MPC7450, MPC7451, MPC7441 Chip Errata

Supports:

MPC7451
MPC7450
MPC7441

This document details all known silicon errata for the MPC7451, MPC7450, and MPC7441. The MPC7450 has the same functionality as the MPC7451 and MPC7441, and any differences between the other microprocessors are noted in the document. The MPC7451, MPC7450, and MPC7441 are PowerPC™ microprocessors. Table 1 provides a revision history for this chip errata document.

Table 1. Document Revision History

Revision Number	Date	Significant Changes
0–4	—	Earlier releases of document.
5	—	Errata 37 and 47: Clarified information.
		Errata 46: Added $\overline{\text{TEA}}$ to affected signals.
6	—	Added errors 49–53.
		Revised Errata 36, including workaround.
7	—	Revised workaround for Errata 40.
		Added Errata 54.
8	—	Revised Errata 42, including workaround and projected solution; error not fixed for instruction fetches in MPC7450, Rev. 2.3 and beyond.
		Added Errata 55 and 56.
9	—	Added Errata 57.
10	—	Added Errata 58 and 59. Updated Table 3 to match errata.

Table 1. Document Revision History (continued)

Revision Number	Date	Significant Changes
11	—	Added Errata 60.
12	—	Added Errata 61–64.
13	—	Erratas 16 and 18: Corrected entries in Table 3; these errata are present in Rev. 1.x devices.
		Errata 49, 52, 55: Changed Projected Solution to none. (This was previously reported as having been changed in Rev. 12 of this document, but was not changed until this revision.)
		Errata 55: This erratum applies only to **dcba** instructions to addresses using Direct-Store Segment Address Translation (T = 1).
		Errata 60: Significantly revised Description, Projected Impact, and Workaround, error causes imprecise result, not an incorrect one.
		Errata 62: Revised and added workaround.
		Errata 63: Corrected error in workaround: Changed 'Do execute...' to 'Do not execute...'
14	—	Added Errata 65 and 66. Did additional reformatting and edit.
14.1	—	Nontechnical reformatting
15	—	Added Errata 67 and 68.
16	05/26/04	Added Errata 69 and 70.
17	—	Added Errata 71 and 72.
18	11/09/04	Revised Errata 46 and Errata 72. Reformatted document.
		Added Errata 73 and Errata 74.
19	05/19/05	Added Error 77.
20	09/07/05	Error 77: added "or 60x bus" to the first sentence in "Description."
		Error 58: updated "Workaround."

Table 2 describes the devices to which the errata in this document apply and provides a cross-reference to match the revision code in the processor version register to the revision level marked on the part.

Table 2. Revision Level to Part Marking Cross-Reference

MPC7450 Revision	MPC7451 Revision	MPC7441 Revision	Part Marking	Processor Version Register
1.0	—	—	N/A	8000 0100
1.1	—	—	N/A	8000 0101
1.2	—	—	N/A	8000 0102
2.0*x*	—	—	A, B, C, D	8000 0200
2.1	—	2.1	E	8000 0201
—	2.3	2.3	G	8000 0203

Table 3 summarizes all known errata and lists the corresponding silicon revision level to which it applies. A 'Y' entry indicates the erratum applies to a particular revision level, while a 'N' entry means it does not apply.

Table 3. Summary of Silicon Errata and Applicable Revision

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
1	AltiVec floating-point instruction failures	Some of the AltiVec floating-point instructions will fail.	Y	N	N	N	N	N
2	Incorrect bus address parity error signaled	Systems that enable bus address parity will be affected.	Y	N	N	N	N	N
3	Back-to-back push deadlock	Systems that allow modified data in L1 and perform address sharing are affected.	Y	N	N	N	N	N
4	External interrupt and FP exception cause hang	Any application running with FP interrupts enabled (MSR[FE0] = 1 or MSR[FE1] = 1) and with external interrupts enabled (MSR[EE] = 1) is affected.	Y	N	N	N	N	N
5	Incorrect data forwarded from data bus	Any 60x or MPX bus system that does not pulse $\overline{\text{DBG}}$ only during the cycle in which it will be qualified may show this problem. The MPC7450 may receive reload data or enter a hang state.	Y	N	N	N	N	N
6	Speculative guarded touches are permitted to go out to the bus	Systems that run in real mode or allow translation to non-existent addresses can be affected.	Y	N	N	N	N	N
7	System hang on tlbsync	Any processor executing such that a code loop may hang, although it is believed to be a rare case, is affected.	Y	N	N	N	N	N
8	System hang during hardware tablewalk	Any processor that has a speculative tlbld issued while a hardware tablewalk is in progress will cause this hang.	Y	N	N	N	N	N
9	FPU hangs the processor	The system hangs when this condition occurs.	Y	Y	N	N	N	N
10	External interrupt and system management interrupt are edge triggered	The second interrupt of a given type will not be taken. This may cause the system to time out due to the second interrupt not being taken.	Y	Y	N	N	N	N
11	Instructions complete after MSR[EE] before external interrupt	Code immediately after the mtmsr instruction that sets the MSR[EE] will get executed prematurely.	Y	Y	Y	N	N	N
12	Back-to-back store to overlapping address causes lost data	Loss of store data.	Y	Y	N	N	N	N
13	'Infinite' code loop in MP system causes hang	This problem will only occur in MP systems. The code sequence is not believed to be used in general practice.	Y	Y	Y	Y	N	N
14	Stalled touch hang	The system will hangs if a guarded touch that stalls in the LSM is hit and the next instruction is a store.	Y	Y	Y	N	N	N

