ORACLE

SPARC T4™ Supplement

to *Oracle SPARC Architecture 2011*

Draft D0.4, 14 Feb 2012

Privilege Levels: *Hyperprivileged, Privileged, and Nonprivileged*

Distribution: Public

Part No: 950-5766-00
Revision: Draft D0.4, 14 Feb 2012

Oracle Corporation
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 650-960-1300

CHAPTER 10

Performance Instrumentation

10.1 Introduction

As in previous UltraSPARC CMT processors such as UltraSPARC T1, UltraSPARC T2, UltraSPARC T2+, and SPARC T3, SPARC T4 supports monitoring processor performance by virtue of a set of performance counters. SPARC T4 expands on the capabilities of previous SPARC CMT processors by adding more counters per virtual processor and by being able to measure additional processor and pipeline events. Significant differences from SPARC T3 are as follows:

1. SPARC T4 supports 4 counters (PICs) per virtual processor instead of two.
2. Each PIC is controlled via a dedicated PCR. Each PCR controls only one PIC.
3. The format of the PCR has changed significantly.
4. Access to the PCRs is via hyperprivileged ASIs only, instead of ASRs. The hypervisor can then permit privileged and user access only to the PICs via PCR.picnht and PCR.picnpt, respectively. The PCRs can thus be allocated to hypervisor, supervisor, or user code in any combination. This also enables virtualization of performance counter and measurement infrastructure to ease future development as processor architecture evolves.
5. Access to the PICs is via non-privileged ASIs only, instead of ASRs. Access is only granted based upon the settings of PCR.picnht and PCR.picnpt as described above.
6. The *pic_overflow* trap no longer exists. Instead, a PIC which overflows due to a precise performance event generates a *precise_performance_event* trap, and a PIC which overflows due to an asynchronous performance event generate a *disrupting_performance_event* trap. Neither trap sets SOFTINT{15}.
7. Precise performance counter overflows have no "skid".

10.2 SPARC Performance Control Registers

Each virtual processor has four hyperprivileged, read/write Performance Control registers: PCR0, PCR1, PCR2, and PCR3. Each PCR controls its corresponding PIC: PCR0 controls PIC0, PCR1 controls PIC1, PCR2 controls PIC2, and PCR3 controls PIC3. Each Performance Control register contains ten fields: ntc, picnht, picnpt, sl, mask, ht, ut, st, toe, and ov. All bits except ntc and ov are always updated on a Performance Control register write. ov is a state bit associated with PIC overflow traps and is provided to allow software to determine whether a PIC counter has overflowed. ntc is also a state bit associated with PIC overflow traps that allows software to handle a special case on a *precise_performance_event* trap: TPC and TNPC point to the instruction which caused the overflow, but hardware already executed the instruction at TPC. In this case software must execute a DONE

instead of a RETRY. ntc and ov can be reset by software but can never be written to 1. sl controls which events are counted in a PIC. mask is used in conjunction with sl to determine which set of subevents are counted in a PIC. toe controls whether a trap is generated when the PIC counter overflows. ut controls whether user-level events are counted. st controls whether supervisor-level events are counted. ht controls whether hypervisor level events are counted.The format of this register is shown in TABLE 10-1. Note that changing a field in the PCR does not directly affect a PIC value. To reliably change the events being monitored, software should perform the following sequence:

1. Disable counting by writing zeroes to PCR.sl and clearing PCR.ut, PCR.ht, and PCR.st.
2. Reset the PIC.
3. Enable the new event via writing a non-zero value to PCR.sl and setting PCR.ut, PCR.ht, or PCR.st, as appropriate.

Programming Note There is no explicit means of synchronizing multiple counters to start counting at the same cycle, as each counter must be separately enabled by writing to its PCR. However, counters which are not counting events in hyperprivileged mode (that is, PCR.ht is set to 0) will not begin counting until the virtual processor has transitioned out of hyperprivileged mode. Thus, software can use the transition from hyperprivileged mode as a barrier for counting user and privileged events.

TABLE 10-1 Performance Control Registers – PCR0-3 (ASI 64_{16}, VA 00_{16}, 08_{16}, 10_{16}, 18_{16})

Bit	Field	Initial Value	R/W	Description
63:19	—	0	RO	*Reserved*
18	ntc	0	RW	Set to 1 when PIC wraps from 2^{32} –1 to 0 on a next-to-commit (ntc) instruction[1]. Once set, ntc remains set until reset by software. Hardware sets ntc whenever it sets ov on a next-to-commit instruction.
17	picnht	0	RW	PIC non-hyperprivileged trap. Privileged software can access the PIC only if picnht = 0, otherwise a *privileged_action* trap occurs. Non-privileged software can access PIC only when picnht = 0 and picnpt = 0, otherwise a *privileged_action* trap occurs.
16	picnpt	0	RW	PIC non-privileged trap. Non-privileged software can access PIC only when picnht = 0 and picnpt = 0, otherwise a *privileged_action* trap occurs.
15:11	sl	0	RW	Selects one of 32 events to be counted for PIC as per the following table.
10:5	mask	0	RW	Mask event for PIC as listed in TABLE 10-2.
4	ht	0	RW	If ht = 1, count events in hyperprivileged mode; otherwise, ignore hyperprivileged mode events.
3	st	0	RW	If st = 1, count events in privileged mode; otherwise, ignore privileged mode events.

TABLE 10-1 Performance Control Registers – PCR0-3 (ASI 64_{16}, VA 00_{16}, 08_{16}, 10_{16}, 18_{16})

Bit	Field	Initial Value	R/W	Description
2	ut	0	RW	If ut = 1, count events in user mode; otherwise, ignore user mode events.
1	toe	0	RW	Trap-on-Event: This field controls whether a precise trap (*precise_performance_event*) or disrupting trap (*disrupting_performance_event*) to hyperprivileged software occurs if the corresponding PIC counter overflows. Hardware **AND**s the value of toe with ov to produce a trap. Events in certain event groups (those marked as Precise in TABLE 10-2) generate a precise *precise_performance_event* trap, assuming that PCR.toe = 1 and PCR.ht = 0 — TPC will contain the address of an instruction that generated the counter overflow event[2]. Events in other event groups are not directly related to the instruction stream and an overflow for one of the asynchronous events generates a *disrupting_performance_event* trap; therefore, the TPC may be some number of instructions later than when the overflow event occurred.
0	ov	0	RW	Set to 1 when PIC wraps from 2^{32} –1 to 0. Once set, ov remains set until reset by software.

