Errata for UltraSPARC-III

Part No. 820-4006-10 *12/19/07*

Sun Microsystems, Inc.
www.sun.com

Submit comments about this document at: `http://www.sun.com/hwdocs/feedback`

Errata for UltraSPARC-III - 12/19/07
Part No. 820-4006-10

1. Errata Table

Table 1: UltraSPARC-III Errata Table

Errata #	Version 3.1	Version 3.2	Version 3.3	4Version 3.4	See ...
1	✔	✔	✔	✔	page 5
2	✔	✔	✔	✔	page 5
3	✔	✔	✔	✔	page 7
4	✔	✔	✔	✔	page 9
5	✔	✔	✔	✔	page 10
6	✔	✔	✔	✔	page 11
7	✔	Not applicable	Not applicable	Not applicable	page 13
8	✔	Not applicable	Not applicable	Not applicable	page 13
9	✔	Not applicable	Not applicable	Not applicable	page 14
10	✔	Not applicable	Not applicable	Not applicable	page 15
11	✔	✔	✔	✔	page 16
12	✔	✔	✔	✔	page 17
13	✔	✔	✔	✔	page 18
14	✔	✔	✔	✔	page 18
15	✔	✔	✔	✔	page 19
16	✔	✔	✔	Not applicable	page 20
17	✔	✔	✔	✔	page 22
18	✔	✔	✔	✔	page 22
19	✔	✔	✔	✔	page 24
20	✔	✔	✔	✔	page 25
21	✔	✔	✔	✔	page 25
22	✔	✔	✔	✔	page 27
23	✔	✔	✔	✔	page 29
24	✔	✔	✔	✔	page 30

Table 1: UltraSPARC-III Errata Table (Continued)

Errata #	Version 3.1	Version 3.2	Version 3.3	4Version 3.4	See ...
25	✔	✔	✔	✔	page 32

2. Errata Descriptions and Workarounds

Erratum #1: An instruction breakpoint is not taken on a mispredicted annulling delay slot.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

This bug is for breakpoint instructions in the delay slot of the predicted not-taken, actually-taken Delayed Control Transfer Instruction (DCTI) case. In this scenario, the delay slot may be requeued for execution from the CPU's mispredict queue (MQ), in which case the breakpoint mechanism will not be triggered. The delay slot instruction is executed, but no breakpoint trap is taken.

Impact:

When breakpoint is used for debugging purposes, this should not be a significant problem. However, it does preclude using the breakpoint feature as a field patch mechanism to repair the execution of an instruction type found to have an implementation bug.

Workaround:

None.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #2: A Store Extended Word into Alternate Space (STXA) to the Instruction Memory Management Unit (IMMU) register is corrupted by an IMMU miss.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

A Store Extended Word into Alternate Space (STXA) instruction that targets an Instruction Memory Management unit (IMMU) register (e.g., ASI_IMMU_SFSR, where ASI = 0x50 and Virtual address = 0x18) is executed. Prior to the programmer visible state being modified, an IMMU miss is taken. Example:

```
pc                  instruction
--                  -----------
0x25105e21ffc:      stxa %g0, [%i0]0x50
0x25105e22000:      IMMU miss
```

It is possible for both the STXA and the IMMU miss to attempt to update the targeted register at the exact same instant. If this occurs, the IMMU miss should take priority. However, in this case, the STXA instruction has higher priority, causing stale data to be stored.

This is a special case of the documentation guideline that specifies:

> "A FLUSH, DONE, or RETRY is needed after stores to internal Address Space Identifies (ASIs) that affect instruction accesses."

The intended requirement is that the FLUSH, DONE, or RETRY should immediately follow the STXA. In this case, the programmer may have inserted such an instruction after the STXA instruction, but this instruction was not processed prior to the IMMU miss.

Workaround:

There are two ways of making sure such a case does not occur:

1. Any code that modifies the IMMU state should be locked down in the Instruction Translation Look-Aside Buffer (ITLB) to prevent the possibility of intervening TLB misses.
2. If suggestion (1) is not possible, the STXA and the subsequent FLUSH, DONE, or RETRY should be kept on the same 8 KB page, again preventing an intervening ITLB miss.

Status:

This erratum updates the documentation.

Erratum #3: A Store Extended Word into Alternate Space (STXA) from an internal Address Space Identifier (ASI) returns corrupted data for a subsequent LDXA from an internal ASI due to an exception that occurred prior to protection from a MEMBAR #Sync.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

In one example of this problem, the following code sequence was being run:

```
101a9e38  init_mondo+24 stxa    %o1, [%g0 + 50] %asi
101a9e3c  init_mondo+28 stxa    %o2, [%g0 + 60] %asi
101a9e40  init_mondo+2c jmp     %o7, 8, %g0
101a9e44  init_mondo+30 membar  #Sync
```

%asi was set to 0x5a for this sequence.

In this particular case, a vector interrupt trap was taken on the JMP instruction. The interrupt trap handler executed an LDXA to the Interrupt Dispatch Register (ASI 0x77 - ASI_INTR_DISPATCH), which returned indeterminant data as the last Store Extended Word into Alternate Space (STXA) was still in progress (in this case, the data returned was the data being written by the last STXA).

The reason the above case failed is that the JMP instruction took the interrupt before the MEMBAR #Sync semantics were invoked, thus leaving the interrupt trap handler unprotected.

