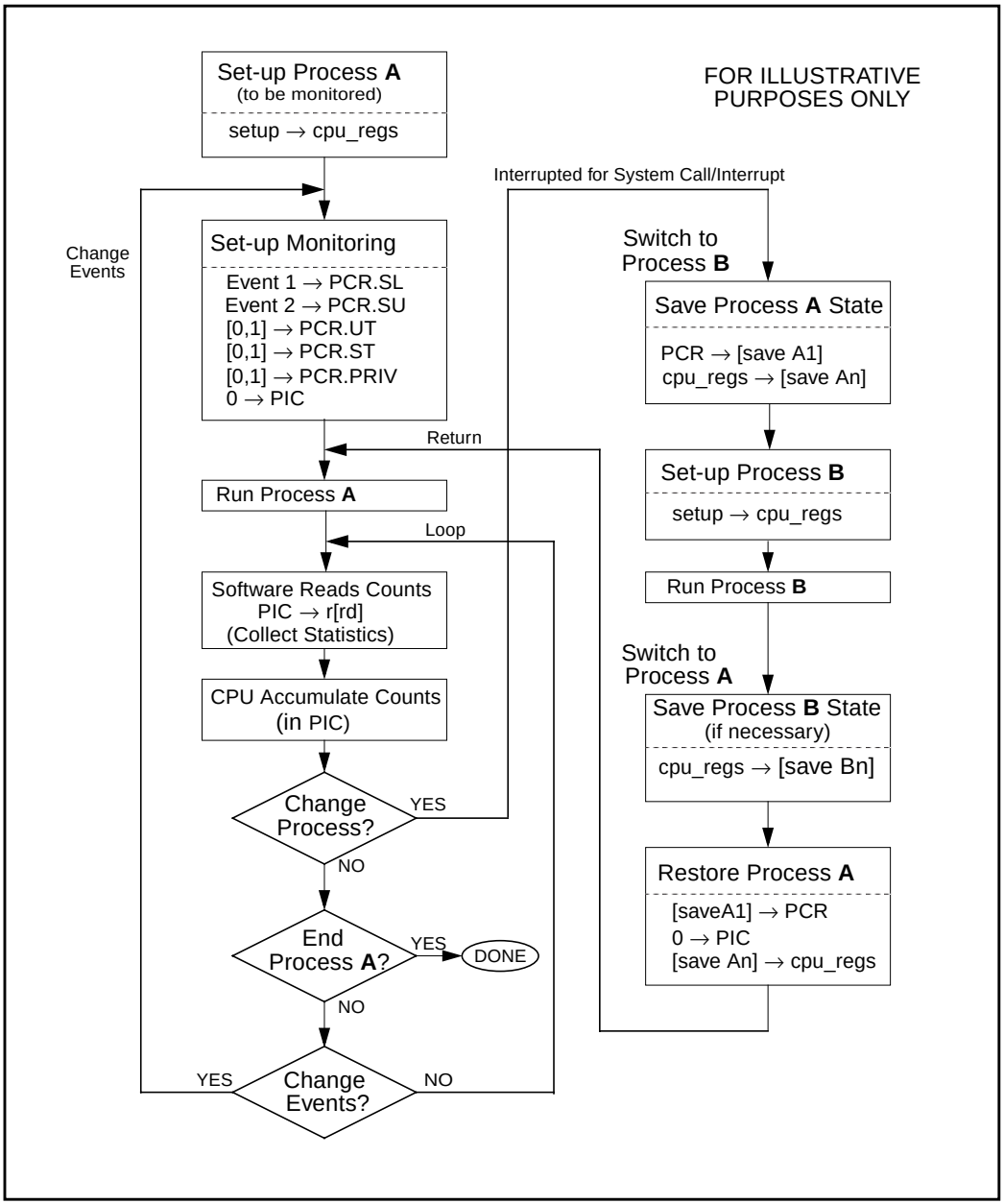


FIGURE P-1 Operational Flow Diagram for Controlling Event Counters

P.1.3 Performance Instrumentation Implementations

Counting events and cycle stalls is sometimes complex because of the dynamic conditions and cancelled activities. In addition, there were anomalies in the monitors inside the previous processors that were fixed for the UltraSPARC IV processor. These anomalies, both fixed and featured, are described for each event and cycle stall counter.

P.1.4 Performance Instrumentation Accuracy

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

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

P.2 Performance Control Register (PCR)

The 64-bit PCR and PIC are accessed through read / write Ancillary State Register (ASR) instructions (RDASR / WRASR). PCR and PIC are located at ASRs 16 (10_{16}) and 17 (11_{16}), respectively.

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

The PCR is a read / write register used to control the counting of performance monitoring events. FIGURE P-2 shows the details of the PCR. TABLE P-1 describes the various fields of the PCR. Counts are collected in the PIC register (see Section P.3, *Performance Instrumentation Counter (PIC) Register*).