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
15	Mismatched **stwcx**. fails to report that store was performed	If this scenario occurs, the application may see a loss of atomic coherency.	Y	Y	Y	Y	N	N
16	L3 hardware invalidate causes erroneous machine check exception	Systems will take unexpected machine check exceptions from an L3 hardware invalidate.	Y	Y	Y	Y	N	N
17	L3 private memory mode not functional when L3 cache disabled	Systems using the L3 as private memory only will lose data written to the L3 SRAMs and receive unknown values for reads from the L3 SRAMs.	N	N	Y	Y	N	N
18	Processor will not enter nap mode after taking an interrupt	Systems that use power management and cannot use the work-around will not see a power savings.	Y	Y	Y	Y	N	N
19	Setting MSSCR0[18–24] will cause machine check exceptions	Software that sets MSSCR0[18–24] bits will cause the system to take spurious and sometimes frequent machine check exceptions.	These bits have been marked '*Reserved for factory use only*' in the *MPC7450 RISC Microprocessor Family User's Manual*.					
20	L3 cache is initialized incorrectly	The L3 interface will not operate correctly.	The workaround is documented in the *MPC7450 RISC Microprocessor Family User's Manual*.					
21	Hardware prefetches ignore L3CR[IONLY]/L3CR[DONLY]	Software that uses L3CR[DONLY] and/or L3CR[IONLY] to prevent the allocation of data or instructions into the L3 will not logically fail but will have degraded performance if hardware prefetches are enabled.	N	N	N	Y	N	N
22	Hardware prefetching impacts dcbz-intensive M = 1 software loops	Software using dcbz-intensive M = 1 loops to load and clear memory will be impacted.	N	N	N	Y	N	N
23	dcbf/dcbst-initiated castout sequence retries external snoop forever	The scenario, if hit, will cause the system to hang. The only remedy is to assert $\overline{\text{HRESET}}$.	Y	Y	Y	Y	N	N
24	Load-store address collisions cause performance loss	Poor floating-point performance will result from using the default LDSTCR values.	N	N	N	Y	N	N
25	TAU reports incorrect temperatures	Programmed trip temperatures will not trigger output interrupts, even if temperatures exceed the expected setpoint by up to 55 degrees.	The thermal assist unit is not supported as a feature and has been removed from all MPC7450 family documentation except this errata sheet.					
26	MPC7450 L3 output pin voltage modes set incorrectly	Systems that have different bus and L3 voltages may see some performance issues with the L3 bus due to different output impedance of the affected pins.	Y	Y	Y	Y	N	N
27	Toggling the TBEN input signal causes unexpected behavior	Systems that toggle the TBEN input signal every cycle will be affected.	Y	Y	Y	N	N	N

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
28	Line reserved in L2 due to dcbst/dcbtst pair	Software using dcbst and dcbtst instructions may experience a gradual degradation of effective L2 size over time.	Y	Y	Y	Y	N	N
29	IMMU speculative tablewalk causes bad instruction fetch	Systems not able to implement the suggested workaround code may see the MPC7450 perform instruction fetches from unintended addresses.	Y	Y	Y	Y	N	N
30	L2 hardware flush may not flush every line from the L2 cache	Data corruption will occur in systems whose software misses in the iL1 during an L2 hardware flush routine.	The workaround is documented in the *MPC7450 RISC Microprocessor Family User's Manual*.					
31	BTIC corruption	The processor may get the wrong answer or hang if BTIC corruption occurs.	Y	Y	Y	Y	N	N
32	IMMU page fault/tlbie collision causes hang	Hangs in multiprocessor systems.	Y	Y	Y	Y	N	N
33	Software tablewalks may corrupt TLB	Systems attempting to use software tablewalks will need to use the workaround.	Y	Y	Y	Y	N	N
34	L1_TSTCLK must be pulled low	Tying L1_TSTCLK high may cause data corruption in the L2 and may limit the frequency of operation.	The workaround has been documented in the revised hardware specifications as correct way to configure L1_TSTCLK and L2_TSTCLK.					
35	Snoop responses are not generated in PLL-bypass mode	The processor will hang and cause memory inconsistencies if it is required to generate a snoop response while operating in PLL-bypass mode. PLL-bypass mode can only be used in systems in which the MPC7450 will not be required to snoop.	Y	Y	Y	Y	Y	Y
36	Bus store queue causes hang in 60x and MPX bus modes	Systems with arbitration policies or queue resources that impose a requirement that a retried load be completed before a subsequent store request may hang.	Y	Y	Y	Y	Y	N
37	Dropped instruction fetch when disabling instruction address translation	Systems where this occurs can see processes die (ISI or DSI) and may see memory corruption. The likelihood of this occurring is much higher in multiprocessor systems than in uniprocessor systems.	Y	Y	Y	Y	Y	N
38	TLB corruption caused by out-of-order I-cache reload and tlbie	This bug can only occur in multiprocessor systems. These systems may see problems including ISI, DSI, or memory corruption.	Y	Y	Y	Y	Y	N
39	Use of power management modes causes L3 read errors	L3 read errors can cause data corruption or illegal instructions.	Y	Y	Y	Y	Y	N

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
40	L3 private memory data corruption if L3 cache disabled	If an internal collision occurs and an unexpected external bus access is created, data corruption to a L3 private memory space or a bus error may result.	Y	Y	Y	Y	Y	N
41	Multiple bus requests asserted during window-of- opportunity	Multiprocessor systems that do not let the MPC7540 perform a push after the window-of- opportunity has been requested will see the processor violate the bus specification in the way explained above.	Y	Y	Y	Y	Y	N
42	TLBMISS register does not contain page index information	Debuggers in systems using software tablewalks have no easy way to determine the address of the exception. Watchpoints or breakpoints for loads and/or stores will not operate correctly.	Y	Y	Y	Y	Y	Y
43	Address parity checked during snooped M = 0 operations	Systems with masters that create M = 0 traffic and do not drive correct address parity on the MPX/60x bus will cause the MPC7450 to take unexpected checkstops or machine check exceptions if HID1[EBA] is set.	Y	Y	Y	Y	Y	N
44	Push request may hang when retried during doze state	Systems that permit snooping in doze state and do not perform fair arbitration after a retried window-of-opportunity request may experience hangs.	Y	Y	Y	Y	Y	N
45	PTE changed (C) bit corruption when data L1 cache disabled	A C bit may be corrupted in systems that keep the dL1 disabled and the L2 and/or L3 caches enabled. Since this may be a transient state for systems flushing the cache hierarchy, L2 hardware prefetching should be disabled when flushing the caches.	N	N	N	Y	Y	Y
46	Earliest transfer of MPX/60x data requirement	Any system where a device may assert TA or TEA for a data tenure before or while the AACK for that transaction is driven.	The MPC7450 family implementation of the earliest transfer for MPX/60x data will remain as documented.					
47	dcbz collision with external snoop may cause dcbz to hang	Devices that can master a kill-type operation followed by a second address streamed snoop to the same address in MPX bus mode may hang.	Y	Y	Y	Y	Y	N
48	Snooped kill followed by snoop to same address hangs processor	Devices that can master a kill-type operation followed by a second address streamed snoop to the same address in MPX bus mode may hang.	Y	Y	Y	Y	Y	N
49	Missed snoop transaction due to ignored QACK deassertion during transition to low power mode	Systems using the L3 interface and power management modes may experience memory corruption or hangs.	N	N	N	N	N	Y