In this code sequence, the JMP is also susceptible to the following traps (in addition to interrupts):

- mem_address_not_aligned
- Illegal instruction
- Instruction breakpoint (debug feature which manifests itself as an illegal instruction, but is currently unsupported)
- Instruction Memory Management unit (IMMU) miss
- Instruction access exception
- Instruction access error
- Fast ECC error

The documentation contains the following note:

```
"*Caution* - ... A Store to an MMU Register requires either a
MEMBAR #Sync, FLUSH, DONE, or RETRY before the point that the
effect must (should be 'can') be visible to load/store/atomic
accesses ... one of these instructions must be executed
before the next ... load of any type and on or before the
delay slot of a delayed control transfer instruction of any
type. This is necessary to avoid corrupting data."
```

The above code sequence does follow these guidelines; however, there is not a hardware interlock in place to guarantee correct operation.

Impact:

In this case, a MEMBAR #Sync was placed on the delay slot, which is insufficient in the current implementation. A better phrasing would be:

```
" ... one of these instructions must be executed before the
next ... load of any type and ["on or" phrase removed] before
the delay slot of a delayed control transfer instruction of
any type. This is necessary to avoid corrupting data."
```

Workaround:

This workaround has two parts.

1 The first part to the specific problem observed is the following code change:

```
101a9e38  init_mondo+24 stxa    %o1, [%g0 + 50] %asi
101a9e3c  init_mondo+28 stxa    %o2, [%g0 + 60] %asi
101a9e40  init_mondo+2c membar  #Sync
101a9e44  init_mondo+30 jmp     %o7, 8, %g0
```

This approach works even though both the second STXA and MEMBAR #Sync can take interrupts. The STXA to an MMU Register and MEMBAR #Sync implicitly wait for all previous stores to complete before starting down the pipeline.

Thus, if the second STXA or Memory Barrier Instruction (MEMBAR) takes an interrupt, it does so only at the end of the pipeline after having made sure that all previous stores were complete.

In the above case, the MEMBAR #Sync is still susceptible to all the traps noted above (except interrupts and mem_address_not_aligned):

- IMMU miss

- Illegal instruction
- Instruction breakpoint (debug feature which manifests itself as an illegal instruction, but is currently unsupported)
- Instruction access exception
- Instruction access error
- Fast ECC error

A DONE or RETRY can take a privileged opcode trap if used in place of the MEMBAR #Sync. This possibility is not considered since as any STXA's that target internal ASIs must be done in privileged mode.

2 The second part of the workaround is to start any of the vulnerable trap handlers with a MEMBAR #Sync, especially if they use LDXA instructions that target internal ASIs (e.g., ASI values: 0x30-0x6f, 0x72-0x77, and 0x7a-0x7f).

In the case of the IMMU miss handler, this approach may result in unacceptable performance reduction. In such a case, it is recommended that both the STXA and the protecting MEMBAR #Sync (or FLUSH, DONE, or RETRY) are always on the same 8K page, thus eliminating the possibility of an intervening IMMU miss (unless the code is otherwise guaranteed to not take an IMMU miss; it was guaranteed to be locked down in the TLB, for example).

The workaround described should be sufficient in all cases where an STXA to an internal ASI is either followed immediately by another such STXA or by one of the protecting instructions: MEMBAR #Sync, FLUSH, DONE, or RETRY.

In cases where other interruptible instructions are used after an STXA and before a protecting instruction, any exception handlers that can be invoked would need similar protection. Such coding style is strongly discouraged and should only be done with great care when there are compelling performance reasons (e.g., in TLB miss handlers).

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #4: Multiple bits of the Asynchronous Fault Status Register (AFSR) are set for the same ECC error.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

When memory data with uncorrectable ECC errors are bypassed to the Data Cache, a deferred trap is signaled 7 CPU cycles after the Data Cache receives the data. We assert the fast_ecc_error signal to the Data Cache even though the data is from memory and not from the External Cache.

Impact:

The traps are prioritized and only the deferred trap is taken. Multiple bits in the AFSR are also set for the same error.

Workaround:

None.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #5: The valid bits in the Instruction Translation Look-Aside Buffer (ITLB) and Data Translation Look-Aside Buffer (DTLB) should remain unchanged after a System Reset.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The documentation states that the Data and Instructions TLB States after a System Reset are unchanged when in fact they are unknown. The documentation will be changed from "unchanged" to "unknown".

Impact:

After a System Reset, all Large Page Entries and Locked 8K Pages entries will be in an unknown state because the Lock, Global and Valid Bits are cleared.

Workaround:

System software (Open Boot Prom [OBP] and kernel) are able to recover the content of the TLB entries.

Status:

This erratum updates the documentation.

Erratum #6: The performance counters have accuracy and counting issues.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

1 The Instruction Translation Look-Aside Buffer (ITLB) and Data Translation Look-Aside Buffer (DTLB) miss performance counter accuracy issue.

The ITLB and DTLB counters can only be counted in system trace capture mode. The ITLB miss and DTLB miss signals arrive in trapcycle 3 of a trap and are used for performance measurements in trapcycle 5. By trapcycle 5, however, the pstate_priv mode has turned on, thus making this performance counter useless for measuring user traces. Currently, any TLB misses can only be counted in system trace capture mode (PCR.ST set).