PCR - Performance Control Register	ASR Register

The PCR selects the events and controls the operating modes of the Performance Instrumentation Counters (PICs).			
ASR 16_{10}	64-bit Read/Write	Privileged Mode, otherwise *privileged_action* trap.	Reset: 0x0000.0000

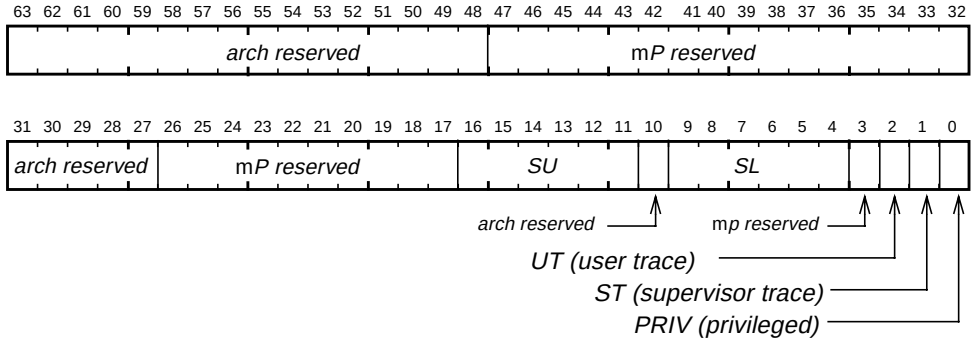


FIGURE P-2 Performance Control Register

TABLE P-1 PCR Bit Description

Bit	Field	Description
63:48	—	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
47:32	—	Unused UltraSPARC IV. Read zero, write zero, or write value read previously (read-modify-write).
31:27	—	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
26:17	—	Unused UltraSPARC IV. Read zero, write zero, or write value read previously (read-modify-write).
16:11	SU	Selects 1 of up to 64 counters accessible in the upper half (bits <63:32>) of the PIC register.
10	—	Reserved by SPARC architecture. Read zero, write zero or write value read previously (read-modify-write).
9:4	SL	Selects 1 of up to 64 counters accessible in the lower half (bits <31:0>) of the PIC register.

TABLE P-1 PCR Bit Description *(Continued)*

Bit	Field	Description
3	—	Unused UltraSPARC IV. Read zero, write zero, or write value read previously (read-modify-write).
2	UT	User Trace Enable. If set to one, counts events in non-privileged mode (User).
1	ST	System Trace Enable. If set to one, counts events in privileged mode (Supervisor). **Notes:** If both `PCR.UT` and `PCR.ST` are set to one, all selected events are counted. If both `PCR.UT` and `PCR.ST` are zero, counting is disabled. `PCR.UT` and `PCR.ST` are global fields which apply to both PIC pairs.
0	PRIV	Privileged. If `PCR.PRIV` = 1, a non-privileged (`PSTATE.PRIV` = 0) attempt to access `PIC` (via a `RDPIC` or `WRPIC` instruction) will result in a *privileged_action* exception.

P.3 Performance Instrumentation Counter (PIC) Register

The difference between the values read from the PIC on two reads reflects the number of events that occurred between register reads. Software can only rely on read-to-read PIC accesses to get an accurate count and not a write-to-read of the PIC counters. FIGURE P-3 shows the details of the PIC. TABLE P-2 describes the various fields of the PIC.

PIC - Performance Instrumentation Counter register	ASR Register

The PIC register provides access to the counter values for the two events being monitored.			
ASR 17_{10}	64-bit Read/Write **Note:** Writes are designed for diagnostic and test purposes.	Accessibility depends on *PCR.PRIV* bit: 0 = Accessible in any mode 1 = Accessible in Supervisor Mode, otherwise *privileged_action* trap	Reset: 0x0000.0000

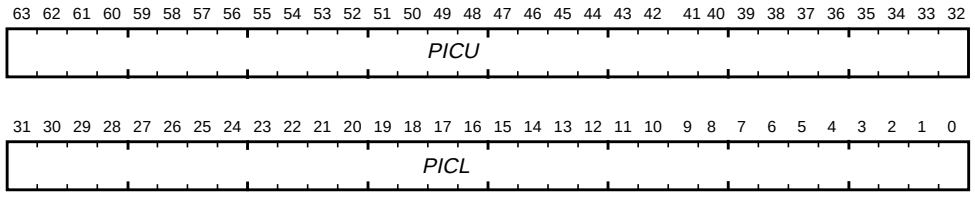


FIGURE P-3 Performance Instrumentation Counter Register

TABLE P-2 PIC Register Fields

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

P.3.1 PIC Counter Overflow Trap Operation

When a PIC counter overflows, an interrupt is generated as described in TABLE P-3.

TABLE P-3 PIC Counter Overflow Processor Compatibility Comparison

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

P.4 Pipeline Counters

P.4.1 Instruction Execution and CPU Clock Counts

The instruction execution count monitors are described in TABLE P-4 for clock and instruction execution counts.

TABLE P-4 Instruction Execution Clock Cycles and Counts

Counter	Description
`Cycle_cnt`	[`PICL 00.0000` and `PICU 00.0000`] Counts clock cycles. This counter increments the same as the SPARC V9 `TICK` register, except that cycle counting is controlled by the `PCR.UT` and `PCR.ST` fields.
`Instr_cnt`	[`PICL 00.0001` and `PICU 00.0001`] Counts the number of instructions completed. Annulled, mispredicted, or trapped instructions are not counted.

Synthesized Clocks Per Instruction (CPI)

The cycle and instruction counts can be used to calculate the average number of instructions completed per cycle: Clock cycles per instruction, CPI = Cycle_cnt / Instr_cnt.