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
50	AltiVec lvxl/stvxl instructions allocate lines in the L2/L3 caches	A loss of performance may occur since memory accessed by lvxl/stvxl instructions that is known to have little re-use will be allocated in the L2/L3 caches of the MPC7455, taking cache entries from other data which may be more frequently accessed.	Y	Y	Y	Y	Y	Y
51	tlbld instruction can cause incorrect instruction translation	Systems using tlbld instructions may cause the hardware tablewalk engine to return an incorrect result for an instruction tablewalk.	Y	Y	Y	Y	Y	Y
52	Variable size memory accesses via COP cannot be performed using the service bus	Emulators and debug tools will experience reduced performance while performing operations that require variable size memory accesses.	Y	Y	Y	Y	Y	Y
53	L2 hardware prefetching occurs for transient requests	Software expecting the alternate sector of a transient address to also be transient and not be allocated in the L2 and L3 caches may see a performance loss.	Y	Y	Y	Y	Y	Y
54	L2/L3 data parity error forces checkstop instead of machine check	Because the MPC7455 will checkstop, systems that enable L2 and/or L3 data parity checking will not be able to recover from these exceptions.	Y	Y	Y	Y	Y	Y
55	Instructions following a dcba mapped T = 1 may not be executed	Any code that executes dcba instructions with T = 1 effective addresses may not see all instructions executed. Since Direct-Store Segment Address Translation was originally included in the architecture to support legacy POWER architecture I/O devices that used this interface, the impact to customers should be minimal.	Y	Y	Y	Y	Y	Y
56	$\overline{\text{CKSTP\_OUT}}$ asserted incorrectly and output signals not disabled	The assertion of the $\overline{\text{CKSTP\_OUT}}$ signal may not be recognized by external logic, especially if the logic is synchronous or not edge-sensitive to changes in this signal.	Y	Y	Y	Y	Y	Y
57	L2 hardware prefetcher not quiesced before QREQ asserted	The issues resulting from the pending prefetches are dependent on how a particular system controller handles $\overline{\text{BR}}$ and/or $\overline{\text{TS}}$ assertions after $\overline{\text{QREQ}}$ has been asserted. If $\overline{\text{QACK}}$ is asserted while $\overline{\text{BR}}$ is asserted, $\overline{\text{BR}}$ may be held asserted until the processor is awakened, potentially confusing the system bus arbiter. If the bus arbiter issues a bus grant to the processor in response to the assertion of $\overline{\text{BR}}$ after $\overline{\text{QREQ}}$ is asserted, the transaction must be allowed to complete.	Y	Y	Y	Y	Y	Y

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
58	$\overline{\text{MCP}}$ signal assertion with MSR[ME] = 1 does not set SRR1[12]	Systems using the external $\overline{\text{MCP}}$ signal and utilizing software that expects SRR1[12] to be set as a result of the assertion of the $\overline{\text{MCP}}$ signal will not function correctly.	Y	Y	Y	Y	Y	Y
59	A tlbli instruction may cause an incorrect instruction translation	If system software does not use any tlbli instructions, then the software can avoid this erratum by guaranteeing that no tlbli instruction will be encountered on a speculative non-taken branch path. Note that 'encountering' a tlbli instruction includes either data, code, or uninitialized memory that decodes to a tlbli instruction. It is, therefore, possible to experience errors as a result of this erratum in any system using hardware tablewalks, even if neither software tablewalks nor tlbli instructions are ever used.	Y	Y	Y	Y	Y	Y
60	AltiVec™ vmaddfp or vnmsubfp can incorrectly return zero	Applications using either the vmaddfp or vnmsubfp instructions will only generate 45 bits of precision for the intermediate (A · C) product in some cases.	Y	Y	Y	Y	Y	Y
61	Back-to-back L2 allocation causes lost data	The alignment of internal arbitration timings required to cause a failure is extremely rare but possible, and has occurred in a system.	Y	Y	Y	Y	Y	Y
62	Six outstanding miss requests may stall the processor	Systems will typically only enter this mode if the system bus becomes congested with store traffic. The stall condition is not a permanent hang state unless snoops never occur once the condition is encountered.	Y	Y	Y	Y	Y	Y
63	dcbt or dcbtst to protected space may not no-op	Systems executing dcbt or dcbtst instructions with addresses that map to protected space using the DBATs or the DTLB may see accesses to undesired addresses on the external bus.	Y	Y	Y	Y	Y	Y
64	PMC2[32] does not increment correctly	Performance analysis using PMC2[32] will receive low counts equal to the number of cycles in which the number of valid completion queue entries was either 8 or 16.	Y	Y	Y	Y	Y	Y
65	Multi-cycle TS assertion in COP soft-stop debug mode	Emulators that utilize the COP soft-stop feature may not work as intended since the processor's violation of the system bus protocol may cause unexpected behavior on the part of the system controller.	Y	Y	Y	Y	Y	Y
66	Snoop during L2 hardware flush can lose modified data	Systems with snooping activity while a processor is performing an L2 hardware flush may see data corruption.	Y	Y	Y	Y	Y	Y