2 Recirculate performance counter accuracy issue.

The current counts in Data Cache (Dcache) for following recirculate conditions seem too low:

raw hazard, load misses Dcache, load misses Dcache and External Cache (Ecache), load has endian mismatch, prefetch load misses Prefetch Cache (Pcache) in the second load (in the MS pipe).

The cause is that the Dcache asserts this count once for each of these recirculate conditions, but the new requirement is that the count should be asserted as long as the recirculating condition is valid and should reflect the penalty taken by the load due to the recirculate by incrementing these counters once for each recirculate cycle.

3 The IC_ref PIC does not count accurately.

The implementation of the performance counter IC_ref[PIC0] does not count Instruction Cache (ICache) misses. It only counts ICache hits. This can cause an unexpectedly low count in the PIC when, for example, executing non-ICache-resident code. It also affects Icache-resident code, although the magnitude of the error would not be as significant.

4 The %pic overflow trap got dropped because of a one-cycle hole.

There is a one-cycle hole in ms_spreg_ctl that fails to flag the performance counter overflow trap (actually interrupt_level_15 trap). The following example shows the scenario.

```
%pcr.ST=0, %pcr.UT=1 (only user mode count)
ie_pstate_priv changed 0 to 1 at cycle X
arch_pic0 causes overflow at cycle X+1
```

A pic_event_flopped signal changes from 1 to 0 at cycle X+1. This change illegally suppressed the %pic overflow trap.

5 The external Cache Unit's (ECU) miss block should not assert EC_miss for non-cacheable requests.

The ECU Miss_block erroneously includes the non-cacheable requests in the count for the total number of Ecache misses that were sent to the System Interface Unit (SIU) for further processing.

6 The ECU should not assert REC_ref for non-cacheable requests.

The definition of EC_ref is as following: Total External Cache references excluding the non-cacheable accesses and 64-byte request is counted as 1. The ECU erroneously includes the non-cacheable accesses in this count.

7 The ECU should not assert EC_rd_miss for Pcache and Wcache requests and non-cacheable requests.

The ECU’s miss_block erroneously includes Pcache requests, Write Cache (Wcache) requests, and non-cacheable requests in the count of Ecache read misses due to Dcache requests.

8 The ECU should not assert EC_ic_miss for no-ncacheable requests.

The ECU’s miss_block erroneously includes the non-cacheable requests in the count of Ecache read misses due to the Icache requests.

9 The Re_EC-miss count is inaccurate.

The behavior of Performance Instrumentation Counter (PIC) bit Re_EC_miss makes it impossible to compute an accurate CPI penalty for recirculates due to an ECache miss because the Re_EC_miss counts some unspecified fraction of the penalty cycles caused by ECache misses. Since it is not possible to count either the number of load recirculates that missed the ECache nor the number of penalty cycles that were not counted in a Re_EC_miss on each of those recirculates, it is often impossible to extrapolate to the actual cycle count.

Impact:

Performance counts are inaccurate.

Workaround:

None.

Status:

These bugs will not be fixed in future releases of the silicon.

Erratum #7: A Write Cache (Wcache) does not follow a Total Store Ordering (TSO) for a nc_st non-cacheable store.

Applicability:

UltraSPARC-III Version 3.1.

Description:

The Write Cache (Wcache) coalescing logic for non cacheable stores is aggressive.

Impact:

Non-cacheable stores may be sent to the SIU out of order.

Workaround:

Turn off the store compression (store merging) for non-cacheable stores.

Status:

This bug is fixed in Version 3.2 of the processor.

Erratum #8: An incorrect write-after-write hazard in detected in the Floating Point and Graphics Unit (FGU).

Applicability:

UltraSPARC-III Version 3.1.

Description:

A potential WAW hazard turns into a false one and causes bypassing to be turned off.

Impact:

Branches mispredict and the delay slot get annulled. The annulled delay slot causes the temporary WAW hazard that vanishes.

Workaround:

None.

Status:

This bug is fixed in Version 3.2 of the processor.

Erratum #9: UltraSPARC-III supports aliasing of the Strong Prefetch to the Weak Prefetch.

Applicability:

UltraSPARC-III Version 3.1.

Description:

UltraSPARC-III supports aliasing of the Strong Prefetch to the Weak Prefetch implemented in UltraSPARC-III version 3.2. This allows application code to be developed to include the Strong Prefetch code with UltraSPARC-III and get the full support of the Strong Prefetch feature later from UltraSPARC-IV+.

The PREFETCH(A) instructions are implemented this way starting in UltraSPARC-III Version 3.2.

Table 2: PREFETCH function code map

fcn	Mapped to fcn	Operation
0x0	0x0	Prefetch, 0 (Prefetch for Several Reads)
0x1	0x1	Prefetch, 1 (Prefetch for One Read)
0x2	0x2	Prefetch, 2 (Prefetch for Several Writes)

Table 2: PREFETCH function code map (Continued)

fcn	Mapped to fcn	Operation
0x3	0x3	Prefetch, 3 (Prefetch for One Write)
0x4 - 0x13	N/A	Causes *illegal_instruction* trap
0x14	0x0	Prefetch, 0 (Prefetch for Several Reads)
0x15	0x1	Prefetch, 1 (Prefetch for One Read)
0x16	0x2	Prefetch, 2 (Prefetch for Several Writes)
0x17	0x3	Prefetch, 3 (Prefetch for One Write)
0x18 - 0x31	N/A	Treated as NOP

Status:

This feature is not implemented in UltraSPARC-III Version 3.1.