P.4.2 IIU Event Counts

The counters listed in TABLE P-5 record branch prediction event counts for taken and untaken branches in the Instruction Issue Unit (IIU). A retired branch in the following descriptions refers to a branch that reaches the D-stage without being invalidated.

TABLE P-5 Counters for Collecting IIU Statistics

Counter	Description
IU_Stat_Br_miss_taken	[PICL 01.0101] Counts retired branches that were predicted to be taken, but in fact were not taken.
IU_Stat_Br_miss_untaken	[PICU 01.1101] Counts retired branches that were predicted to be untaken, but in fact were taken.
IU_Stat_Br_Count_taken	[PICL 01.0110] Counts retired taken branches.
IU_Stat_Br_Count_untaken	[PICU 01.1110] Counts retired untaken branches.

P.4.3 IIU Dispatch Stall Cycle Counts

IIU stall counts, listed in TABLE P-6, are the major cause of pipeline stalls (bubbles) from the instruction fetch and decode pipeline. Stalls are counted for each clock cycle at which the associated condition is true.

FIGURE P-4 illustrates the first two considerations described below.

P.4.3.1 Dispatch Counter Considerations

1. Dispatch Counters count when the buffer is empty, regardless of whether the execution pipeline can accept more instructions from the instruction queue.
2. It is difficult to associate an empty queue to the reason it is empty. Multiple reasons together or separately can cause the instruction queue to be empty. The hardware picks the most recent disruptive event that is in the Fetch Unit to choose a counter to assign the empty queue cycles.
3. Count accuracy is also subject to the conditions described in Section P.1.4, *Performance Instrumentation Accuracy* for all counters.

FIGURE P-4 Dispatch Counters

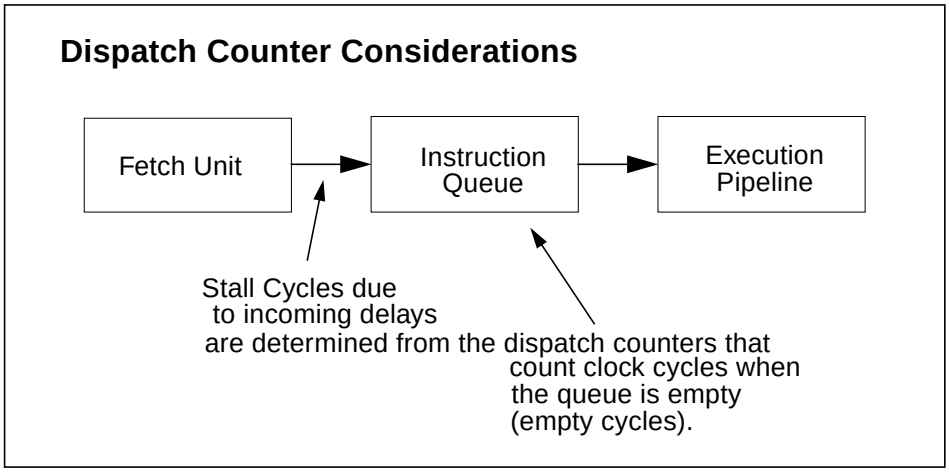


TABLE P-6 Counters for IIU Stalls

Counter	Description
`Dispatch0_IC_miss`	`[PICL 00.0010]` Counts the stall cycles due to the event that no instructions are issued because I-queue is empty from instruction cache miss. This count includes L2-cache miss processing if a L2-cache miss also occurs.[1]
`Dispatch0_mispred`	`[PICU 00.0010]` Counts the stall cycles due to the event that no instructions are issued because I-queue is empty due to branch misprediction. 1
`Dispatch0_br_target`	`[PICL 00.0011]` Counts the stall cycles due to the event that no instructions are issued because I-queue is empty due to a branch target address calculation. 1
`Dispatch0_2nd_br`	`[PICL 00.0100]` Counts the stall cycles due to the event of having two branch instructions line-up in one 4-instruction group causing the second branch in the group to be re-fetched, delaying its entrance into the I-queue. 1
`Dispatch_rs_mispred`	`[PICL 01.0111]` Counts the stall cycles due to the event that no instructions are issued because the I-queue is empty due to a Return Address Stack misprediction. 1

1. *Dispatch Counter Considerations* on page 85 for important information.

P.4.4 R-stage Stall Cycle Counts

Stalls are caused by dependency checks (data not ready for use by the instruction ready for dispatch) and by resources not being available (out-of-pipeline execution units needed, but are in-use).

The counters in TABLE P-7 count the stall cycles at the R-stage of the pipeline. Stalls are counted for each clock at which the associated condition is true.

TABLE P-7 Counters for R-stage Stalls

Counter	Description
`Rstall_storeQ`	`[PICL 00.0101]` Counts R-stage stall cycles for a store instruction which is the next instruction to be executed, but is stalled due to the store queue being full, that is, cannot hold additional stores. Up to eight entries can be in the store queue.
`Rstall_FP_use`	`[PICU 00.1011]` Counts R-stage stall cycles due to the event that the next instruction to be executed depends on the result of a preceding floating-point instruction in the pipeline that is not yet available.
`Rstall_IU_use`	`[PICL 00.0110]` Counts R-stage stall cycles due to the event that the next instruction to be executed depends on the result of a preceding integer instruction in the pipeline that is not yet available.