1. The following instructions are next-to-commit instructions: MD5, SHA1, SHA256, SHA512, MPMUL, MONTMUL, MONTSQR, loads and stores to I/O space, CAS{X}A, LDSTUB, SWAP, WRHPR, WRASR, WRPR, RDHPR, RDPR, RDASR instructions, and any non-translating load or store alternate instruction as defined in Table 9-3, "SPARC T4 ASI Usage," on page 76. When hardware takes a *precise_performance_event* trap on a next-to-commit instruction, the instruction has already been executed. Therefore, trap handler software should execute a DONE instruction; it must not execute a RETRY instruction. Software can examine the ntc bit to determine whether to execute a DONE or a RETRY instruction.
2. A *precise_performance_event* trap can not be generated in hyperprivileged mode (HPSTATE.hpriv=1) or when monitoring hyperprivileged events (PCR.ht = 1). See section 10.3 for more detail.

[

Programming Note Eacn PCR.ov bit is independent of other PCR.ov bits. Specifically, multiple PICs may be set up to count events and trap on overflows. Hardware does not prioritize the traps across the PCRs/PICs for the purposes of setting the ov bit. For example, PIC0 may overflow due to an ITLB miss event and PIC1 may simultaneously overflow due to a data cache miss event. Both PCR0.ov and PCR1.ov will be set, despite the fact that the trap priority of the ITLB miss event (2.06.1) is higher than the data cache miss event (12.09.1).

Programming Note It is possible that a write to a PCR causes the associated PIC to overflow when counting hyperprivileged events: for example, this occurs when the PCR is counting instructions and the PIC is set to FFFF_FFFF$_{16}$ just prior to the write. In this case, if the PCR write sets PCR.ov to zero, hardware re-writes PCR.ov to one. Software should take care to handle this case, for example by disabling counting with the first PCR write, then writing the PCR a second time to ensure PCR.ov remains set to 0.

TABLE 10-2 describes the settings of the sl field. Most sl fields have a mask associated with them. Setting multiple mask bits at the same time can lead to multiple events being counted as one event. Some sl groups do not use all of the mask bits; setting unused mask bits has no effect. More details are described in TABLE 10-2.

TABLE 10-2 sl Field Settings *(1 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
0	—	Disabled	—	—	Performance counting is disabled; no events are counted, and performance counters are power-managed off. mask bits are ignored.
1	01_{16}	Sel-pipe-drain-cycles	No	Yes	Cycles a strand waits at Select with correct path instructions after a mispredicted branch for the branch to be committed.[10] This is one component of Sel-0-wait cycles below.
	02_{16}	Sel-0-wait			Cycles a strand waits at Select for some condition to be resolved; includes Sel-pipe-drain-cycles above, but does not include Sel-0-ready below and cycles waiting due to ROB/PQ/SB/LB tag stalls.
	04_{16}	Sel-0-ready			Cycles a strand was ready to be selected but another strand was selected instead. This can be used to estimate pipeline oversubscription.
	08_{16}	Sel-1			Cycles that only one instruction or micro-op was selected
	10_{16}	Sel-2			Cycles that 2 instructions or micro-ops were selected
	Any other value	—			Count cycles as determined by the setting of individual mask bits (e.g., a mask value of $1F_{16}$ counts all select events - pipe drain cycles, sel-0-wait, sel-0-ready, sel-1, and sel-2). Events sel-0-wait, sel-0-ready, sel-1, and sel-2 form a disjoint set of events whose union encompasses all cycles, exluding cycles where the thread was ready but was stalled due to tag stalls. However, the only useful mask settings with multiple bits set are 6_{16} and 18_{16}, since the sel-0-wait and sel-0-ready events are not cycle-by-cycle mutually exclusive with sel-1 and sel-2 due to implementation issues. mask[5] is ignored.
2	01_{16}	Pick-0	No	Yes	Cycles when no instruction is picked
	02_{16}	Pick-1			Cycles when 1 instruction or micro-op is picked
	04_{16}	Pick-2			Cycles when 2 instructions or micro-ops are picked
	08_{16}	Pick-3			Cycles when 3 instructions or micro-ops are picked
	$0E_{16}$	Any pick			Cycles when at least 1 instruction or micro-op is picked
	Any other value	—			Count cycles when instructions are picked identified by a 1 in the corresponding mask bit; mask[5:4] is ignored. May provide meaningless data.
3	01_{16}	Branches	Yes	Yes	All Bicc, BPcc, BPr, CALL, CBcc, FBFcc, FBPFcc, JMPL, JPRIV, and RETURN instructions are counted. Tcc, DONE, RETRY, and SIR are not counted as branches.
	02_{16}	FGU + crypto arithmetic instructions[5]			See Footnote 5 below for a list of the instructions counted in this group.
	04_{16}	Load instructions[6]			See Footnote 6 below for a list of instructions counted in this group.
	08_{16}	Store instructions[7]			See Footnote 7 below for a list of instructions counted in this group.
	10_{16}	SPR ring ops[8]			See Footnote 8 below for a list of instructions counted in this group.
	20_{16}	Other instructions[9]			See Footnote 9 below for a list of instructions counted in this group.
	Any other value	Any subset of instructions			Count instruction types identified by a 1 in the corresponding mask register bit; e.g., $3F_{16}$ counts all instructions.