Erratum #10: The ec_fill_data_dont_fill signal is off by 1 cycle, resulting in wrong data installed in the Instruction Cache (Icache).

Applicability:

UltraSPARC-III Version 3.1.

Description:

In the External Cache Unit (ECU), requests are not always processed in the order in which they arrive to the ECU. This out-of-orderness does not cause any problem, since the ECU always probes the Write Cache (Wcache) to get the latest data and change the state of the Wcache line whenever it accesses the External Cache (Ecache)/Ecache Tag.

The Wcache also sends out the invalidate request to the Icache and Pcache when it sends the exclusive request to the ECU to invalidate the line if it is present in the Icache and Pcache. In the case the primary cache request misses the Ecache, the ECU will not probe the Wcache. Instead, there are special logic in the ECU to ensure that the processor coherency is still maintained.

When the SIU returns data for the Read To Share (RTS), the special logic in the ECU compares the address of the SIU request against that of the pending Wcache exclusive request in the ECU and asserts dont_fill to the Icache so that the returned data will not be installed in the Icache.

This is needed since ECU gives the exclusive grant permission to the Wcache for that same line. Since the Wcache will not send out an invalidate request to the Icache and Prefetch Cache (Pcache) once it has the line in an exclusive state, the data in the Icache or Pcache will be wrong if it is a copy.

The dont_fill signal is asserted 1 cycle too early, hence the returned data is installed in the Icache.

Impact:

However, stale data can end up in the Instruction Cache (Icache). In this particular bug, there are a Wcache exclusive request and Icache request to the same address to the ECU. Both requests miss the Ecache. The Wcache request comes to the ECU first; however the Icache request is serviced first and a Read To Share (RTS) is sent out to the Fireplane bus.

Workaround:

Do not use self-modifying code.

Status:

This bug is fixed in Version 3.2 of the processor.

Erratum #11: A Quad Load may return stale data under certain conditions.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

Quad Loads always get data from memory and not from Dcache. In this case, when the store misses Dcache, then the Quad Load gets correct data.

```
sth     DATA,[foo+8]     // Store hits Data Cache (Dcache).
ldda    [foo]%asi,%reg   // asi has to be one of QUAD_LDD
                         // variants for atomic TTE accesses
                         // ... which is privileged ...
```

Impact:

In this case, the ldda accessing a 16 B aligned chunk of memory may not get the correct data from the store when the Store hits Dcache.

Workaround:

Use MEMBAR #Sync before Quad Loads from a cacheable space.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #12: The Asynchronous Fault Status Register (AFSR) PRIV bit is captured but not held for the TO and Bus error from system bu (BERR) errors in privilege mode.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

In privilege mode, the Asynchronous Fault Status Register AFSR.PRIV bit is captured but not held when a TO error occurs.

Impact:

The bit is captured as set, but is cleared 1 CPU cycle later.

Workaround:

The trap handler should use TSTATE.PSTATE.PRIV instead of the AFSR PRIV bit. However, such a deferred trap caused by the user code could be taken at a later time, so that there is some possibility of a kernel panic.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #13: The Asynchronous Fault Status Register AFSR.PRIV bit is not implemented as a RW1C.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The Asynchronous Fault Status Register AFSR.PRIV bit is supposed to be reset to zero by writing a 1 to it via the Address Space Identifier (ASI). However, it is reset to zero when the associated error bit is reset to 0, similar to the behavior of E_SYND & M_SYND.

For examp[le, assume that the AFSR = 0. If an Interrupt vector uncorrectable ECC error (IVU) occurs in the privilege mode, then the processor will capture IVU and PRIV, so that:

```
IVU = 1, PRIV = 1, E_SYND = the syndrome bit sent from SIU
```

If the trap handler write 1'b1 to AFSR IVU bit, then:

```
IVU = 0, PRIV = 0, E_SYND = 0
```

Impact:

All errors must be cleared before the AFS register PRIV bit can be reset.

Workaround:

None.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #14: The Asynchronous Fault Status Register AFSR.ME bit is not implemented as a RW1C.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The Asynchronous Fault Status Register AFSR.ME bit is not reset to zero by writing a 1 to it. It is cleared when all errors are cleared.

Impact:

All errors must be cleared before the AFS register ME bit can be reset.

Workaround:

None.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #15: A fadd or fsub instruction produces a subnormal result traps to the kernel with an unfinished trap even in nonstandard mode.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

Run the following test program:

```
>
        equivalence (d,id)
        equivalence (x,ix)
        equivalence (y,iy)
        x=1.24895E-38
        y=1.17550E-38
        print *,' x ',x,ix
        print *,' y ',y,iy
        d=x-y
        if (d .eq. 0) print *,' d is zero '
        print *,' d ',d,id
        end
>
```

Compile and execute to get the following results:

```
>
        x     1.24895E-38  8912805
        y     1.17550E-38  8388648
        d   0.  524157
>
```

The subnormal DIFFERENCE x-y is computed correctly in nonstandard mode.

