



UltraSPARC-III+/III++ Errata

Part No. 820-4007-10

12/12/07

Sun Microsystems, Inc.
www.sun.com

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

Copyright 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, docs.sun.com, Sun Blade, SunVTS, SunSolve, SunService, Sun Fire, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and in other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054, États-Unis. Tous droits réservés.

Sun Microsystems, Inc. possède les droits de propriété intellectuels relatifs à la technologie décrite dans ce document. En particulier, et sans limitation, ces droits de propriété intellectuels peuvent inclure un ou plusieurs des brevets américains listés sur le site <http://www.sun.com/patents>, un ou les plusieurs brevets supplémentaires ainsi que les demandes de brevet en attente aux les États-Unis et dans d'autres pays.

Ce document et le produit auquel il se rapporte sont protégés par un copyright et distribués sous licences, celles-ci en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Tout logiciel tiers, sa technologie relative aux polices de caractères, comprise, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit peuvent dériver des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays, licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, docs.sun.com, Sun Blade, SunVTS, SunSolve, SunService, Sun Fire, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface utilisateur graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox dans la recherche et le développement du concept des interfaces utilisateur visuelles ou graphiques pour l'industrie informatique. Sun détient une licence non exclusive de Xerox sur l'interface utilisateur graphique Xerox, cette licence couvrant également les licenciés de Sun implémentant les interfaces utilisateur graphiques OPEN LOOK et se conforment en outre aux licences écrites de Sun.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DÉCLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES DANS LA LIMITE DE LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE À LA QUALITÉ MARCHANDE, À L'APTITUDE À UNE UTILISATION PARTICULIÈRE OU À L'ABSENCE DE CONTREFAÇON.



Adobe PostScript



1. Errata Table

Table 1: UltraSPARC-III+/UltraSPARC-III++ Errata Table

Errata Number	UltraSPARC-III+			UltraSPARC-III++			See ...
	Version 2.1	Version 2.2	Version 2.3	Version 1.0	Version 1.0.1	Version 1.1	
1	✓	✓	✓	✓	✓	✓	page 4
2	✓	✓	✓	✓	✓	✓	page 5
3	✓	✓	Not applicable	Not applicable	Not applicable	Not applicable	page 5
4	✓	✓	✓	Not applicable	Not applicable	Not applicable	page 7
5	✓	✓	✓	✓	✓	✓	page 8
6	✓	✓	✓	✓	✓	Not applicable	page 10
7	✓	✓	✓	✓	✓	✓	page 12
8	✓	✓	✓	✓	✓	✓	page 13
9	✓	✓	✓	✓	✓	✓	page 15
10	✓	✓	✓	✓	✓	✓	page 17



2. Errata Descriptions and Workarounds

Erratum #1: A diagnostic read of the fully associative Translation Table Entry (TLB) may return incorrect data.

Applicability:

UltraSPARC-III+ Versions 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

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 (Idxa 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 (Idxa 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 #2: A simultaneous foreign system bus write of IO for store (WIO) and Address Space Identifier (ASI) access to the Memory Controller Unit (MCU) control registers is not allowed.

Applicability:

UltraSPARC-III+ Versions 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

Description:

When an outgoing transaction buffer in the Transaction Out Buffer (TOB) is full, it can not accept any outgoing interrupt. If the Coherent Pending Queue (CPQ) is processing a foreign system bus write of IO for store (WIO) to write one of the Memory Controller Unit (MCU) control registers at the same time, data contention occurs between Address Space Identifier (ASI) write data and foreign WIO data.

Impact:

Incorrect data is written into the MCU control registers.

Workaround:

Issue a foreign WIO request only when the processor is in the quiescent state without any outstanding outgoing request.

Status:

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

Erratum #3: A Block store can cause the system to hang when the CPU is running with a 7:1 (1,050 MHz) or 8:1 (1,200 MHz) Fireplane clock ratio.

Applicability:

UltraSPARC-III+ Versions 2.1 and 2.2.

**Description:**

Block stores write 64-byte chunks of data to memory. Typically, the affected memory locations are not resident in the D Cache, so the Block store is optimized assuming a D Cache miss on both 32-byte halves (i.e., two D Cache lines) of the 64-byte chunk.