TABLE 10-2 sl Field Settings *(2 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
4	01_{16}	Branches	Yes	Yes	Bicc, BPcc, BPr, CALL, CBcc, FBfcc, FBPfcc, JMPL, JPRIV, RETURN. Tcc, DONE, RETRY, and SIR are not counted as branches.
	02_{16}	Taken branches			Bicc, BPcc, BPr, CALL, CBcc, FBfcc, FBPfcc, JMPL, JPRIV, RETURN.
	04_{16}	`sethi %hi(fc000`$_{16}$`), %g0`			Software count instructions.
	08_{16}	atomics			Atomics are LDSTUB/A, CASA/XA, and SWAP/A
	10_{16}	Software prefetch			PREFETCH and PREFETCHA instructions
	20_{16}	Block loads and stores			Counts LDDFA_0X16, LDDFA_0X1E, LDDFA_0XF0, LDDFA_0XF8, STDFA_0X17, STDFA_0XE0, STDFA_0XF0, STDFA_0XF8, STDFA_0XF9 where the _0XZZ suffix denotes an ASI value of ZZ_{16}.
	Any other value	Any subset of instructions			Count instruction types identified by a 1 in the corresponding mask register bit
5	01_{16}	Icache miss and either L2 cache, local L2 cache-to-cache, or local L3 cache hit	Yes	Yes	**Note:** This counts only primary instruction cache misses.[11]
	02_{16}	Icache miss and local memory hit (implies L2 and local L3 miss)			
	04_{16}	Icache miss and remote L3 cache or remote memory hit			
	03_{16}, 05_{16}, 06_{16},	Different combinations of Icache misses			
	07_{16}	Icache miss			Instruction cache misses of all types
	08_{16}	BTC miss			Branch Target Cache miss
	10_{16}	ITLB misses			Successful and unsuccessful hardware tablewalks. Note: Instruction fetches resulting in ITTM, out-of-pipe ITTP, or out-of-pipe ITDP RAS errors are counted as ITLB misses. An out-of-pipe ITTP or ITDP RAS error occurs if there is an instruction cache miss on the instruction fetch.
	Any other value	Different combinations of events			There are three different types of unrelated events in this group: instruction cache misses, branch target cache misses, and ITLB misses, so setting multiple mask bits may be meaningless; mask bit 5 is ignored.
6	01_{16}	ITLB fill for 8KB page	No	Yes	Hardware tablewalk fills ITLB with translation for 8KB page
	02_{16}	ITLB fill for 64KB page			Hardware tablewalk fills ITLB with translation for 64KB page
	04_{16}	ITLB fill for 4MB page			Hardware tablewalk fills ITLB with translation for 4MB page
	08_{16}	ITLB fill for 256MB page			Hardware tablewalk fills ITLB with translation for 256MB page
	10_{16}	ITLB fill for 2GB page			Hardware tablewalk fills ITLB with translation for 2GB page
	20_{16}	ITLB fill trap			Hardware tablewalk unsuccessful. Note: The counter does not increment if a RAS error occurs in the RA2PAC or TSB configuration registers.
	Any other value	Different combinations of ITLB miss events			When set to $3F_{16}$ counts all ITLB misses asynchronously; sl=5, mask=10_{16} must be used to count ITLB misses synchronously.

TABLE 10-2 sl Field Settings *(3 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
7	01_{16}	Instruction cache microtag miss	No	Yes	
	02_{16}	Instruction cache microtag miss, ptag hit			
	04_{16}	Instruction cache microtag hit, ptag miss			
	08_{16}	Instruction cache microtag hit,ptag hit, way mismatch			
	05_{16}, 06_{16}, 09_{16}, $0A_{16}$, $0C_{16}$, $0D_{16}$, $0E_{16}$	Combinations of microtag hits and misses			mask[0] and mask[1] events are not mutually exclusive. mask bits [1], [2], and [3] are mutually exclusive, as are mask bits [0], [2], and [3]. mask[5:4] is ignored.
8	01_{16}	Fetch-0	No	Yes	Cycles when no valid instructions are fetched for this strand. This counter increments when no instruction fetch is performed or a fetch is performed but due to a microtag miss, cache miss, or TLB miss, no instructions were written to the instruction buffer.
	02_{16}	Fetch-0-all		No	Cycles when no valid instructions are fetched for any strand[4]. This event includes cycles an instruction fetch was attempted for any strand but no valid instructions were written to any strand's instruction buffer as noted above for fetch-0 events.
	04_{16}	Instruction buffer full cycles		Yes	Cycles when the strand's instruction buffer is full, preventing instruction fetch for the strand
	08_{16}	BTC target incorrect			Number of times the target predicted from the branch target cache is incorrect
	Any other value	Not useful			Since this group contains independent and unrelated events, setting multiple mask bits is not useful. mask[5:4] is ignored.
9	01_{16}	PQ tag wait cycles	No	No	Cycles Rename waits for a Pick Queue tag[4].
	02_{16}	ROB tag wait cycles			Cycles Select waitsfor a ROB tag[4].
	04_{16}	LB tag wait cycles			Cycles Select waits for a Load Buffer tag[4].
	08_{16}	SB tag wait cycles			Cycles Select waits for a Store Buffer tag[4].
	06_{16}, $0A_{16}$, $0C_{16}$, $0E_{16}$	Select tag wait cycles			Count cycles for instructions that wait at Select for a ROB, LB, or SB tag. This is also a subset of Sel-0-wait[4]. Although these events can overlap, the counter is only incremented once for each cycle that any of the waits occur.
	10_{16}	DTLB miss tag wait cycles			Cycles load and store instructions are stalled at Pick waiting for a DTLB miss tag[4].
	Any other value	Not useful			Since this group contains independent and unrelated events, other mask bit settings are not useful. mask[5] is ignored.