Impact:

The detection of the subnormal results and then forcing them to a predetermined value within the targeted cycle time is difficult and therefore, an unfinished trap is generated to ask the kernel to handle the rest.

The kernel recomputation software never expects to be called in nonstandard mode, so it does not interrogate the state of the Graphics Status Register GSR.IM bit before computing the result.

Workaround:

Change the kernel recomputation software to examine the Floating Point Status Register FSR.NS and GSR.IM bits in the nonstandard mode.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #16: A Pixel Component Distance (Pdist) instruction recirculate causes a Store Queue (STQ) mux-exclusivity violation, resulting in wrong data being stored in Data Cache (Dcache).

Applicability:

UltraSPARC-III Versions 3.1, 3.2, and 3.3.

Description:

When a pdist instruction recirculate occurs, the mux signals for selecting the Store Queue (STQ) data to retire to the Data Cache (Dcache) cause wrong data to be retired to the Dcache.

If there is a non-store ms pipe instruction before the pdist and the next store, the problem will not occur. Example:

```
pipe
------
fm       pdist (possible recirculate)
ms       non-store instruction (e.g., load, rdasr)

------ instruction group broken ----

ms      store
```

Impact:

Wrong data to be stored in the Dcache.

Workaround:

If there are enough intervening instructions after the pdist and between the store (although perhaps not the ms pipe instructions), the problem will not occur. Example:

```
pipe
------

fm        pdist (possible recirculate)
a0         integer instruction (eg., add %g0,%g0,%g0)
a1         integer instruction (eg., add %g0,%g0,%g0)

------ instruction group broken ----
a0         integer instruction (eg., add %g0,%g0,%g0)
ms       store
```

The pdist recirculate, itself, can be avoided. For good performance, the rd operand of the pdist should not reference the result of a non-pdist instruction in the previous five instruction groups.

The store that ends up in the recirculated pdist's instruction group must be the eighth one outstanding. If one can guarantee that this is not the case, then the problem will also not occur.

Status:

This bug is fixed in Version 3.4 of the processor.

Erratum #17: An interrupt sent encountered a timeout, which might set the Multiple Event (ME) bit.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

When the processor sends an interrupt to a non existent device, the interrupt comes back unmapped, which causes an Asynchronous Fault Status Register Timeout AFSR.TO to be set and a trap. However, the Multiple Event (ME) bit is also being set, which is not expected.

Impact:

An incoming interrupt xact with an unmapped address is dequeued from the Coherent Pending Queue (CPQ). Queue Control (QCTL) might assert a Timeout (TO) signal more than 2 cycles, so that Cregs will set the TO bit in the AFSR and also set the ME bit.

Workaround:

None.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #18: A window of vulnerability exists within which stale data could be installed in Prefetch Cache (Pcache) for Floating Point loads encountering Read-After-Write or Instruction Cache (Icache) for self-modifying code.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The failure is a Floating Point (FP) load encountering the RAW case. For FP-loads, the Read-After-Write (RAW) situation is like a store to address A followed by a FP-load from address A+32. A similar exposure exists in the handling of the self modifying code. This failure mechanism affects all of the supported External Cache (Ecache) modes. The case for self-modifying code involves address A only because Ecache automatically returns the 2nd 32-byte to Instruction Cache (Icache) as part of the instruction prefetch.

A window of vulnerability exists that allows stale data to be installed in the Prefetch Cache (Pcache), for the FP-load encountering RAW case, or Icache, for the self-modifying code case. The window of vulnerability is about 8 to 12 CPU cycles wide. When hitting either case, the Ecache controller doesn't assert the don't_fill signal to tell either the Pcache or Icache not to install stale data. The following conditions are needed to hit this window of vulnerability:

- **Condition 1**: A store to address A initiated an request to the Ecache controller for exclusive ownership. A FP-load from address A+32 follows or an Icache request to fetch code from address A follows. The FP-load or the Icache fetch request missed the Ecache and caused a RTS (read to share) to be sent to the Fireplane bus. Wcache sends an invalidation request to Pcache and Icache at the time it sends the exclusive request to Ecache controller. This request for the exclusive ownership gets blocked by Ecache controller's index blocking mechanism.
- **Condition 2**: Some time later, the CPQ (coherent transactions pending queue) in the System Interface Unit (SIU) just retired a locally initiated RTS (read to share) type transaction (i.e., the entire 64-byte data have been received and sent to the L1 caches; this local_RTS has nothing to do with the RAW situation or the self modifying code) and now the CPQ also found out that the first 32-byte data for the RTS triggered by the FP-load or the Icache fetch have just arrived, but its companion 2nd 32-byte data have not been received yet. Moreover, this RTS is sitting at the head of the CPQ ready to be dequeued.
- **Condition 3**: The CPQ launches the critical-data-forwarding operation to send the critical 32-byte (the first 32-byte data) to Pcache (for the FP-load) or Icache (for the instruction fetch) knowing that the second 32-byte data will be received sometime later but doesn't know when it'll show up. The condition of the CPQ initiating a critical data forwarding operation is also crucial here because the CPQ will launch a L1_fill operation instead should the entire 64-byte data have been received.

Impact:

When all three conditions have been satisfied, there exists a window of vulnerability so that the Ecache controller might not assert the don't_fill signal to either Pcache or Icache to tell them not to install the forwarded data. When the exclusive ownership is later granted to Wcache, the stale data have been installed in Pcache or Icache and there is no mechanism for an invalidation.