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision					
			1.0	1.1	1.2	2.0	2.1	2.3
67	Processor does not correctly single step lmw/stmw/lsw/stsw instructions in COP debug mode	Emulators that utilize the COP soft-stop feature may not work as intended when attempting to single step a multiple or string word operation.	Y	Y	Y	Y	Y	Y
68	SRR0/SRR1/PC values incorrectly updated if instruction at breakpoint in COP debug mode causes an exception	Emulators that utilize the COP soft-stop feature may not work as intended when attempting to set a breakpoint at an instruction that causes an exception.	Y	Y	Y	Y	Y	Y
69	Data corruption with M=0 and L2 prefetching enabled.	Systems with software mapping memory as non-coherent and enabling L2 hardware prefetching may see loss of data.	Y	Y	Y	Y	Y	Y
70	Cacheable load/store during L2 hardware flush can lose modified data	If the recommended code loop for flushing the L2 cache is used, it keeps the cacheable loads or stores in a sequential code stream from requesting access to the L2 during the flush. However, an interrupt or exception taken during the L2 hardware flush may have an interrupt handler that makes a data miss request. If the recommended L2 hardware flush routine is not used, or a system encounters interrupts during this routine, data corruption could occur.	Y	Y	Y	Y	Y	Y
71	Data parity errors not checked during L2 hardware flush	Any L2 data cache lines with an incorrect bit that would normally be flagged as either a machine check exception or a checkstop will be passed to the L3 or external bus during an L2 hardware flush. L2 parity errors are not expected to occur during normal system operation.	Y	Y	Y	Y	Y	Y
72	Processor may hang when mtmsr/isync transitions MSR[IR] from 1->0 and isync instruction resides in unmapped page	Any systems that disable instruction translation using mtmsr, and for which the required isync may reside in a different page whose page table entry is not available in the memory hierarchy may hang.	Y	Y	Y	Y	Y	Y
73	Scanning MSS_NRM chain via COP during softstop may cause unexpected interrupts upon resumption of normal operation	Emulators and debug tools accessing the MSS_NRM scan chain via COP during softstop.	Y	Y	Y	Y	Y	Y
74	tlbie snoop during Doze state may cause processor hang	Systems performing tlbie snoops during Doze state where the index of the tlbie matches that of the instruction code that caused entry into Nap mode may hang.	Y	Y	Y	Y	Y	Y
77	Unexpected instruction fetch may occur when transitioning MSR[IR] 0->1	Systems executing code as described above may encounter unexpected machine checks or system hangs.	Y	Y	Y	Y	Y	Y

Errata No. 1: AltiVec floating-point instruction failures

Description:

Selected AltiVec floating-point instructions will fail.

Mux-selects 1–3 in a set of dynamic muxes that select appropriate ‘B-Shift’ amounts in the AltiVec floating-point pipeline are shorted and will cause incorrect instruction results for selected AltiVec floating-point instructions.

Projected Impact:

Some of the AltiVec floating-point instructions will fail.

The list of instructions that fail are:

vaddfp

vctsxs

vctuxs

vexptefp

vrfim

vrfin

vrfip

vrfiz

vsubfp

Workaround:

None

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 2: Incorrect bus address parity error signaled

Description:

This erratum affects only the system bus, not the L3 bus. If bus address parity checking is enabled during a snooped address tenure, a false parity error will be reported.

If bus address parity checking is enabled during the second cycle of a snooped address tenure for which address parity checking was disabled for the first cycle of the address tenure, a bus address parity error will be signaled.

Projected Impact:

Systems that enable bus address parity will be affected.

Workarounds:

Either of the following will serve as a workaround to this erratum:

1. Disable bus address parity checking in HID1.
2. Only enable bus parity checking while the system is not presenting address tenures to be snooped.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 3: Back-to-back push deadlock

Description:

Back-to-back snoops may cause system hang.

If a back-to-back external snoop and an internal snoop hit on an address in the L1 castout queue, the second snoop will not detect the collision and will erroneously allocate a push buffer, causing a hang.

Projected Impact:

Systems that allow modified data in L1 and perform address sharing are affected.

Workaround:

Lock all lines in L1 cache using HID0:DLOCK to force write-through mode of the L1 data cache.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 4: External interrupt and FP exception cause hang

Description:

If an FP exception (overflow, underflow, etc.) occurs at the same time as any of the asynchronous interrupts ($\overline{\text{INT}}$, $\overline{\text{SMI}}$, decrementer, thermal, or performance monitor), a hang may occur.

Projected Impact:

Any application running with FP interrupts enabled (MSR[FE0] = 1 or MSR[FE1] = 1) and with external interrupts enabled (MSR[EE] = 1) is affected.

Workaround:

Software must set MSR[FE0,FE1] to 0,0, or no FP instructions that cause exception may be run.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 5: Incorrect data forwarded from data bus

Description:

A parked $\overline{\text{DBG}}$ in 60x mode or a parked or early $\overline{\text{DBG}}$ in MPX bus mode may cause incorrect data streaming behavior resulting in either bad reload data or a hung machine.

If $\overline{\text{DBG}}$ is asserted during the time between the final two $\overline{\text{TA}}$ signals of a read or write data tenure where there is at least one dead cycle between the $\overline{\text{TA}}$ signals, and an operation of the same type (read or write) also has a data tenure outstanding, the MPC7450 will erroneously qualify the $\overline{\text{DBG}}$ and begin the second operation early. The first operation will never receive its last beat of data. Meanwhile, the second operation will be forwarded the last beat of the first operation’s data as its first beat.

In addition, the MPC7450 erroneously tries to stream data in 60x mode.

Projected Impact:

Any 60x or MPX bus system that does not pulse $\overline{\text{DBG}}$ only during the cycle in which it will be qualified may show this problem. The MPC7450 may receive reload data or enter a hang state.

Workaround:

Do not park the data bus. Only pulse $\overline{\text{DBG}}$ during the cycle of a previous data tenure’s TA if streaming is desired. A more severe, but temporary workaround is to not let a new address tenure begin before the previous transactions data tenure has completed.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 6: Speculative guarded touches are permitted to go out to the bus

Description:

Speculative guarded touches are permitted to go out to the MSS, and therefore, the bus.

Speculative guarded touches are supposed to be no-oped, but are not. If a speculative guarded touch to a non-existent address reaches the system bus, it will cause a $\overline{\text{TEA}}$ and machine check.

Projected Impact:

Systems that run in real mode or allow translation to non-existent addresses can be affected.

Workaround:

No-op all touches and DSTs in HID0.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 7: System hang on tlbsync

Description:

Heavy traffic of outgoing **eieio** instructions in the internal output store queue may keep load operations from getting a fair chance at the address bus.

A code loop generating a continuous flow of **eieio** instructions keeps the bus store queue at higher priority than the bus load queue and may prevent a load in the bus load queue from accessing the address bus. This is a problem if the load is speculative and a **tlbsync** is presented on the bus.

Projected Impact:

Any processor executing such that a code loop may hang, although it is believed to be a rare case, is affected.

Workaround:

Change the offending code.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 8: System hang during hardware tablewalk

Description:

The system hang seems to be caused when the LSM is doing a hardware tablewalk, a **tlbld** is dispatched to the LSM.

This hang is caused by an instruction (load/store) that missed in the DMMU. When it missed in the DMMU, the instruction causes a hardware tablewalk.

While the hardware tablewalk is in progress, a **tlbld** instruction is issued to the LSM and waits in LSM RS0 until it receives a LSM sync from completion. Completion will not send a LSM sync until the **tlbld** is next to complete.