TABLE 10-2 sl Field Settings *(4 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
10	01_{16}	ITLB hardware tablewalk references which hit L2	No	Yes	Counts each HWTW access due to an ITLB miss that hits in the L2 cache
	02_{16}	ITLB hardware tablewalk references which hit L3			Counts each HWTW access due to an ITLB miss that misses in the L2 cache and hits in the L3 cache or hits in another L2 cache and results in an L2 cache to L2 cache transfer.
	04_{16}	ITLB hardware tablewalk references which miss L3			Counts each HWTW access due to an ITLB miss that misses in the L2 and L3 caches
	07_{16}	ITLB hardware tablewalk references			Count all HWTW accesses due to ITLB misses
	08_{16}	DTLB hardware tablewalk references which hit L2			Counts each HWTW access due to a DTLB miss that hits in the L2 cache
	10_{16}	DTLB hardware tablewalk references which hit L3			Counts each HWTW access due to a DTLB miss that misses in the L2 cache and hits in the L3 cache or hits in another L2 cache and results in an L2 cache to L2 cache transfer.
	20_{16}	DTLB hardware tablewalk references which miss L3			Counts each HWTW access due to a DTLB miss that misses in the L2 and L3 caches
	38_{16}	DTLB hardware tablewalk references			Count all HWTW accesses due to DTLB misses
	Any other value	Combinations of ITLB and DTLB hardware tablewalk references			
11[1]	01_{16}	Instruction cache miss and local L2 cache, local L2 cache to cache, or local L3 hit	No	Yes	
	02_{16}	Instruction cache miss and local memory hit, remote L3 hit, or remote memory hit			
	03_{16}	Any instruction cache miss			
	Any other value	Ignored			
12-15	—	—	—	—	*Reserved*
16	01_{16}	L1 data cache miss and local L2 cache, local L2 cache to cache, or local L3 cache hit	Yes	Yes	Counts primary and duplicate misses. Does not count block loads, twin loads, prefetches, or atomics.
	02_{16}	L1 data cache miss + local memory hit			
	04_{16}	L1 data cache miss + remote memory or L3 hit			
	07_{16}	L1 data cache miss			
	Any other value	Other combinations of L1 dcache misses			mask[5:3] is ignored.

TABLE 10-2 sl Field Settings *(5 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
17	01_{16}	DTLB fill for 8KB page	No	Yes	Hardware tablewalk fills DTLB with translation for 8KB page
	02_{16}	DTLB fill for 64KB page			Hardware tablewalk fills DTLB with translation for 64KB page
	04_{16}	DTLB fill for 4MB page			Hardware tablewalk fills DTLB with translation for 4MB page
	08_{16}	DTLB fill for 256MB page			Hardware tablewalk fills DTLB with translation for 256MB page
	10_{16}	DTLB fill for 2GB page			Hardware tablewalk fills DTLB with translation for 2GB page
	20_{16}	DTLB fill trap			Hardware tablewalk unsuccessful. Note: The counter does not increment if a RAS error occurs in the RA2PAC or TSB configuration registers.
	Any other value	Different combinations of DTLB miss events			When set to $3F_{16}$ counts all DTLB misses asynchronously. Unlike the ITLB, there is no way to count DTLB misses synchronously.
18[2]	01_{16}	Dropped L1 data cache hardware prefetch due to data cache hit	No	No	
	02_{16}	Dropped software prefetch at data cache due to data cache hit		Yes	
	04_{16}	Dropped software prefetch at data cache due to miss buffer being full		Yes	Note: Software prefetches are not dropped when only one thread is running, so this count will always read 0 if only one thread is running (unparked and unhalted).
	06_{16}	Dropped software prefetches at data cache		Yes	
	Any other value	Combinations of dropped hardware and software prefetches			mask[5:3] is ignored.
19	01_{16}	Full RAW hit in store buffer	No	Yes	
	02_{16}	Partial RAW hit in store buffer			
	03_{16}	RAW hit in store buffer			
	04_{16}	Full RAW hit in store queue			
	08_{16}	Partial RAW hit in store queue			
	$0C_{16}$	Full or partial RAW hit in store queue			
	Any other value	Combinations of RAW hits			Note: Store buffer RAW hits supercede store queue RAW hits. mask[5:4] is ignored.

TABLE 10-2 sl Field Settings *(6 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
20	01_{16}	Instruction cache eviction invalidations	No	No	
	02_{16}	Instruction cache snoop invalidations			
	03_{16}	Instruction cache invalidations			
	04_{16}	Data cache eviction invalidations			
	08_{16}	Data cache snoop invalidations			
	$0C_{16}$	Data cache invalidations			
	$0A_{16}$	L1 cache snoop invalidations			Both instruction and data cache snoop invalidations
	$0F_{16}$	Any L1 cache invalidation			Both instruction and data cache invalidations.
	10_{16}	Store queue tag wait cycles			See footnote [4].
	Any other value				Not meaningful. mask[5:3] is ignored.
21	01_{16}	Data prefetch hits in L2	No	Yes	Note: this group only counts prefetches that were destined for the L2 (e.g., it excludes prefetch instructions with function codes equal to 0_{16} or 14_{16}).
	02_{16}	Data prefetches dropped by L2			
	04_{16}	Data prefetch hits in L3			
	08_{16}	Data prefetch hits to local memory			
	10_{16}	Data prefetch hits to remote memory			
	20_{16}	Data prefetches dropped by L3			
	Any other value	Different combinations of data prefetch events			
22	01_{16}	Store hits in L2	No	Yes	
	02_{16}	Store hits in L3			Stores that miss in L2 but hit in L3
	04_{16}	Store L2 local C2C			Stores that transfer a line from another core's L2 cache on the same die
	08_{16}	Store L2 remote C2C			Stores that transfer a line from another core's L2 cache on a different die
	10_{16}	Stores to local memory			
	20_{16}	Stores to remote memory			
	Any other value	Different store prefetches			

TABLE 10-2 sl Field Settings *(7 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
23[3]	01_{16}	L1 data cache miss and local L2 cache, local L2 cache to cache, or local L3 cache hit	No	Yes	Counts primary and duplicate misses. Does not count block loads, twin loads, prefetches, or atomics.
	02_{16}	L1 data cache miss + local memory hit			
	04_{16}	L1 data cache miss + remote memory or L3 hit			
	07_{16}	L1 data cache miss			
	Any other value	Other combinations of L1 dcache misses			mask[5:3] is ignored.
24	01_{16}	Clean L2 evictions	No	No	
	02_{16}	Dirty L2 evictions			
	04_{16}	L2 fill buffer full			
	08_{16}	L2 writeback buffer full			
	10_{16}	L2 miss buffer full			
	20_{16}	L2 pipeline stalls			
	Any other value	Different combinations of L2 events			Not all events are mutually exclusive.
25	01_{16}	Branch direction mispredicts	Yes	Yes	
	02_{16}	Branch target mispredict due to far table or return stack			Includes both far table mispredicts and return stack mispredicts. To count only far table mispredicts, use two counters with one set to count this event and the other set to count return stack mispredicts (mask[5:0]==08_{16}) and subtract them.
	04_{16}	Branch target mispredict due to indirect table			
	08_{16}	Branch target mispredict due to return stack			
	$0E_{16}$	Branch target mispredict			
	$0F_{16}$	Branch mispredict			
	Any other value				mask bits 3, 2, and 1 can be combined to detect multiple combinations of branch target mispredicts. mask[5:4] is ignored.
26	Any value	Cycles in hyperprivileged mode, privileged mode, or user mode	No	Yes	Counts cycles in each mode as follows: 1. If PCRx.ht is set, count cycles in hyprivileged mode. 2. If PCRx.st is set, count cycles in privileged mode. 3. If PCRx.ut is set, count cycles in user mode. The mask field is ignored.
27		—			*Reserved*