P.4.5 Recirculation Stall Cycle Counts

Recirculation instrumentation is implemented through the counters listed in TABLE P-8.

TABLE P-8 Counters for Recirculation

Counter	Description
`Re_DC_ovhd` **Note:** See Section P.5.6.	`[PICU 00.0100]` Counts the stall cycles from when a D-cache load misses (causes a recirculation), but L2-cache hit/miss has not been reported. Counts portion/overhead of stall cycles due to D-cache load miss from the point the load reaches D-stage (about to be recirculated) to the point L2-cache hit/miss for the load is reported.
`Re_endian_miss`	`[NA]` Event counter does not exist in the UltraSPARC IV processor.
`Re_RAW_miss`	`[PICU 10.0110]` Counts stall cycles due to recirculation when there is a load in the E-stage which has a non-bypassable read-after-write (RAW) hazard with an earlier store instruction. This condition means that load data are being delayed by completion of an earlier store.
`Re_FPU_bypass`	`[PICU 00.0101]` Counts stall cycles due to recirculation when an FPU bypass condition that does not have a direct bypass path occurs.
`Re_DC_miss`	`[PICU 00.0110]` Counts stall cycles due to loads that miss D-cache and L2-cache and get recirculated. Includes cacheable loads only.

TABLE P-8 Counters for Recirculation *(Continued)*

Counter	Description
Re_EC_miss	[PICU 00.0111] Counts stall cycles due to loads that miss D-cache and L2-cache and get recirculated. Stall cycles from the point when L2-cache miss is detected to the D-stage of the recirculated flow are counted. Includes cacheable loads only.
Re_PC_miss	[PICU 01.0000] Counts stall cycles due to recirculation when a prefetch cache miss occurs on a prefetch predicted second load.

P.5 Cache Access Counters

Instruction, data, prefetch, write, and L2-cache access events can be collected through the counters listed in TABLE P-9. Counts are updated by each cache access, regardless of whether the access will be used.

P.5.1 Instruction Cache Events

TABLE P-9 describes the counters for instruction cache events.

TABLE P-9 Counters for Instruction Cache Events

Counter	Description
IC_ref	[PICL 00.1000] Counts I-cache references. I-cache references are fetches (up to four instructions) from an aligned block of eight instructions. I-cache references are generally speculative and include instructions that are later cancelled due to mis-speculation.
IC_miss	[PICU 00.1000] Counts I-cache misses. Includes fetches from mis-speculated execution paths which are later cancelled.
IC_miss_cancelled	[PICU 00.0011] Counts I-cache misses cancelled due to mis-speculation, recycle, or other events.
ITLB_miss	[PICU 01.0001] Counts I-TLB miss traps taken.

P.5.2 Data Cache Events

TABLE P-10 describes the counters for data cache events.

TABLE P-10 Counters for Data Cache Events

Counter	Description
DC_rd	[PICL 00.1001] Counts D-cache read references (including accesses that subsequently trap). References to pages that are not virtually cacheable (TTE CV bit = 0) are not counted.
DC_rd_miss	[PICU 00.1001] Counts recirculated loads that miss the D-cache. Includes cacheable loads only.
DC_wr	[PICL 00.1010] Counts D-cache cacheable store accesses encountered (including cacheable stores that subsequently trap). Non-cacheable accesses are not counted.
DC_wr_miss	[PICU 00.1010] Counts D-cache cacheable store accesses that miss D-cache. (There is no stall or recirculation on store miss.)
DTLB_miss	[PICU 01.0010] Counts memory reference instructions which trap due to D-TLB miss.

P.5.3 Write Cache Events

TABLE P-11 describes the counters for write cache events.

TABLE P-11 Counters for Write Cache Events

Counter	Description
WC_miss	[PICU 01.0011] Counts W-cache misses.
WC_snoop_cb	[PICU 01.0100] Counts W-cache copybacks generated by a snoop from a remote processor.
WC_scrubbed	[PICU 01.0101] Counts W-cache hits to clean lines.
WC_wb_wo_read	[PICU 01.0110] Counts W-cache writebacks not requiring a read.

P.5.4 Prefetch Cache Events

TABLE P-12 describes the counters for prefetch cache events.

TABLE P-12 Counters for Prefetch Cache Events

Counter	Description
PC_MS_miss	[PICU 01.1111] Counts FP loads through the MS pipeline that miss P-cache.
PC_soft_hit	[PICU 01.1000] Counts FP loads that hit a P-cache line that was prefetched by a software-prefetch instruction.
PC_hard_hit	[PICU 01.1010] Counts FP loads that hit a P-cache line that was prefetched by a hardware prefetch.
PC_snoop_inv	[PICU 01.1001] Counts P-cache invalidates generated by a snoop from a remote processor and stores by a local processor.
PC_port0_rd	[PICL 01.0000] Counts P-cache cacheable FP loads to the first port (general-purpose load path to D-cache and P-cache via MS pipeline).
PC_port1_rd	[PICU 01.1011] Counts P-cache cacheable FP loads to the second port (memory and out-of-pipeline instruction execution loads via the A0 and A1 pipelines).