When the hardware tablewalk completes, it will attempt to update the TLB with PTE information. However, the **tlbld** that is in LSM RS0 causes the DMMU's PTE and PTCMP mux to incorrectly select the software tablewalk data instead of hardware tablewalk data. The result is another DMMU miss that causes another hardware tablewalk. This action will keep the load/store instruction from completing and keep the **tlbld** in LSM RS0, thus a deadlock.

Projected Impact:

Any processor that has a speculative **tlbld** issued while a hardware tablewalk is in progress will cause this hang.

Workaround:

None. The **tlbld** instruction can be in uninitialized memory location, thus possibly speculatively dispatched.

Projected Solution:

Fixed in Rev. 1.1.

Errata No. 9: FPU hangs the processor

Description:

fctiw, **fctiwz** instruction stalls the FPU pipeline.

If the following three conditions are simultaneously present, the FPU may cause the processor to hang.

1. Instructions: **fctiw** or **fctiwz**
2. Operands: a) $+2^{31} - 1$ b) -2^{31}
3. FPU invalid exception is enabled

Projected Impact:

The system hangs when this condition occurs.

Workarounds:

Either of the following will serve as a workaround to this erratum:

1. Disable the FPU invalid exception
2. Avoid **fctiw** and **fctiwz**

Projected Solution:

Fixed in Rev. 1.2.

Errata No. 10: External interrupt and system management interrupt are edge triggered

Description:

Asserting either the external or system management interrupt pins will result in only one interrupt being taken.

If $\overline{\text{INT}}$ is held low, only one external interrupt will be taken. Similarly, if $\overline{\text{SMI}}$ is held low, only one system management interrupt will be taken.

Projected Impact:

The second interrupt of a given type will not be taken. This may cause the system to time out due to the second interrupt not being taken.

Workaround:

Deassert the interrupt signal for at least one bus cycle at the end of the interrupt handler before executing the **rfi**.

Projected Solution:

Fixed in Rev. 1.2.

Errata No. 11:Instructions complete after MSR[EE] before external interrupt

Description:

Instructions following a **mtmsr** instruction that sets the MSR[EE] bit complete before vectoring to the external interrupt handler. This affects all external type interrupts: $\overline{\text{INT}}$, $\overline{\text{SMI}}$, decrementer, thermal, and performance monitor.

Projected Impact:

Code immediately after the **mtmsr** instruction that sets the MSR[EE] will get executed prematurely.

Workaround:

Place an **isync** after the **mtmsr**.

Projected Solution:

Fixed in Rev. 2.0.

Errata No. 12:Back-to-back store to overlapping address causes lost data

Description:

Back-to-back stores to overlapping address may cause lost data.

If two stores to overlapping bytes follow a series of three unrelated stores that hit, miss, and miss in the L1 cache, respectively, and the two stores are followed some cycles later by a load to the same cache line but non-overlapping bytes, the load may bypass the two stores and reload the L1 cache after the first store has missed in the cache. The second store is allowed to access the cache and hit before the first store re-accesses the cache, causing the second store’s data to be overwritten by the first store’s data.

Projected Impact:

Loss of store data.

Workarounds:

Either of the following will serve as a workaround to this erratum:

1. Disable the L1 cache or perform all stores as cache-inhibited.
2. Lock the L1 cache or perform all stores as write-through.
3. Insert **eieio** instructions between stores to overlapping addresses or every four stores to prevent pipelining of stores.

Projected Solution:

Fixed in Rev. 1.2.

Errata No. 13:‘Infinite’ code loop in MP system causes hang

Description:

An MP system may hang if a conditional loop on a processor requires a snoop to occur to exit, or if the processor is in an infinite loop, and either loop creates a continuous stream of store-type traffic.

The first processor may retry the second processor forever if (1a) a processor is looping on an address that hits in the L1 while expecting a second processor to snoop and then modify this address location in order to cause forward progress, or (1b) the processor is simply in an infinite loop; and (2) this conditional or infinite loop also produces a continuous stream of ‘store-type’ operations such as **eieio** or cache-inhibited or write-through stores; and (3) the load-store unit of the processor creates either (a) cancelled speculative load, (b) vector touch request, or (c) an L1 castout; and (4) a second processor executes a **tlbie** on the system bus while at least one of the ops from ‘3’ is in the load-store or memory subsystem; and (5) the second processor follows the **tlbie** instructions with a **tlbsync**.

Projected Impact:

This problem will only occur in MP systems. The code sequence is not believed to be used in general practice.

Workaround:

Do not generate these code sequences.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 14:Stalled touch hang

Description:

A guarded touch that stalls in the LSM may cause a hang.

If a guarded touch that is next-to-complete is stalled (for example, due to L1 miss queue full), it may be marked as speculative again when the completion buffer moves beyond the touch ID. Once the actual stall resolves, part of the LSM logic thinks the touch is stalled as a speculative guarded load, while the rest allows the next instruction to advance. If the next instruction is a store, it will be sent to the load miss queue as well as the store queue. Sending a store to the load queue causes a hang (‘store’ in load miss queue never receives reload).

Projected Impact:

The system will hangs if a guarded touch that stalls in the LSM is hit and the next instruction is a store.

Workaround:

No-op all touch instructions (set bit 31 of HID0).

Projected Solution:

Fixed in Rev. 2.0.

Errata No. 15:Mismatched stwcx. fails to report that store was performed

Description:

A mismatched **stwcx.** succeeds in storing its data but reports that the store did not occur when a second processor attempts to kill the reservation.

Processor 0 gains a reservation and performs a **stwcx.** to a coherency granule that does not match it.

Processor 1 performs an operation that could potentially kill processor 0’s reservation, that is, a store to processor 0’s reservation granule.

If this operation hits a small window, it could cause processor 0 to think the reservation was cancelled after the store was performed.

Processor 0 may then report that the **stwcx.** failed even though it succeeded.

The actual fail mechanism not yet known at this time and may be more complicated than described above.

Projected Impact:

If this scenario occurs, the application may see a loss of atomic coherency.

Example: A fetch-add primitive might add twice.

Workarounds:

Either of the following will serve as a workaround to this erratum:

1. Match all **lwarx stwcx.** pairs.

   This is good practice because the Book I definition of **stwcx.** states that it is undefined whether or not the store is performed when there are unmatched **lwarx stwcx.**
2. If performing a mismatched **stwcx.**, do not allow operations that might cancel the processor’s reservation after it is obtained.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 16: L3 hardware invalidate causes erroneous machine check exception

Description:

An L3 hardware invalidate will cause erroneous machine check exceptions if L3CR[L3PE] is set during the invalidate. Also, the status bit MSSSR0[L3TAG] will be set.