TABLE 10-2 sl Field Settings *(8 of 8)*

sl[4:0]	mask[5:0]	Event	Precise Event	Virtual processor-specific	Description
28[4]	01_{16}	Commit-0	No	Yes	No micro-op[12] committed for this strand. Micro-ops that trap also occupy the commit pipeline; however, micro-ops that trap are not counted as having committed, and therefore count as a commit-0 event.
	02_{16} or 03_{16}	Commit-0-all		No	No micro-op committed for any strand[13].
	04_{16}	Commit-1		Yes	One micro-op committed.
	08_{16}	Commit-2		Yes	Two micro-ops committed (this increments the counter by one each time a pair of micro-ops is committed). If a counter overflow occurs the TPC is of the second micro-op of the pair.
	$0C_{16}$	Commit-1 or Commit-2		Yes	Increments by 1 when at least one micro-op commits.
	Any other value	Not useful			mask[5:4] is ignored; using mask[3:2] in conjunction with mask[1:0] produces meaningless results.
29-31		—			*Reserved*

1. sl 11 counts instruction cache misses that occur regardless of whether or not the instruction is committed or flushed, whereas sl 5 counts instruction cache misses only on instructions that are committed. These counts differ with sl 11 counts always being at least as high as sl 5. Since the core fetches instructions speculatively, an instruction cache miss may occur for an instruction that is later flushed due to an exception, a branch misprediction, or as a result of a thread hog policy flushing the thread.
2. For sl 18, hardware may drop prefetches due to temporal thread-specific miss buffer capacity restrictions. These dropped prefetches are not counted by any of the events in this or any other group.
3. sl 23 counts data cache miss events which occur on instructions regardless of whether they are committed or flushed, whereas sl 16 counts data cache miss events which occur on instructions that are committed. The counts differ with sl 23 counts always being at least as high as sl 16. Since the core issues loads speculatively, a data cache miss may occur for an instruction that is flushed due to an exception, a branch misprediction, or as a result of a thread hog policy flushing the thread.
4. See the Programming Note for sl 28 below.
5. The following instructions are categorized as FGU + crypto: AES_DROUND01, AES_DROUND01_LAST, AES_DROUND23, AES_DROUND23_LAST, AES_EROUND01_LAST, AES_EROUND23_LAST, AES_KEXPAND0, AES_KEXPAND1, AES_KEXPAND2, ALIGNADDRL, ARRAY16, ARRAY32, ARRAY8, BMASK, BSHUFFLE, CAMELLIA_F, CAMELLIA_FL, CAMELLIA_FLI, CMASK16, CMASK32, CMASK8, CRC32C, DES_IIP, DES_IP, DES_KEXPAND, DES_ROUND, EDGE16L, EDGE16LN, EDGE16N, EDGE32L, EDGE32LN, EDGE32N, EDGE8, EDGE8L, EDGE8LN, EDGE8N, FABSD, FABSS, FADDD, FADDS, FALIGNDATA, FANDNOT1S, FANDNOT2, FANDNOT2S, FANDS, FCHKSM16, FCMPD, FCMPED, FCMPEQ16, FCMPEQ32, FCMPES, FCMPGT16, FCMPGT32, FCMPLE16, FCMPLE32, FCMPNE16, FCMPNE32, FCMPS, FDIVD, FDIVS, FDTOI, FDTOS, FDTOX, FEXPAND, FHADDD, FHADDS, FHSUBD, FHSUBS, FITOD, FITOS, FLCMPD, FLCMPS, FMADDD, FMADDS, FMEAN16, FMOVDFN, FMOVDN, FMOVRDGEZ, FMOVRDGZ, FMOVRDLEZ, FMOVRDLZ, FMOVRDNZ, FMOVRDZ, FMOVRSGEZ, FMOVRSGZ, FMOVRSLEZ, FMOVRSLZ, FMOVRSNZ, FMOVRSZ, FMOVSFN, FMOVSN, FMSUBD, FMSUBS, FMUL8SUX16, FMUL8ULX16, FMUL8X16AL, FMUL8X16AU, FMULD8SUX16, FMULD8ULX16, FMULS, FNADDD, FNADDS, FNANDS, FNEGD, FNEGS, FNHADDD, FNHADDS, FNMADDD, FNMADDS, FNMSUBD, FNMSUBS, FNMULD, FNMULS, FNOR,FNORS, FNOT1, FNOT1S, FNOT2S, FNSMULD, FONE, FONES, FORNOT1, FORNOT1S, FORNOT2S, FORS, FPACK16, FPACK32, FPACKFIX, FPADD16, FPADD16S, FPADD32S, FPADD64, FPADDS16, FPADDS16S, FPADDS32, FPADDS32S, FPMADDXHI, FPMERGE, FPSUB16S, FPSUB32S, FPSUB64, FPSUBS16S, FPSUBS32S, FSLAS16, FSLAS32, FSLL16, FSLL32, FSMULD, FSQRTD, FSQRTS, FSRA16, FSRA32, FSRC1, FSRC1S, FSRC2, FSRC2S, FSRL16, FSRL32, FSTOD, FSTOI, FSTOX, FSUBD, FSUBS, FUCMPEQ8, FUCMPGT8, FUCMPLE8, FUCMPNE8, FXNORS, FXOR, FXORS, FXTOD, FZEROS, KASUMI_FI_FI, KASUMI_FI_XOR, KASUMI_FL_XOR, LZD, MD5, MONTSQR, MOVSTOSW, MOVSTOUW, MOVWTOS, MPMUL, MULSCC, PDISTN, POPC, SDIV, SDIVCC, SDIVX, SHA1, SHA256, SHA512, SMUL, SMULCC, UDIVCC, UDIVX, UMUL, UMULCC, UMULXHI, XMULX, and XMULXHI.