P.5.5 L2-Cache Events

The L2-cache write hit count is determined by subtraction of the read hit and the instruction hit count from the total L2-cache hit count. The L2-cache write reference count is determined by subtraction of the D-cache read miss and I-cache misses from the total L2-cache references. Because of write caching, this is not the same as D-cache write misses. TABLE P-13 describes the counter for L2-cache events.

Note – A block load or store access is counted as eight references. For atomics, the read and write events are counted individually.

TABLE P-13 Counters for L2-Cache Events

Counter	Description
EC_ref	[PICL 00.1100] Counts L2-cache reference events. A 64-byte request is counted as one reference. Includes speculative D-cache load requests that turn out to be a D-cache hit. Count includes cacheable accesses only.
EC_misses	[PICU 00.1100] Counts L2-cache miss events sent to the System Interface Unit. Includes I-cache, D-cache, P-cache, W-cache exclusive (store), read stream (BLD), write stream (BST) requests that miss L2-cache. Count includes cacheable accesses only.

TABLE P-13 Counters for L2-Cache Events *(Continued)*

Counter	Description
EC_write_hit_RTO	[PICL 00.1101] Counts W-cache exclusive requests that hit L2-cache in S, O, or O_s state and thus, do a read-to-own (RTO) bus transaction.
EC_wb	[PICU 00.1101] Counts dirty subblocks that produce writebacks due to L2-cache miss events.
EC_snoop_inv	[PICL 00.1110] Counts L2-cache invalidates generated from a snoop by a remote processor.
EC_snoop_cb	[PICU 00.1110] Counts L2-cache copybacks generated from a snoop by a remote processor.
EC_rd_miss	[PICL 00.1111] Counts L2-cache miss events (including atomics) from D-cache requests. Cacheable D-cache loads only.
EC_ic_miss	[PICU 00.1110] Counts L2-cache read misses from I-cache requests. The counter counts all I-cache misses including those for instructions from the mis-speculated execution path. Cacheable requests only.

P.5.6 Separating D-cache Stall Cycle Counts

The D-Cache stall cycle counts can be measured separately for L2-cache hits and misses by using the *Re_DC_missovhd* counter. The *Re_DC_missovhd* stall cycle counter is used with the recirculation and cache access events to separately calculate the D-cache loads that hit and miss the L2-cache. TABLE P-14 describes the *Re_DC_missovhd* stall cycle counter processor compatibility.

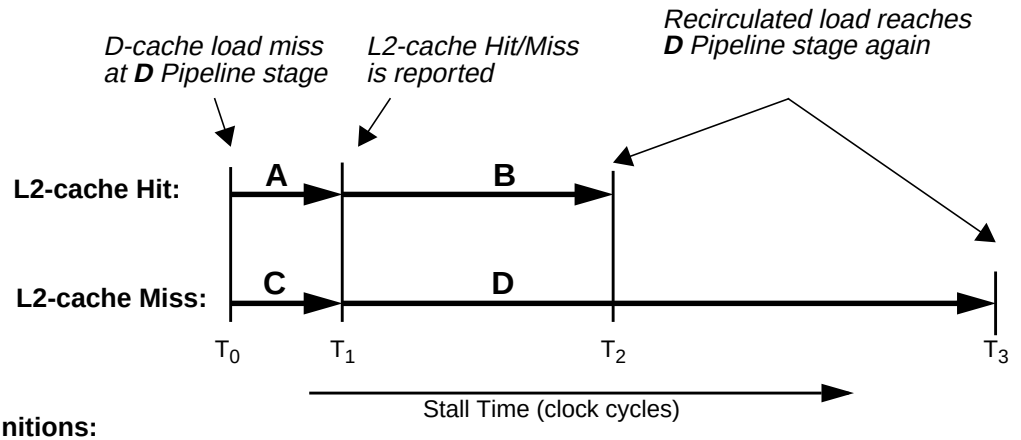
TABLE P-14 *Re_DC_missovhd* Stall Cycle Counter Processor Compatibility

Function	Description
Miss Overhead Cycle Monitor	The Re_DC_missovhd cycle stall counter is defined in TABLE P-8 and in the equations.

Synthesizing Individual Hit and Miss Stall Times

To explain the synthesis for L2-cache hit and miss stall times separately, consider the four stall regions A, B, C, and D shown in FIGURE P-5 and the definitions and calculations that follow.

D-cache misses to L2-cache



Definitions:

Re_DC_missovhd (stall cycles) = (A + C) stall cycles

Re_EC_miss (stall cycles) = (D) stall cycles

Re_DC_miss (stall cycles) = (A + B + C + D) stall cycles

Fraction of D-cache misses that miss L2-cache $= \frac{\text{miss L2}}{\text{miss D-cache}} = \frac{\textit{EC\_rd\_miss}\ \text{(events)}}{\textit{DC\_rd\_miss}\ \text{(events)}} =$ Miss L2 Ratio

Synthesized Stall Cycle Counts:

(C) Stall Cycles = *Re_DC_missovhd* * Miss L2 Ratio

L2-cache Miss Stall Cycles = (C + D) = (*C) + Re_EC_miss*

L2-cache Hit Stall Cycles = (A + B) = *Re_DC_miss* - (C + D)

FIGURE P-5 D-Cache Load Miss Stall Regions

P.6 Memory Controller Counters

This section describes the memory controller counters in the UltraSPARC IV processor.