When an L3 hardware invalidate is performed, invalid tag data is read out of the L3 tag. The invalid tag data can trigger the L3 logic to incorrectly report a parity error to the exception logic, even though the data read out of the L3 tag is a don’t-care during an invalidate. The exception logic will either trigger a machine check exception or a checkstop based on the status of the MSR[ME] bit.

In addition, the status bit MSSSR0[L3TAG] will be set erroneously. During a normal machine check, the MSSSR0 register provides information as to the cause of the exception if SRR1 register bit 11 (MSS error), is set. Since the L3 hardware invalidate sets MSSSR0[L3TAG] erroneously, the cause of a subsequent machine check may be difficult to determine if software does not clear the MSSSR0[L3TAG] bit after the invalidate.

Projected Impact:

Systems will take unexpected machine check exceptions from an L3 hardware invalidate.

Workaround:

Software should keep L3CR[L3PE] disabled during an L3 hardware invalidate and clear MSSSR0[L3TAG] after the invalidate has finished.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 17: L3 private memory mode not functional when L3 cache disabled

Description:

The L3 will not function if it is operated as private memory only.

The L3 data output drivers and receivers are currently not enabled for private memory-only mode; therefore, the L3 external SRAMs cannot be written to or read from in this mode. The drivers/receivers should be enabled in this mode.

Projected Impact:

Systems using the L3 as private memory only will lose data written to the L3 SRAMs and receive unknown values for reads from the L3 SRAMs.

Workaround:

For 2-Mbyte SRAM, the L3 can be operated in one-half private memory and one-half cache mode. No workaround exists for 1-Mbyte SRAM.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 18:Processor will not enter nap mode after taking an interrupt

Description:

If an external or system management interrupt is taken, the processor will not be able to get into the nap/sleep power management modes afterwards.

After taking an external or system management interrupt, the COP unit fails to clear a flag that tells it that there is a pending interrupt. The COP unit will then not allow the processor back into nap or sleep because an interrupt is a wake-up condition.

Projected Impact:

Systems that use power management and cannot use the work-around will not see a power savings.

Workarounds:

Either of the following will serve as a workaround to this erratum:

1. Do not take an external or system management interrupt on the processor.
2. Setting the MSR:EE bit to a 0 allows the processor to go into nap and sleep modes. The system can then wake up the processor with a NMI (non-maskable interrupt) like $\overline{\text{SRESET}}$. Unfortunately, maskable interrupts will not wake up the processor with MSR:EE = 0.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 19: Setting MSSCR0[18–24] will cause machine check exceptions

Description:

The MSSCR0 register fields 18–24, if not all zeros, can cause the MPC7450 to take a machine check on internally triggered events without warning.

The MSSCR0 register fields 18–24 was originally intended for internal lab debug only. Setting one or more of these bits can cause the MPC7450 to take machine checks (or a checkstop based on MSR[ME]) due to unspecified internal events. These events, while useful for initial internal lab debug, have ceased to be of value and may cause undesired system behavior if set.

Projected Impact:

Software that sets MSSCR0[18–24] bits will cause the system to take spurious and sometimes frequent machine check exceptions.

Work-Around:

Do not set MSSCR0[18–24].

Projected Solution:

These bits have been marked '*Reserved for factory use only*' in the *MPC7450 RISC Microprocessor Family User's Manual*.

Errata No. 20: L3 cache is initialized incorrectly

Description:

If the L3 clock (L3CR[L3CLKEN]) is enabled before the L3 (L3CR[L3E]) is enabled, then the L3 interface will not receive data properly from the external SRAM.

The L3 cache initialization routine in early documentation specified that the L3 clock be enabled and stay enabled before the L3 is enabled. This initialization routine will cause the MPC7450 L3 to incorrectly receive data from the external SRAM.

Projected Impact:

The L3 interface will not operate correctly.

Work-Around:

The workaround for L3 initialization routine:

1. POR
2. Disable L3 cache
3. Set all L3CR register bits to their desired values except L3CLKEN, L3E, L3PE, and L3I.
4. Set L3CLKEN to a 1
5. Set L3I to trigger the hardware invalidate routine
6. Wait for invalidate to finish
7. Disable L3 clock by setting L3CLKEN to a 0
8. Perform a delay loop
9. Set L3E and L3CLKEN to a 1
10. Perform a delay loop

Projected Solution:

The workaround has been documented in the *MPC7450 RISC Microprocessor Family User's Manual* as the correct way to initialize the L3 cache.

Errata No. 21:Hardware prefetches ignore L3CR[IONLY]/L3CR[DONLY]

Description:

Hardware prefetch accesses triggered by instruction access misses will allocate in the L3 even if L3CR[DONLY] is set. Hardware prefetch accesses triggered by data access misses will allocate in the L3 even if L3CR[IONLY] is set.

If a cacheable access misses in the L1, L2, and L3, a line will be allocated in each cache, and a request will be made to the external bus. To make room for the allocating entry, an older cache line in each cache may be victimized by either being cast out or invalidated. The allocated line will be filled by data returned from the external bus.

Setting L3CR[DONLY] permits only data access cache misses to allocate in the L3. Similarly, setting L3CR[IONLY] permits only instruction access cache misses to allocate in the L3. Setting both of these bits at the same time effectively prevents both instruction and data access cache misses from allocating in the L3, potentially victimizing older L3 lines.

If hardware prefetching is enabled by setting MSSCR0[L2PFE], a cacheable access that misses in the caches will trigger a prefetch of an adjacent 32-byte line if the following are true: (1) There is a cacheable instruction miss and L2CR[DONLY] is not set, and/or (2) There is a cacheable data miss and L2CR[IONLY] is not set. Otherwise, no prefetch access will begin.

However, once these requirements have been met and a prefetch has been initiated, this prefetch is not considered either an instruction or data access by the L3. Therefore, if the L3CR[DONLY] bit is set, and the prefetch was triggered by an instruction miss, the prefetch data will be allocated in the L3. If the L3CR[IONLY] bit is set, and the prefetch was triggered by a data miss, the prefetched data will be allocated in the L3. Finally, if both L3CR[DONLY] and L3CR[IONLY] are set, any hardware prefetch will still allocate in the L3.