Workaround:

Disable Pcache by clearing the PE bit in the ASI_DCU_CONTROL_REGISTER. This prevents the Pcache failure case.

For the instruction fetch case, the only scenario that can cause all three conditions to occur in the failing order is a self-modifying code sequence that is modifying instruction lines in the current instruction stream being executed.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #19: Diagnostic read of a fully associative TLB translation table entry (TTE) will return incorrect data.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

This problem happens under following conditions.

- **Data TLBs**: Any memory access instruction that misses the Data TLB is followed by a diagnostic read access (ldxa or lddfa from ASI_DTLB_DATA_ACCESS_REG, ASI=0x5d) from the fully associative TLBs and the target TTE has page size is set to 64 KB, 512 KB, or 4 MB.
- **Instruction TLBs**: Any instruction that misses the Instruction TLB is followed by a diagnostic read access (ldxa or lddfa from ASI_ITLB_DATA_ACCESS_REG, ASI=0x55) from the fully associative TLBs and the target TTE has page size set to 64 KB, 512 KB, or 4 MB.

Impact:

The data returned from the Translation Table Entry (TLB) will be incorrect.

Workaround:

This problem can be overcome by reading the fully associative TLB TTE twice, back to back. The first access may return incorrect data if the above conditions are met; however, the second access will return correct data.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #20: It is not legal to relax the bstore/bstore* and MEMBAR #Sync rule when Physical Address PA[12:5] match. Order is required.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The Memory Barrier Instruction (MEMBAR) rules table (i.e., table 8-3 in the UltraSPARC-III Users Manual) shows that neither a bstore followed by bstore commit case nor a bstore followed by the bstore case require a MEMBAR #Sync when there is a Virtual Address VA[12:5] match. This is incorrect. The MEMBAR #Sync is required.

Impact:

The failure mode is “wrong data”.

Status:

This erratum updates the documentation.

Erratum #21: One CPU modifying a currently executing load instruction on a second CPU, without explicit synchronization, can cause both hangs and data corruption.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

Basically, one CPU (CPU "A") was executing a load instruction that could not be serviced by the on-chip D Cache. Examples of such loads would include D Cache miss/E Cache hits, E Cache misses, and noncacheable references. When this happens, the instruction is refetched and recirculated down the memory pipeline. At the time of recirculation, a state machine is initiated which depends on the exact same load being fed down the pipeline a second time.

In the failure case, another CPU (CPU "B") modified CPU A's load instruction during a window of time that spans the initial fetching of the load instruction into the instruction queues (which feed the execution pipelines) until the instruction has been refetched the second time.

A load that misses the D Cache, for example, is re-executed along with an extra copy of identical load instructions (referred to as helpers) which are necessary to resynchronize the pipeline with the returning data and simultaneously update the D Cache.

Impact:

The machine's operation depends on how the load is modified.

If the load is turned into a store, the dispatching logic illegally dispatches this as a store and two helpers. The dispatching logic, which keeps track of the number of entries in the Store Queue (STQ), considers this as one store. The STQ logic, on the other hand, interprets these 3 identical-looking store instructions as 3 independent stores and so puts 3 entries on the STQ. When these 3 stores are dequeued, the Store Queue informs the dispatch logic in turn in order to keep the two counters in sync.

In this case, the dispatch logic's counter is decremented 2 extra times and underflows. The counter enters an illegal state from which it cannot recover. A subsequent instruction (e.g., MEMBAR #Sync) that requires a certain STQ state (empty or with a free entry, for example) will not see the required condition and so will wait indefinitely.

Even if the load in not turned into a store, the same behavior can result if the instruction is so modified (into a NOP or Arithmetic Logic Unit (ALU) operation, for example) that some subsequent store is the next valid instruction to be sent down the pipeline. When this occurs, the same helper behavior as described above can result.

Silent data corruption can also result. If the load instruction was only modified in terms of its virtual address (for example, a register reference was changed from %g1 to %g2), the load data may return for address [%g1] and returned properly to a correctly operating pipeline. But the data may be installed in the incorrect D Cache location (since the D Cache index used is a function of %g2, instead of %g1).

This case will not occur in the case of a single CPU that is executing self-modifying code as long as it follows the rules for doing so in SPARC-V9 or JPS1 manuals.

Workaround:

Do not modify code running on another CPU without explicit synchronization. In the case above, the result of the modification of CPU A's instruction are non-deterministic since there is no way to guarantee that the modification occurs before execution or after. If an instruction must be modified without synchronization, all types of load instructions must be avoided.

An operating system-level work around is to ensure that memory mapped in the Instruction Translation Look-Aside Buffer (ITLB) of one CPU is not mapped in a writable state in the Data Translation Look-Aside Buffers (DTLBs) of other CPUs.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #22: A fdiv or fsqrt instruction followed by a pdist instruction with both source and destination register dependencies on its rd field causes incorrect results.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

The particular code sequence is limited in nature and requires two very unlikely events in addition to the perfect line-up of 3 Floating Point (FP) instructions with the pdist:

- There is a fdiv or fsqrt instruction that has a write-after-write (WAW) hazard with a subsequent FP/graphics operation (e.g., the program issued a divide or square root, and then followed it up with another instruction [e.g., fadd] that overwrote the divide or square root result before anyone could use it).
- There is a second fdiv or sqrt instruction that is followed by a pdist instruction with both a read-after-write (RAW) and WAW hazard with the rd of the pdist. (pdist is a special instruction that not only uses the standard rs1 and rs2 as source operands, but also uses rd as a source operand, does a computation, and then puts the result back in rd.) This sequence is unlikely because the pdist is a graphics instruction that uses the 64-bit integer value in rd to do its computation, often in video compression. In this case, the producer of this integer value is a fpdiv or fsqrt instruction.