P.6.1 Memory Controller Read Request Events

If the snoop indicates that the line exists in some other device, the memory read request is cancelled. The cancelled read requests are not counted. TABLE P-15 describes the counters for memory controller read events.

TABLE P-15 Counters for Memory Controller Read Events

Counter	Description
MC_reads_0	[PICL 10.0000] Counts read requests completed to memory bank 0.
MC_reads_1	[PICL 10.0001] Counts read requests completed to memory bank 1.
MC_reads_2	[PICL 10.0010] Counts read requests completed to memory bank 2.
MC_reads_3	[PICL 10.0011] Counts read requests completed to memory bank 3.

P.6.1.1 Memory Controller Write Request Events

TABLE P-16 describes the counters for memory controller write events.

TABLE P-16 Counters for Memory Controller Write Events

Counter	Description
MC_writes_0	[PICU 10.0000] Counts write requests completed to memory bank 0.
MC_writes_1	[PICU 10.0001] Counts write requests completed to memory bank 1.
MC_writes_2	[PICU 10.0010] Counts write requests completed to memory bank 2.
MC_writes_3	[PICU 10.0011] Counts write requests completed to memory bank 3.

P.6.1.2 Memory Request Stall Cycles

The stall cycles may be generated due to bus contention, a bank being busy, data availability for a write, etc. TABLE P-17 describes the counters for memory stall cycles.

TABLE P-17 Counters for Memory Controller Stall Cycles

Counter	Description
MC_stalls_0	[PICL 10.0100] Counts clock cycles that requests were stalled in the MCU queues because bank 0 was busy with a previous request.
MC_stalls_1	[PICU 10.0100] Counts clock cycles that requests were stalled in the MCU queues because bank 1 was busy with a previous request.

TABLE P-17 Counters for Memory Controller Stall Cycles *(Continued)*

Counter	Description
MC_stalls_2	[PICL 10.0101] Counts clock cycles that requests were stalled in the MCU queues because bank 2 was busy with a previous request.
MC_stalls_3	[PICU 10.0101] Counts clock cycles that requests were stalled in the MCU queues because bank 3 was busy with a previous request.

P.7 Data Locality Counters for Scalable Shared Memory Systems

There are four data locality performance event counters in the UltraSPARC IV processor. These event counters are provided to improve the ability to monitor and exploit performance in Scalable Shared Memory (SSM) systems where there are multiprocessor system clusters using Shared Memory Protocol (SMP) that are tied to other clusters using fabric interconnect utilizing the SSM architecture. TABLE P-18 describes the counters for data locality events.

TABLE P-18 Counters for Data Locality Events

Counter	Description
EC_miss_local	[PICL 01.1010] Counts any transaction to an *LPA* for which the processor issues an RTS/RTO/RS transaction.
EC_miss_mtag_remote	[PICL 01.1011] and [PICU 10.1000] Counts any transaction to an LPA address in which the processor is required to generate a retry transaction.
EC_miss_remote	[PICU 10.1001] Counts the events triggered whenever the processor generates a remote (R_*) transaction and the address is to a non-LPA portion (remote) of the physical address space, or an R_WS transaction due to block store (BST)/block store commit (BSTC) to any address space (LPA or non-LPA), or an R_RTO due to Store/Swap request on O_s state to LPA space.
EC_wb_remote	[PICL 01.1001] Counts the retry event when any victimization for which the processor generates an R_WB transaction to *non-LPA* address region. In practice, these are all NUMA cases, since for Coherence Memory Replication (CMR or COMA), the protocol insures that the processor only generates WB transactions.

SSM Systems

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

The data locality event counters are only valid for Shared System Memory architectures in SSM mode.

P.7.1 Event Tree

The event and cycle stall counters are illustrated in FIGURE P-6. The diagram includes actual counters and synthesized counts.