6. The following instructions are counted as loads, where the _0XZZ suffix denotes an alternate-form load with an ASI value of ZZ_{16}: LDD, LDDA_0X14, LDDA_0X15, LDDA_0X1C, LDDA_0X1D, LDDA_0X22, LDDA_0X23, LDDA_0X26, LDDA_0X27, LDDA_0X2A, LDDA_0X2B, LDDA_0X2E, LDDA_0X2F, LDDA_0X80, LDDA_0X82, LDDA_0X88, LDDA_0X8B, LDDA_0XE2, LDDA_0XE3, LDDA_0XEA, LDDA_0XEB, LDDF, LDDFA_0X16, LDDFA_0X1E, LDDFA_0X4, LDDFA_0X80, LDDFA_0X88, LDDFA_0X89, LDDFA_0X8A, LDDFA_0XC, LDDFA_0XD0, LDDFA_0XDA, LDDFA_0XF0, LDDFA_0XF8, LDF, LDFA_0X15, LDFA_0X1D, LDFA_0X4, LDFA_0X80, LDFA_0X81, LDFA_0X82, LDFA_0X83, LDFA_0X88, LDFA_0X89, LDFA_0X8B, LDFA_0XC, LDFSR, LDSB, LDSBA_0X14, LDSBA_0X15, LDSBA_0X1C, LDSBA_0X1D, LDSBA_0X31, LDSBA_0X4, LDSBA_0X80, LDSBA_0X81, LDSBA_0X88, LDSBA_0X8B, LDSBA_0XC, LDSH, LDSHA_0X14, LDSHA_0X1C, LDSHA_0X1D, LDSHA_0X36, LDSHA_0X38, LDSHA_0X39, LDSHA_0X3E, LDSHA_0X80, LDSHA_0X82, LDSHA_0X83, LDSHA_0X88, LDSHA_0X8A, LDSHA_0X8B, LDSW, LDSWA_0X14, LDSWA_0X15, LDSWA_0X1C, LDSWA_0X1D, LDSWA_0X36, LDSWA_0X80, LDSWA_0X83, LDSWA_0X88, LDSWA_0X89, LDSWA_0X8B, LDUBA_0X14, LDUBA_0X15, LDUBA_0X1C, LDUBA_0X1D, LDUBA_0X30, LDUBA_0X38, LDUBA_0X4, LDUBA_0X80, LDUBA_0X88, LDUBA_0X8B, LDUH, LDUHA_0X14, LDUHA_0X15, LDUHA_0X1C, LDUHA_0X1D, LDUHA_0X39, LDUHA_0X3E, LDUHA_0X4, LDUHA_0X80, LDUHA_0X83, LDUHA_0X88, LDUHA_0XC, LDUW, LDUWA_0X14, LDUWA_0X15, LDUWA_0X1C, LDUWA_0X1D, LDUWA_0X31, LDUWA_0X39, LDUWA_0X3E, LDUWA_0X4, LDUWA_0X80, LDUWA_0X81, LDUWA_0X82, LDUWA_0X83, LDUWA_0X88, LDUWA_0X89, LDUWA_0X8A, LDUWA_0X8B, LDUWA_0XC, LDX, LDXA_0X14, LDXA_0X15, LDXA_0X1C, LDXA_0X1D, LDXA_0X4, LDXA_0X41, LDXA_0X63, LDXA_0X80, LDXA_0X81, LDXA_0X82, LDXA_0X83, LDXA_0X88, LDXA_0X89, LDXA_0X8B, LDXA_0XC, RDASI, RDASR, RDCCR, RDFPRS, RDGSR, RDHPR, RDPR, RDTICK, and RDY.
7. The following instructions are counted as stores, where the _0XZZ suffix denotes an alternate-form store with an ASI value of ZZ_{16}: FLUSH, MEMBAR except MEMBAR #Lookaside and MEMBAR #LoadStore, STBA_0X14, STBA_0X15, STBA_0X1C, STBA_0X1D, STBA_0X30, STBA_0X31, STBA_0X36, STBA_0X38, STBA_0X39, STBA_0X3E, STBA_0X4, STBA_0X80, STBA_0X81, STBA_0X88, STBA_0X89, STBA_0XC, STBAR, STD, STDA_0X14, STDA_0X15, STDA_0X1C, STDA_0X1D, STDA_0X27, STDA_0X2F, STDA_0X4, STDA_0X80, STDA_0X88, STDA_0XEA, STDA_0XEB, STDF, STDFA_0X17, STDFA_0X4, STDFA_0X80, STDFA_0X88, STDFA_0X89, STDFA_0XC0, STDFA_0XC2, STDFA_0XC4, STDFA_0XC8, STDFA_0XCA, STDFA_0XCB, STDFA_0XCC, STDFA_0XD1, STDFA_0XD3, STDFA_0XD9, STDFA_0XE0, STDFA_0XF0, STDFA_0XF8, STDFA_0XF9, STF, STFA_0X14, STFA_0X15, STFA_0X1C, STFA_0X4, STFA_0X80, STFA_0X81, STFA_0X88, STFA_0X89, STFA_0XC, STFSR, STHA_0X14, STHA_0X15, STHA_0X1C, STHA_0X1D, STHA_0X30, STHA_0X31, STHA_0X4, STHA_0X80, STHA_0X88, STHA_0XC, STW, STWA_0X14, STWA_0X15, STWA_0X1C, STWA_0X1D, STWA_0X38, STWA_0X4, STWA_0X80, STWA_0X81, STWA_0X88, STWA_0XC, STX, STXA_0X14, STXA_0X15, STXA_0X1C, STXA_0X1D, STXA_0X22, STXA_0X23, STXA_0X26, STXA_0X27, STXA_0X2A, STXA_0X2B, STXA_0X2E, STXA_0X2F, STXA_0X41, STXA_0X73, STXA_0X80, STXA_0X81, STXA_0X88, STXA_0XC, STXA_0XE2, STXA_0XE3, STXA_0XEA, STXA_0XEB, STXA_0XF2, STXA_0XF3, STXA_0XFA, and STXA_0XFB.
8. The following instructions are counted as SPR ring ops, where the _0XZZ suffix denotes an alternate-form load or store with an ASI value of ZZ_{16}: LDXA_0X20, LDXA_0X21, LDXA_0X25, LDXA_0X45, LDXA_0X48, LDXA_0X49, LDXA_0X4C, LDXA_0X4E, LDXA_0X4F, LDXA_0X52, LDXA_0X54, LDXA_0X58, LDXA_0X64, LDXA_0X74, LDXA_0XB0, STXA_0X20, STXA_0X21, STXA_0X25, STXA_0X42, STXA_0X45, STXA_0X4C, STXA_0X4E, STXA_0X4F, STXA_0X50, STXA_0X52, STXA_0X54, STXA_0X57, STXA_0X58, STXA_0X5C, STXA_0X5F, STXA_0X64, STXA_0X72, STXA_0XB0, WRASI, WRASR, WRCCR, WRFPRS, WRGSR, WRHPR, WRPAUSE, WRPR, and WRY. Accesses to STICK_ENABLE do not go onto the SPR ring but are still counted as SPR ring ops.
9. All other instructions: ADD, ADDCcc, ADDXCcc, ALLCLEAN, AND, ANDcc, ANDN, ANDNcc, CASA, CASXA, DONE, FLUSHW, HALT, INVALW, LDSTUB, LDSTUBA, MEMBAR #Lookaside, MEMBAR #LoadStore, MOVdTOX, MOVFN, MOVR, MOVxTOd, NOP, NORMALW, OR, ORcc, ORNcc, OTHERW, PREFETCH, PREFETCHA, RDCFR, RDPC, RESTORED, RETRY, SAVE, SAVED, SETHI_%G0, SIAM, SLLX, SRAX, SRLX, SUB, SUBcc, SUBCcc, SWAP, SWAPA, TADDcc, TADDccTV, TN, TSUBcc, TSUBccTV, WRCFR, XNOR, XNORcc, XOR, and XORcc.
10. Once SPARC T4 detects a mispredicted branch at branch execution time, it computes the correct path address and starts fetching down that path. However, instructions wait at the SEL pipe stage until the mispredicted branch (and its delay slot, if any) commit. This mask value counts cycles that the strand is stalled with correct path instructions ready to be issued but waiting on commit to catch up to the mispredicted branch.
11. A duplicate miss is a miss for which another thread has already missed in the cache for the line, and the cache fill is pending. SPARC T4 does not count duplicate I-cache misses but does count duplicate D-cache misses.
12. The vast majority of SPARC instructions are decoded into one micro-op. Notable exceptions include block loads and stores, which have 8 micro-ops each, and twin loads, which have 2 micro-ops each.
13. These events cross strands. However, each strand's PCR determines which events will be counted based upon only that strand's privilege level. For example, if only strands 0 and 1 are active, PCR0 is set to count fetch-0-all cycles (PCR.sl=8, PCR.mask=2_{16}) in user and privilege modes, strand 1 is in hyperprivileged mode, and neither strand fetches an instruction during a given cycle, that cycle will be counted in strand 0's PCR. In general, specifying privivilege modes is not useful for these events, so software should set PCRx.ut, PCRx.st, and PCRx.ht to 1 when counting these events.