Data is pushed out of the 8-entry Store Queue quickly to free up space for subsequent store operations, thereby increasing store throughput. The dispatching unit is informed that the Store Queue is now empty through a special interface indication (i.e., `stq_clear`).

In the case that both 32-byte chunks miss the D Cache, the optimization is enabled; otherwise, 8 bytes at a time are moved to the D Cache for the line or lines that hit.

In the 7:1 and 8:1 clock ratio modes, D Cache updates can be inadvertently required because of the unintentional reporting of a D Cache hit due to a 1-cycle conflict on the dual-ported SRAM array that stores the valid bits for the D Cache.

- In 7:1 (1,050 MHz), this occurs the downstream memory system is attempting to invalidate unrelated locations in the D Cache to maintain cache coherency at the same time as the Block store is interrogating the D Cache.
- In 8:1 (1,200 MHz), there is the added condition of a D Cache fill occurring virtually at the same time as well. Each cycle where hit is incorrectly reported requires a D Cache update for the corresponding 8-bytes of data (i.e., `stq_decrement`).

In the problem scenario, there is a conflict between the special `stq_clear` mechanism and the normal `stq_decrement` logic, where both are activated for the same block store. In the failing cases, the `stq_clear` operation informs the dispatching unit that the Store Queue is empty. A subsequent `stq_decrement` operation then forces the counter that keeps track of the number of outstanding stores into an unrecoverable underflow condition.

Impact:

A subsequent operation that looks for an empty Store Queue (e.g., `membar #Sync`) will wait forever because the Store Queue is in an undefined state. Besides hanging the machine, stores that appear to have been completed may in fact be totally dropped.

**Workaround:**

Before executing a Block Store, make sure a 32-byte chunk at the 64-byte store address is resident in D Cache. This prevents the conflict of the `stq_clear` and `stq_decrement` operations.

Status:

This bug is fixed in Version 2.3 of the processor.

Erratum #4: A Block Store Commit to a line in write cache followed by a store byte, half_word, word, or doubleword to the same 64-byte line can cause data corruption.**Applicability:**

UltraSPARC-III+ Versions 2.1, 2.2, and 2.3.

Description:

Data corruption can occur when a Block Store Commit (BST) to Address A is followed by a store to any of the 64 bytes at address A without a Memory Barrier Instruction (MEMBAR) (as explicitly required by programming rules):

```
BST [A]          rather than:  BST [A]
ST [A]           membar #Sync
                 ST [A]
```

Such a code sequence is inherently nonsensical as without the `membar #Sync`, there is no guaranteed order between the two operations; thus, the data at [A] is indeterminant.

The additional requirements for this problem to occur are:

- First or second 32 bytes of the BST commit hits D Cache.
- All four Write Cache (W\$) lines at the index referenced by [A] are valid and in M (Modified) state with different 64-byte blocks (i.e., [A] is not resident in write cache).
- The movement of the Block store data towards the Fireplane Bus outgoing buffer is delayed in hardware. This condition allows the store to arrive at the W Cache while the BST commit is still being processed.

**Impact:**

The data referenced by the ST [A] operation is put into one of the four unrelated W Cache entries, thus corrupting the data at that address (call it [X]) as well as dropping the update to [A].

Workaround:

Use a membar #Sync between the Block store and store.

Status:

This bug has been fixed in UltraSPARC-III++ Version 1.0.

Erratum #5: 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 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

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 is 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 #6: 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 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0 and 1.0.1.

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 in 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 #7: In privileged mode, a store alternate using Address Space Identifier (ASI) 0x64 hangs the processor.

Applicability:

UltraSPARC-III+ Versions 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

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 #8: 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 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

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 #9: 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 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

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 reqeud 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 bypassable Store (E) .
B 0x5f16c564 : sth     %o5, [%i1 - 0x300]
C 0x5f16c568 : st      %f20, [%g2 + 0x2c] <<< RAW Haz ( D )
0x5f16c56c : addcc  %l1, %l0, %l6
0x5f16c570 : stb     %o4, [%g2 + 0x27] <<< RAW Haz ( D )
0x5f16c574 : addcc  %l4, %l0, %o1
0x5f16c578 : sth     %l7, [%g2 + 0x26] <<< RAW Haz ( D )

```