Projected Impact:

Software that uses L3CR[DONLY] and/or L3CR[IONLY] to prevent the allocation of data or instructions into the L3 will not logically fail but will have degraded performance if hardware prefetches are enabled.

Workaround:

Disable hardware prefetching MSSCR0[L2PFE] if setting L3CR[IONLY] and/or L3CR[DONLY].

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 22: Hardware prefetching impacts dcbz-intensive M = 1 software loops

Description:

The hardware prefetcher fetches the alternate sector to an M = 1 **dcbz**/**dcba** access and treats the prefetch as a read requiring data instead of an address-only **dcbz**/**dcba**.

An M = 1 **dcbz** will produce an address-only kill operation (ttype = 0x0c) on the external bus. A **dcbz** is frequently used by software to clear and initialize large tracts of memory without requiring a data tenure.

L2 alternate sector prefetches perform read operations (ttype = 0x0a) on an external bus that requires data tenures. L2 alternate sector hardware prefetches occur when a read (load, instruction fetch, or write-back store) misses in all of the MPC7450 caches and goes out to the external bus. An M = 1 **dcbz** operation is also treated as a read by the prefetcher and will initiate a hardware prefetch to its alternate sector. This prefetch will require a data tenure. In addition, subsequent M = 1 **dcbz** instructions will be internally blocked from accessing the external address bus until the previous prefetches data has been returned.

Projected Impact:

Software using **dcbz**-intensive M = 1 loops to load and clear memory will be impacted.

Workaround:

Disable hardware prefetching (MSSCR0[L2PFE]) before software enters a **dcbz**-intensive M = 1 code loop. This is not possible for user software since access to the MSSCR0 is privileged. An operating system will have to provide a hook for the user to enable/disable hardware prefetching.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 23: dcbf/dcbst-initiated castout sequence retries external snoop forever

Description:

An external snoop can collide with a **dcbf**- or **dcbst**-initiated L1 data cache (dL1) castout operation to the same address in the MPC7450 internal pipeline coincident with an L2 castout to this address, such that the MPC7450 will continuously assert retry to the external snoop and be inhibited from making forward progress.

An address can exist as modified in all three levels of the MPC7450 caches simultaneously, with the highest level of cache holding the most recent data. In addition, each cache level performs allocation and deallocation independently. As a result, up to three castout operations to the same address can exist in the MPC7450 memory pipeline simultaneously between the dL1 and L3. These castouts progress stage-by-stage through the MPC7450 memory subsystem pipeline to the bus interface unit, where they await arbitration for the external bus.

A window exists in the memory subsystem pipeline such that an L1 castout to address ‘A,’ initiated by a **dcbf** or a **dcbst**, and an L2 castout to address ‘A,’ due to the MPC7450 L2 replacement algorithm or by a **dcbf** or **dcbst**, may be corrupted by an external snoop to address ‘A,’ if the castouts reside in adjacent pipeline stages at the time of the external snoop. The L2 castout, although it has stale data, will allow the L1 castout to bypass it to the external bus. In addition, the L2 castout will enter a state in which it will retry the external snoop continuously. The scenario, if hit, will cause the system to hang, and the only remedy is to assert $\overline{\text{HRESET}}$.

The size of the window in time, in which an external snoop can cause the corruption of the L1/L2 castout sequence, is dictated by the frequency with which store-type traffic, including castouts, pushes, write-though and cache-inhibited stores, **sync** instructions, **eieio** instructions, and the like, are processed by the external bus.

Projected Impact:

The scenario, if hit, will cause the system to hang. The only remedy is to assert $\overline{\text{HRESET}}$.

Workarounds:

Workarounds to decrease the frequency of the failing scenario include limiting **dcbf**/**dcbst** code sequences during coherent I/O in a uniprocessor system, and increasing external bus throughput.

Workarounds to avoid the fail completely are less efficient: precede all **dcbf** instructions and **dcbst** instructions with a **sync**, or limit the L2 to instruction-only mode to eliminate L2 castouts.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 24:Load-store address collisions cause performance loss

Description:

Load-store address collisions in the load/store unit cause a performance loss, especially in applications that move data to/from floating-point registers.

A load-store collision is supposed to upgrade the priority of the store in L1 data cache (dL1) cache arbitration, but does not. The collision does not resolve until an internal store pacing mechanism kicks in. The default setting of LDSTCR will upgrade the store priority after it fails to win L1 cache arbitration several times.

This causes a performance loss for applications that have many loads following stores to the same cache line. Applications that move data to or from floating-point registers are particularly affected.

Projected Impact:

Poor floating-point performance will result from using the default LDSTCR values.

Workaround:

Determine if setting reserved bits LDSTCR[17–19] improve overall performance. The POR default setting in this register is 0 for those bits. If performance is not improved, no workaround exists, and a performance penalty will result.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 25: TAU reports incorrect temperatures

Description:

The thermal assist unit (TAU) on the MPC7450 and MPC7455 reports temperatures between 35 to 55 degrees lower than expected.

This erratum effects only customers using the TAU. If a trip temperature is programmed into the sensor's control registers, the output interrupt is never received, even if temperatures exceed the expected setpoint by up to 55 degrees (even after calibration). A control application will not be alerted of excessive temperatures and this could lead to damage of the part.

Projected Impact:

Programmed trip temperatures will not trigger output interrupts, even if temperatures exceed the expected setpoint by up to 55 degrees.

Work-Around:

None

Projected Solution:

None. This erratum will not be fixed in future revisions of the MPC7450. The TAU is not supported on the MPC7450.

Errata No. 26:MPC7450 L3 output pin voltage modes set incorrectly

Description:

The L3_CLK[0:1], L3_CNTL[0:1], and L3_ADDR[0:1] output pins are incorrectly programmed with the BUS_VSEL pins instead of the L3_VSEL pins.

The output impedance of all L3 pins should be programmed with the L3 bus voltage select (L3_VSEL) external pin. However, the following pins have their voltage mode incorrectly connected to bus voltage select (BUS_VSEL) instead of L3_VSEL:

L3_CLK[0:3]

L3_CNTL[0:1]

L3_ADDR[0:17]

All of the affected pins are output-only pins. The incorrect connection will set the affected pins to the incorrect output impedance for the applied GV_{DD}. The affected pins are connected to the correct GV_{DD} power bus, not the OV_{DD} power bus.