Programming Note (sl 28) For sl 28 (Commit micro-ops), software should take notice of the following race condition. If the associated PIC is set to -1, -2, or -3, and the PCR.mask field is set to 01_{16}, 02_{16}, or 03_{16}, the PIC increments within two cycles after the PCR is set. If software also tries to write to the PCR during this period, it will overwrite the about-to-be-set PCR.ov bit. To avoid this situation, software should set the PIC outside of the range -1..-3 inclusive.

Programming Note The trap handler for a *disrupting_performance_event* trap must clear PCRx.ov, otherwise hardware may continuously trigger a trap.

Programming Note To reliably clear PCRx.ov when counting asynchronous events software should stop counting (clear PCRx.ut, PCRx.ht, and PCRx.st). Otherwise, hardware may overwrite PCRx.ov if the associated PIC overflows.

Programming Note STXA %pcr, WRHPT %hpstate, and WRPR %pstate affect an asynchronous count immediately (before the instruction changing the PCR, HPSTATE, or PSTATE has retired from the pipeline).

10.3 SPARC Performance Instrumentation Counter

Each virtual processor has four Performance Instrumentation Counter registers: PIC0, PIC1, PIC2, and PIC3. PCR0 controls PIC0, PCR1 controls PIC1, PCR2 controls PIC2, and PCR3 controls PCR3. Access privilege is controlled by the settings of PCR.picnht and PCR.picnpt. When PCR.picnht = 1 an attempt to access this register in privileged or nonprivileged mode causes a *privileged_action* trap. When PCR.picnpt = 1 an attempt to access this register in nonprivileged mode causes a *privileged_action* trap.

The PIC counter contains a single 32-bit counter field. The field counts the event selected by PCR.sl. The ut, st, and ht fields for PCR control which combination of user, supervisor, and/or hypervisor events are counted.

Performance counter overflows a) set PCR.ov, and b) generate a hyperprivileged trap if PCR.toe is set. The trap which is generated depends upon whether the event being counted is synchronous or asynchronous, as denoted in TABLE 10-2 above. If the event is asynchronous, a *disrupting_performance_event* trap is generated; otherwise, a *precise_performance_event* trap is generated. For precise traps, the instruction that caused the overflow will not have been executed, and the PC and NPC of the instruction will be captured on the trap stack, with the following caveat. The *precise_performance_event* trap is delivered precisely to hypervisor in user or privileged mode only (HPSTATE.hpriv == 0). If HPSTATE.hpriv == 1, the trap is lost[1], even if PCR.ht == 0. Additionally, setting PCR.ht == 1 prevents any *precise_performance_event* trap from occurring. Thus, a hypervisor wishing to monitor performance of hyperprivileged events should poll the counters instead of using precise overflow traps.

The format of the PIC registers are shown in TABLE 10-3.