The failing sequence looks like this:

```
fdiv/fsqrt{s,d} -> rd1:
[...]
fpop -> rd1
[...]
fdiv/fsqrt{s,d} -> rd2
[...]
pdist rd1, rd?  -> rd2
```

There can be arbitrary instructions interspersed in this sequence and still result in a failure. The important criteria is the relative spacing between the instructions.

Impact:

The WAW hazard before the first fdiv or fsqrt instruction and the subsequent FP or GR operation is used incorrectly as a WAW to affect the state of the later and second fdiv or fsqrt instruction, making it unable to bypass its result to following instructions (here, the pdist). When the pdist shows up, the rd register that it wants to use in its computation appears to be available in the FP register file, instead of being still computed by the fdiv or fsqrt. The pdist executes at the same time as the fdiv or fsrt (which is asynchronous to the pipeline) and uses the old rd from the FPRF incorrectly.

The fdiv or fsqrt and pdist, although issued as above, hypothetically will update the FPRF is any order. Thus, the two symptoms one might see:

1. The pdist result is broken, appearing as if the fdiv or fsqrt didn’t happen.
2. The pdist result appears to be dropped completely, with the fdiv or fsqrt result left in

the rd in question.

The problem has been created before the pdist begins execution and its window of vulnerability extends until the fdiv or fsqrt result is in the register file and is available to use as a source operand.

Workaround:

Avoid the code sequence described in this erratum.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #23: In privileged mode, a store alternate using Address Space Identifier (ASI) 0x64 hangs the processor.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

In privileged mode, a store alternate operation using Address Space Identifier (ASI) 0x64 should be flagged as usage of an illegal ASI and result in a *data_access_exception* trap. However, this checking is not present and the store is allowed to execute. The store, however, is never acknowledged since ASI 0x64 doesn't exist.

Impact:

The machine hangs in privileged mode. User mode behavior is correct, so a malevolent user cannot hang the machine. ASI 0x64 behavior is as follows:

	user mode	privileged mode
ASI LOAD	*privileged_action* trap	*data_access_exception* trap
ASI STORE	*privileged_action* trap	CPU HANG

Workaround:

Privileged code should not issue store alternates to ASI 0x64.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #24: A delay slot involving Delayed Control Transfer Instructions (DCTIs) may not be properly executed, and subsequent execution of certain instructions may result in skipping the original instruction stream and instead executing a different instruction stream.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

One example of a failing code sequence is as follows: The dcti-couple is represented here as back-to-back branches: branch_1 and branch_2. The last instruction, the add in the delay slot of a branch, is dropped on the nth iteration through this code, but the number n is not significant:

```
              ldsb        [%l1], %o6
              bcs,pt      %icc, .-0xC
              fmovs       %f4, %f17
              nop
              fsubs       %f17, %f5, %f17
              be,a,pt     %xcc, .+0x4
              subcc       %o7, 0x2E4, %o7
              fcmps       %fcc0, %f17, %f6

branch_1:  bn,pn       %icc, .+0x0
branch_2:  fbue        .-0x18
d_slot_1:  andn        %o7, %o6, %o7
branch_3:  brlz,a,pt   %o5, .-0x54
d_slot_2:  movre       %i0, -0x0C3, %i3
branch_4:  brlez,pt    %i1, .-0x138
failure:   add         %o0, 0x001, %o0 // add does not execute
```

The delay slot of a mispredicted Delayed Control Transfer Instruction (DCTI) may not be properly executed if the DCTI is last part of a dcti-couple or triple, or follows within several pipeline stages of a dcti-couple instruction pair. Symptoms of the failure may include:

- The execution of a delay slot instruction from an older or a younger DCTI.
- Non-execution of the real delay slot.
- Skipping the instruction stream starting at a subsequent refetched instruction (e.g., a mispredicted or Jump and Link Instruction (JMPL) or return from subroutine (RET) delay slot, recirculating Load Integer Instruction (LD), following FLUSH or certain Write Privileged or Write State Register) or trapping instruction, and instead executing the instruction stream at PC=0x80 greater than the desired instruction.

Spurious increment of the PC by 0x80 may also occur in the value saved by RDPC, TPC or TnPC on trap, or the return address of the CALL or JMPL.

The following conditions are necessary for delay slot failure to occur:

- The DCTI instruction is mispredicted.
- The Delay slot that will not be handled properly reaches I-stage before its DCTI reaches E-stage (delay slot must be in same cache line as DCTI or hit in icache).
- There must be at least two older DCTI instructions in C-stage or earlier that are not JMPL/RET and not in delay slot of an earlier DCTI when mishandled delay slot instruction reaches I-stage.
- There must be an older dcti-couple in C-stage or earlier when the mispredicted DCTI reaches I-stage.
- Older unresolved DCTIs than the one with mishandled delay must all be predicted correctly.