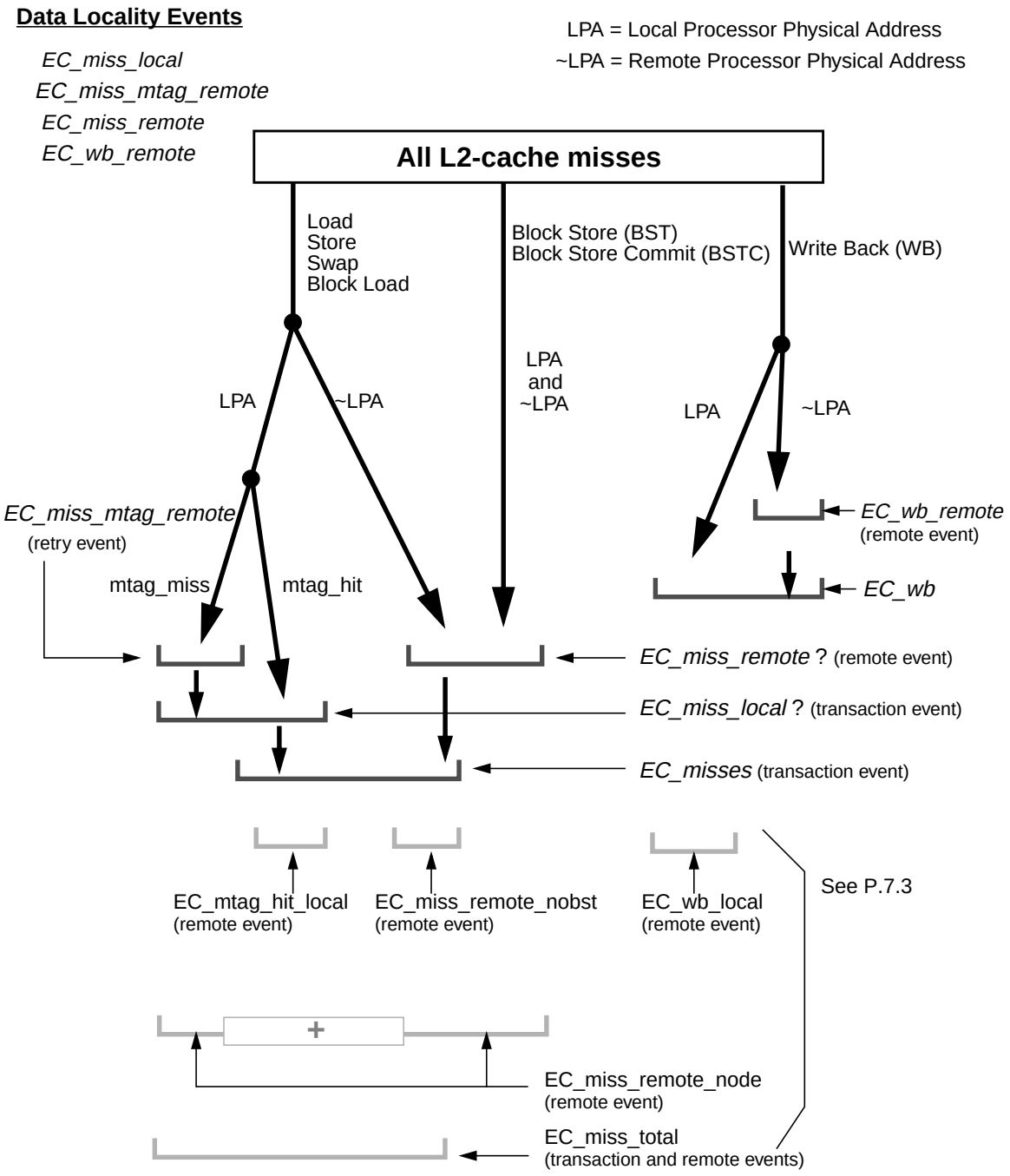


FIGURE P-6 Data Locality Event Tree for L2-Cache Misses

P.7.2 Data Locality Event Matrix

TABLE P-19 shows the data locality event matrix.

TABLE P-19 Data Locality Events

MODE	Combined State	Processor action					
		Load	Store/ Swap	Block Load	Block Store	Block Store with Commit	Write Back
LPA	I	Miss: RTS	Miss: RTO	Miss: RS	Miss: R_WS	Miss: R_WS	None
	E	Hit	Hit: E->M	Hit	Hit: E->M		
	S		MTag miss: RTO		Miss: R_WS		
	O						WB
	Os		MTag miss: R_RTO				WB
	M	Hit					WB
LPA Retried	I	MTag miss: R_RTS	MTag miss: R_RTO	MTag miss: R_RS	Invalid		None
	E	Invalid					
	S	Invalid	MTag miss: R_RTO	Invalid			None
	O						
	Os						
	M	Invalid					
~LPA	I	Miss: R_RTS	Miss: R_RTO	Miss: R_RS	Miss: R_WS	Miss: R_WS	None
	E	Hit	Hit: E->M	Hit	Hit: E->M		
	S	Hit	MTag miss: R_RTO	Hit	Miss: R_WS		
	O						Miss: R_WB
	Os						
	M	Hit					

LPA Retried Events

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

P.7.3 Synthesized Data Locality Events

The Data Locality event assignments allow the software to create synthetic events based on arithmetic combinations of events assigned to PICL and PICU, as shown in TABLE P-20.

TABLE P-20 Synthesized Data Locality Events

Synthesized Event		PICL	PICU
EC_wb_local	=	**+** EC_wb	– EC_wb_remote
EC_miss_remote_nobst (remote misses excluding BST/BSTC)	=	**+** EC_misses	– EC_miss_local
EC_miss_total (L2-cache misses + retries)	=	**+** EC_misses	+ EC_miss_mtag_remote
EC_miss_remote_node (CMR, NUMA)	=	**+** EC_miss_remote	**+** EC_miss_mtag_remote
EC_mtag_hit_local	=	– EC_miss_mtag_remote	**+** EC_miss_local

EC_miss_total

EC_miss_total counts all EC misses, which is EC_misses plus all retries. Retry means you have two transactions for each miss (first it misses MTag and then reissued).

P.8 Miscellaneous Counters

P.8.1 System Interface Events and Clock Cycles

System interface statistics are collected through the counters listed in TABLE P-21.

TABLE P-21 Counters for System Interface Statistics

Counter	Description
SI_snoop	[PICL 01.0001] Counts snoops from remote processor(s) including RTS, RTSR, RTO, RTOR, RS, RSR, RTSM, and WS.
SI_ciq_flow	[PICL 01.0010] Counts system clock cycles when the flow control (PauseOut) signal is asserted.
SI_owned	[PICL 010011] Counts events where owned_in is asserted on bus requests from the local processor.