```
0x5f16c57c : st      %f20, [%g2 + 0x3c] <<< RAW Haz ( D )
Br 0x5f16c580 : be      0x5f16c588 <<< MISPRED,NOT-TAKEN
D  0x5f16c584 : ldsb   [%g2 + 0x37], %17 <<< RAW check fail
E  0x5f16c588 : ld      [%g2 + 0x14], %f21 /
0x5f16c590 : andcc   %11, %14, %13
```

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.

Erratum #10: Installing an instruction memory management unit (IMMU) entry in a fully associative ITLB without processing an Instruction translation look-aside buffer (ITLB) miss may displace the locked entry 0.**Applicability:**

UltraSPARC-III+ Versions 2.1, 2.2, and 2.3.
UltraSPARC-III++ Versions 1.0, 1.0.1, and 1.1.

Description:

Two TLBs make up the instruction memory management unit (IMMU):

- A 128-entry 2-way set associative TLB (T8) that maps unlocked 8KB pages.
- A 16-entry fully associative TLB (T16) that maps locked and unlocked 64KB, 512KB, and 4MB pages, as well as locked 8KB pages.

Locked pages are those that the operating system decides should never be displaced and carry a special Lock bit in each entry to prevent that entry from being considered as part of the normal replacement policy.

Each entry has a Used bit associated with it, which is set when the associated TLB entry has been used for a successful translation.

If the replacement is directed to the T16, there are the following alternatives:

1. The first invalid entry is replaced (measuring, or starting, from entry 0 sequentially



through entry 15). If there is no invalid entry, then:

2. The first unused, unlocked (Used = 0; Lock = 0) entry will be replaced (measuring, or starting, from entry 0 sequentially through entry 15). If there is no unused, unlocked entry, then:
3. All used bits are reset, and the process is repeated from 2.

This description is only true if each replacement of a T16 entry is preceded by a single Instruction translation look-aside buffer (ITLB) miss. The reason this is the case is that the clearing of the Used bits in step 3 above occurs at the instant that an ITLB miss has been detected, not at the time when all entries have either their Locked or Used bits set.

Impact:

A real failing scenario is as shown in the following table.

Table 2: Comparison of T16 replacement scenarios

TLB Entry	Valid	Used	Lock	Expected Replace	Actual Replace
Entry 0	1	0	1	0	2nd (All entries either Locked or Used)
Entry 1	1	1	0	2nd (All Used set to 0)	0
Entry 2	1	1	0	0	0
Entry 3	1	1	0	0	0
Entry 4	1	1	0	0	0
Entry 5	1	1	0	0	0
Entry 6	1	1	0	0	0
Entry 7	1	1	0	0	0
Entry 8	1	1	0	0	0
Entry 9	1	1	0	0	0
Entry 10	1	1	0	0	0
Entry 11	1	1	0	0	0
Entry 12	1	1	0	0	0



Table 2: Comparison of T16 replacement scenarios (Continued)

TLB Entry	Valid	Used	Lock	Expected Replace	Actual Replace
Entry 13	1	1	0	0	0
Entry 14	1	0	0	1st (Used = 0)	1st (Used = 0)
Entry 15	1	1	1	0	0

The table shows the state of the T16 when an ITLB miss is detected. Software attempted to perform an insertion of two locked pages without an intervening miss. There is one entry, Entry 14, that has neither the Lock or Used bit set, so it is used to hold the first of the new locked pages. Additionally, this scenario also prevents the Used bits from being reset. After Entry 14 is created with the first of the locked pages, all entries are now Locked or Used. However, since there is not a second ITLB miss as the triggering event, the Used bits are not reset to make room for the 2nd locked page.

When the 2nd replacement is attempted, the hardware looks for 1 of the 16 entries with either Lock or Used bits deasserted to create a 4-bit index with which to store a new entry. Since there is no such entry, the 4-bit index created is "0000". This results in entry 0 being selected, even though it is Locked, as the page to be replaced.

Workaround:

In situations where new entries are installed without a corresponding ITLB miss, the operating system should check whether all entries are used or locked beforehand.

Another approach is to guarantee that entry 0 is only allowed to hold an unlocked entry. After an entry has been installed without a corresponding ITLB miss, check entry 0's Lock bit.

- If it is not set, there is no problem.
- If it is set, the newly inserted entry needs to be moved to another location. If the entry is not moved, the problem may occur later in the program.

Status:

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