1. The hypervisor design team does not wish to take precise performance monitor traps in the hypervisor.

TABLE 10-3 Performance Instrumentation Counter Register – PIC0-3 (ASI $B0_{16}$, VA 00_{16}, 08_{16}, 10_{16}, 18_{16})

Bit	Field	Initial Value	R/W	Description
63:32	—	0	RW	*Reserved*
31:0	counter	0	RW	Programmable event counter, event controlled by PCR.sl.

10.4 PXM Coverage Counters

The crossbar interface between processor core pairs and the crossbar contains a counter select register and two counters that can be configured to count various crossbar events. See Section 21.4.3, Section 21.4.4, and Section 21.4.5 for more detail.

10.5 DRAM Performance Counter

The memory controller has four performance counters, 32 bits each. Two counters together form a group and a group is kept in a register. There are two suchregisters: DRAM_PERF_COUNT01_REG and DRAM_PERF_COUNT23_REG. There is a control register, DRAM_PERF_CTL_REG, that allows independant control of the 4 performance counters.

There are four COU ports in a MCU and the MCU has a counter for each COU port. For example, when asked to count reads only, each counter counts the read requests from its respective COU port. So the total number of reads to a MCU is the sum of all four counters. Likwise, when asked to monitor writes or reads+writes, each counter does that for its assigned COU port. The total count is the sum of all four counters.

The four performance counters are divided into two groups. DRAM_PERF_COUNT01_REG is one group and DRAM_PERF_COUNT23_REG is the other group. The two groups of counters are exlcusively assigned to two different pieces of software (i.e. Solaris has been assigned DRAM_PERF_COUNT01_REG and does not use DRAM_PERF_COUNT23_REG registers). The Power Manangement (PM) Software has been assigned DRAM_PERF_COUNT23_REG and does not use DRAM_PERF_COUNT01_REG.

In light of this division of the counters among the two software entities, SPARC T4 has to make sure that with only two counters, either Solaris or the PM software is able to count the entire read, write and read+write traffic coming to a memory controller. In addition, the PM counters DRAM_PERF_COUNT23_REG are capable of independently counting the memory accesses to the two memory channels. Also to be noted is that counters in DRAM_PERF_COUNT01_REG are identical in behavior to counters in DRAM_PERF_COUNT23_REG.

TABLE 10-4 DRAM Performance Control Register – DRAM_PERF_CTL_REG (F804 0000 0408_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
63:16	—	X	RO	*Reserved*
15:12	sel3	0	RW	Select code for performance counter 3 which counts the events from cou1 port1.

TABLE 10-4 DRAM Performance Control Register – DRAM_PERF_CTL_REG (F804 0000 0408_{16}) (Count 4 Step 4096)

Bit	Field	Initial Value	R/W	Description
11:8	sel2	0	RW	Select code for performance counter 2 which counts the events from cou1 port0.
7:4	sel1	0	RW	Select code for performance counter 1 which counts the events from cou0 port1.
3:0	sel0	0	RW	Select code for performance counter 0 which counts the events from cou0 port0.

TABLE 10-5 shows the format of the DRAM Performance Counter 0 and 1 Register.

TABLE 10-5 DRAM Performance Counter 0 and 1 Register – DRAM_PERF_COUNT01_REG (F804 0000 0410_{16}) (Count 4 Step 4096)

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

TABLE 10-6 shows the format of the DRAM Performance Counter 2 and 3 Register.

TABLE 10-6 DRAM Performance Counter 2 and 3 Register – DRAM_PERF_COUNT23_REG (F804 0000 0418_{16}) (Count 2 Step 4096)

Bit	Field	Initial Value	R/W	Description
63	sticky3	0	RW	Sticky overflow for counter 3.
62:32	counter3	0	RW	Performance counter 3.
31	sticky2	0	RW	Sticky overflow for counter 2.
30:0	counter2	0	RW	Performance counter 2.

TABLE 10-7 DRAM Performance Counter Select Codes

Select	Description
0000_2	Read transactions. The counter will count reads from its respective COU port only.
0001_2	Write transactions. The counter will counter writes from its respective COU port only.
0010_2	Read + write transactions. The counter will count reads+writes from its respective COU port only.
0011_2	Bank busy stalls; incremented by 1 each cycle there are requests in the queue, but none can issue because of bank conflicts
0100_2	Read queue latency; incremented by *n* each cycle, where *n* is the number of read transactions in the queue.
0101_2	Write queue latency; incremented by *n* each cycle, where *n* is the number of write transactions in the queue
0110_2	(Read + Write) queue latency; incremented by *n* each cycle, where *n* is the number of transactions in the queue
0111_2	Writeback buffer hits; incremented by 1 each time a read transaction is deferred because it conflicts with a queued write transaction.
1000_2	IThe combined count of all reads (from the four COU ports) to both the memory channels, 0 and 1. This field results in each counter independently counting the total read traffic to a memory controller.
1001_2	The number of times a write starved.
1010_2	The combined count of all writes (from the four COU ports) to both the memory channels, 0 and 1. This field results in each counter independently counting the total write traffic to a memory controller.
1011_2	The number of reads and writes issued to memory channel 0. To get the total number of memory transactions (reads and writes) in a memory controller, Solaris must program one of it's two counters with this select field and it's other counter with select value = C_{16}.

TABLE 10-7 DRAM Performance Counter Select Codes

Select	Description
1100_2	The number of reads and writes issued to memory channel 1. To get the total number of memory transactions (reads and writes) in a memory controller, Solaris must program one of it's two counters with this select value and it's other counter with select value = B_{16}.
1101_2	*Reserved*
$111x_2$	*Reserved*

10.6 PCI-EX Performance Counters

Each PCI-Express function implemented in the DMU and the PEU clusters maps a pair of 64-bit performance counters and a corresponding event selection register to PCIe memory space, through a PCI-compliant BAR in the given function's PCIe Configuration Space. Section 22.8, *Performance Counters*, on page 602 contains a summary of the events that can be counted in both the DMU and the PEU. The detailed definitions of the performance counter and event selection CSRs are given in the CSR definitions subsections for the DMU and PEU, respectively, in Chapter 22.

10.7 Ethernet Performance Counters

Ethernet performance counters are described in *DRR Performance Monitoring* on page 983, *Port Scheduler* on page 934, and *Receive Performance Management and Discard Statistics* on page 951.