The erroneous PC increment of 0x80 additionally requires:

- The mispredicted branch is actually not taken.
- The PC[6] of DCTIs delay slot and PC[6] of the next sequential instruction are both 0, and PC[6] of the falsely requeued delay slot is 1.

Impact:

A delay slot of a dcti-couple or a DCTI closely following a dcti-couple that reaches issue stage with older unresolved (C-stage or earlier) DCTIs may not be properly executed, and subsequent execution of either a refetched instruction or trapping instructions may result in skipping the original instruction stream and instead executing the instruction stream starting at PC=0x80 greater than the original instruction’s PC.

Workaround:

Avoid DCTI couples, as per the V8 specification of unpredictable results from SPARC V8 manual: If the first instruction of a DCTI couple is a conditional branch, the targets of the DCTI are within the same address space as the DCTI couple, but are otherwise unpredictable. Given the relative rarity of dcti-couples, this problem is not viewed as particularly severe.

Status:

This bug will not be fixed in future releases of the silicon.

Erratum #25: A Read-after-Write address checking failure for load in the delay slot of a mispredicted Delayed Control Transfer Instruction (DCTI) can result in stale Data Cache (D Cache) data.

Applicability:

UltraSPARC-III Versions 3.1, 3.2, 3.3 and 3.4.

Description:

In one special circumstance, the read-after-write (RAW) checking fails for currently executing loads existing stores pending in the Store Queue, resulting in stale data being installed in the Data Cache (D Cache). From a program point of view, it will appear as if a store operation completed normally to External Cache (E Cache) and Memory, but did not update the D Cache.

This situation can only manifest itself when there is a load that has a RAW hazard in the delay slot of a Delayed Control Transfer Instruction (DCTI) (e.g., conditional branch) that is mispredicted. There is a distinction in RAW checking where some loads are bypassable and some are not. A read-after-write load that is bypassable can obtain the data directly from the Store Queue. A read-after-write load that is not bypassable must wait until the youngest store (there can be more than one) that touches the D Cache line in question has left the Store Queue. Here is an example sequence and the conditions that need to be met for this bug to occur.

```
ST A   << Hits in D Cache.
..
ST B   << Miss D Cache
..
..
```

```
BR X   << Must be mis-predicted
LD C   << Same 32 Byte line as B (Non-bypassable RAW)
LD A   << Line A invalidated in D Cache before this LD
        (Bypassable RAW)
..
X:
```

In this case of the branch being mispredicted taken, but actually taken, the execution pipe trace may look like this. When a branch is mispredicted, the delay slot is always cancelled and re-executed (unless it was annulled).

```
BR X    R  E  C  M  W  X  T  D          < Mis-predicted
LD C       R  E (c)(m)(w)(x)(t)(d)      < Cancelled in C
LD A          R  E (c)(m)(w)(x)(t)(d)   < Cancelled in C
..
LD C                      R  E  C  M  W  X
                          < Delay slot requed from MisPred Q
```

The conditions are as follows:

- Branch is mispredicted and LD C and LD A were cancelled due to a misprediction in the C Stage.
- LD A data is raw bypassable from the previous store; the store data has been retired into W Cache and is waiting to update D Cache; and the line A has been invalidated in D Cache.

Impact:

When the above conditions are met, the Store Queue fails to detect the RAW condition from ST B and fetches line C (same line as B) and installs it into the D Cache. The ST B in the Store Queue is not aware of this action since it was a D Cache miss when it was entered onto the Store Queue (and, if working properly, a store that misses the D Cache should never become a D Cache hit). When Store B exits the Store Queue, it only updates the data in W Cache, not the line in the D Cache.

The branch is mispredicted and not-taken.

```
       0x5f16c554 : ld     [%i1 + 0xa74], %o3
       0x5f16c558 : sth    %l6, [%g2 + 0x3c]
       0x5f16c55c : st     %l3, [%g2 + 4]
A      0x5f16c560 : st     %f20, [%g2 + 0x14]
                                 <<< RAW bypasable Store (E).
```

```
B    0x5f16c564 : sth    %o5, [%i1 - 0x300]
C    0x5f16c568 : st     %f20, [%g2 + 0x2c] <<< RAW Hazard (D)
     0x5f16c56c : addcc  %l1, %l0, %l6
     0x5f16c570 : stb    %o4, [%g2 + 0x27] <<< RAW Hazard (D)
     0x5f16c574 : addcc  %l4, %l0, %o1
     0x5f16c578 : sth    %l7, [%g2 + 0x26] <<< RAW Hazard (D)
     0x5f16c57c : st     %f20, [%g2 + 0x3c] <<< RAW Hazard (D)
Br   0x5f16c580 : be     0x5f16c588 <<< MISPRED,NOT-TAKEN
D    0x5f16c584 : ldsb   [%g2 + 0x37], %l7 <<< RAW check fail
E    0x5f16c588 : ld     [%g2 + 0x14], %f21 /
     0x5f16c590 : andcc  %l1, %l4, %l3
```

Workaround:

Turn off the RAW bypass enable (DCU.RE) bit in D-Cache Control Register (ASI=0x45, VA=0x0). This situation will not occur if RAW bypassing is disabled.

Status:

This bug will not be fixed in future releases of the silicon.