P.8.2 Software Events

Software statistics are collected through the counters listed in TABLE P-22.

TABLE P-22 Counters for Software Statistics

Counter	Description
SW_count0	[PICL 01.0100] Counts software-generated occurrences of sethi %hi(0xfc000),%g0 instruction.
SW_count1	[PICU 01.1100] Counts software-generated occurrences of sethi %hi(0xfc000), %g0 instruction.

Note – Both counters measure the same event; thus, the count can be programmed to be read from either the PICL or the PICU register.

P.8.3 Floating-Point Operation Events

Floating-point operation statistics are collected through the counters listed in TABLE P-23.

TABLE P-23 Counters for Floating-Point Operation Statistics

Event Counter	Description
FA_pipe_completion	[PICL 01.1000] Counts instructions that complete execution on the Floating-point / Graphics ALU pipelines.
FM_pipe_completion	[PICL 10.0111] Counts instructions that complete execution on the Floating-point / Graphics Multiply pipelines.

P.9 PCR.SL and PCR.SU Encodings

TABLE P-24 lists PCR.SL and PCR.SL selection bit field encoding. Shaded blocks show SL and SU field duplications with light shading.

TABLE P-24 PIC.SL and PIC.SU Selection Bit Field Encoding

PCR.SL and PCR.SU Encodings	PICL Event Selection	PICU Event Selection
00.0000	Cycle_cnt	Cycle_cnt
00.0001	Instr_cnt	Instr_cnt
00.0010	Dispatch0_IC_miss	Dispatch0_mispred
00.0011	Dispatch0_br_target	IC_miss_cancelled
00.0100	Dispatch0_2nd_br	Re_DC_missovhd
00.0101	Rstall_storeQ	Re_FPU_bypass
00.0110	Rstall_IU_use	Re_DC_miss
00.0111	*Reserved*	Re_EC_miss
00.1000	IC_ref	IC_miss
00.1001	DC_rd	DC_rd_miss
00.1010	DC_wr	DC_wr_miss
00.1011	*Reserved*	Rstall_FP_use
00.1100	EC_ref	EC_misses
00.1101	EC_write_hit_RTO	EC_wb
00.1110	EC_snoop_inv	EC_snoop_cb
00.1111	EC_rd_miss	EC_ic_miss
01.0000	PC_port0_rd	Re_PC_miss
01.0001	SI_snoop	ITLB_miss

TABLE P-24 PIC.SL and PIC.SU Selection Bit Field Encoding *(Continued)*

PCR.SL and PCR.SU Encodings	PICL Event Selection	PICU Event Selection
01.0010	SI_ciq_flow	DTLB_miss
01.0011	SI_owned	WC_miss
01.0100	SW_count0	WC_snoop_cb
01.0101	IU_Stat_Br_miss_taken	WC_scrubbed
01.0110	IU_Stat_Br_count_taken	WC_wb_wo_read
01.0111	Dispatch_rs_mispred	*Reserved*
01.1000	FA_pipe_completion	PC_soft_hit
01.1001	EC_wb_remote	PC_snoop_inv
01.1010	EC_miss_local	PC_hard_hit
01.1011	EC_miss_mtag_remote	PC_port1_rd
01.1100	*Reserved*	SW_count1
01.1101	*Reserved*	IU_Stat_Br_miss_untaken
01.1110	*Reserved*	IU_Stat_Br_count_untaken
01.1111	*Reserved*	PC_MS_miss
10.0000	MC_reads_0	MC_writes_0
10.0001	MC_reads_1	MC_writes_1
10.0010	MC_reads_2	MC_writes_2
10.0011	MC_reads_3	MC_writes_3
10.0100	MC_stalls_0	MC_stalls_1
10.0101	MC_stalls_2	MC_stalls_3
10.0110	*Reserved*	Re_RAW_miss
10.0111	*Reserved*	FM_pipe_completion
10.1000	*Reserved*	EC_miss_mtag_remote
10.1001	*Reserved*	EC_miss_remote
10.1010 - 11.1111	*Reserved*	*Reserved*

Bibliography

General References

Boney, Joel. “SPARC Version 9 Points the Way to the Next Generation RISC,” *SunWorld*, October 1992, pp. 100-105.

Cohen, D., “On Holy Wars and a Plea for Peace.” *Computer* 14:10, October 1981, pp. 48-54.

Comer, Douglas. “The Ubiquitous B-Tree.” *ACM Computing Surveys*, Vol. 11, No. 2, June 1979.

Implementation Characteristics of Current SPARC V9-based Products, Revision 9.x, SPARC International, Inc.

Knuth, Donald. *The Art of Computer Programming, Volume 3, Searching and Sorting. Addison-Wesley, 1974.*

Weaver, David L., editor. *The SPARC Architecture Manual, Version 8*, Prentice-Hall, Inc., 1992.

Weaver, David L., and Tom Germond, eds. *The SPARC Architecture Manual-Version 9*, Prentice-Hall, Inc., 1994.

Index

B

C

D

E

F

G

H

I

L

M

N

P

R

S

T

V

W

X