Projected Impact:

Systems that have different bus and L3 voltages may see some performance issues with the L3 bus due to different output impedance of the affected pins.

Workaround:

Set the bus and L3 voltage selects to the same voltage mode.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 27:Toggling the TBEN input signal causes unexpected behavior

Description:

The time base and decrementer do not increment correctly when the TBEN signal is toggled every cycle.

The time base enable (TBEN) pin is used to enable the incrementing of both the time base and decrementer. In normal operation, the TBEN signal is held in the active state at all times and is sampled once per system clock. Therefore, a timer will increment faster in a system that has a higher bus frequency.

To keep timers in sync for systems running at even multiples of each other, 66- and 133-MHz, for example, TBEN can be toggled every cycle in the faster processor, and therefore, only increment the time base every other system clock.

The MPC7450 implementation of the TBEN signal employs a two-stage digital noise filter. This filter requires that the TBEN signal be kept active for two consecutive bus clocks for the active state to be recognized inside the processor. This filtering causes this single cycle toggling TBEN scheme to fail.

Projected Impact:

Systems that toggle the TBEN input signal every cycle will be affected.

Workaround:

The system can toggle the TBEN signal in a two-cycles-on, two-cycles-off fashion.

Projected Solution:

Fixed in Rev. 2.0.

Errata No. 28:Line reserved in L2 due to dcbst/dcbtst pair

Description:

A **dcbst** followed by a **dcbtst** to the same address may result in that address being reserved indefinitely in the L2, causing a loss in L2 effective size, and therefore, performance.

The MPC7450 performs ‘front-end’ allocation in the L2 cache. If a request misses in the L2, the L2 allocates an entry at the time of the miss to make room for the subsequent reload by setting an ‘allocated’ bit in the L2 cache status for that line. This bit is cleared when the reloaded data is received. This allocated bit prevents this line from being evicted before the reload occurs. The L2 allocated bit can also be cleared by a castout from the L1 data cache (dL1).

The failing scenario is as follows: address ‘A’ is modified in the dL1 and shared in the L2. A **dcbst** instruction changes the state of the dL1 to invalid, and creates a write-with-clean operation with the modified data that becomes queued internally, waiting for arbitration into the L2. Next, a **dcbtst** instruction to address ‘A’ arbitrates into the L2, prior to the write-with-clean winning arbitration, but gets internally retried by the waiting write-with-clean. However, the **dcbtst** incorrectly queues up an operation for the L2 to set the allocated bit for the **dcbtst**. Normally, the operation to set the allocated bit is the next to win arbitration into the L2. However, an internal pipeline full condition due to store-type traffic backed up on the external bus delays the set-allocated operation for a cycle. During this cycle the original write-with-clean due to the **dcbst** wins L2 arbitration and changes the state of the line from shared to exclusive. The write-with-clean writes its modified data to the L2 and continues on to the L3. Finally, the set-allocated operation queued for the **dcbtst** wins arbitration to the L2 and sets the allocated bit, even though the line is now exclusive. Next, the **dcbtst** itself finally wins arbitration into the L2. Since address ‘A’ is now exclusive in the L2 due to the **dcbst**, the **dcbtst** no longer requires a reload from the external bus. The dL1 is reloaded to the exclusive state, but the allocated bit remains set in the L2.

If address ‘A’ in the dL1 eventually gets modified (cacheable store to dL1), this modified line will eventually be evicted from the dL1 and clear the allocated bit in the L2. Therefore, no L2 effective size degradation will be seen.

However, if the line in the dL1 transitions to invalid without a castout to the L2, the line will remain in the allocated state in the L2. This will prevent this line in the L2 from being allocated for any other address besides address ‘A.’ If address ‘A’ is ever invalidated in the L2 while the allocated bit is set (with a **dcbf**, for example), this L2 cache entry becomes unusable by any future addresses until the next system reset.

Projected Impact:

Software using **dcbst** and **dcbtst** instructions may experience a gradual degradation of effective L2 size over time.

Workaround:

Use a **dcbf** instead of a **dcbst**, or issue a **sync** after every **dcbst**.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 29:IMMU speculative tablewalk causes bad instruction fetch

Description:

Supervisor mode translation code needs to clear the MSR[IR] bit in a particular fashion, or the instruction memory management unit (IMMU) may not cancel speculative tablewalk requests.

A test program uses the following code sequence to change the MSR[IR] bit from 1 to 0.

mtmsr <---- change MSR[IR] to 0

isync

After a period of time, the MPC7450 will execute code from unintended address and produce unpredictable results.

What should happen:

The **mtmsr** instruction changes the MSR[IR] bit from 1 to 0. The **isync** instruction should cancel any outstanding IMMU tablewalk request and should reset the IMMU tablewalk state machine to idle state.

Instead:

The MPC7450 IMMU is designed to execute translation type operations such as instruction storage interrupt, instruction translation, hardware tablewalk, etc., only when MSR[IR] = 1. The sequencer fetch unit speculatively fetches instruction past the code sequence shown above. Before the above code sequence is executed, the IMMU detects a TLB miss, which causes a hardware tablewalk pending while fetching down the speculative path. After the **mtmsr** is executed, the MSR[IR] bit is 0. The **isync**, in this case, does not reset the IMMU tablewalk state machine because MSR[IR] = 0, so the MPC7450 does not cancel the pending hardware tablewalk, and the tablewalk is executed because the MPC7450 assumes that the tablewalk is not speculative. When the hardware tablewalk completes at a later time, the IMMU sends the RA to the I-cache which sends a quad word of instructions back to the sequencer fetch unit. At this point, the I-cache and the IMMU’s address is out of sync with the fetch unit’s address. The fetch unit expects the instruction returned for the most recent address. Instead, the fetch unit receives instructions for the tablewalk address that was cancelled.

Projected Impact:

Systems not able to implement the suggested workaround code may see the MPC7450 perform instruction fetches from unintended addresses.

Workaround:

Clearing the MSR[IR] bit is a supervisor mode function. If it is necessary for a supervisor code to clear the MSR[IR] bit, use the SRR1 register to load the new value into the MSR by executing an **rfi** or **sc** instruction. Using an **rfi** or **sc** instruction to change the MSR guarantees the IMMU hardware tablewalk state machine is reset before the MSR[IR] bit changes.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 30: L2 hardware flush may not flush every line from the L2 cache

Description:

A valid line in the L2 may be ignored during an L2 hardware flush if instruction fetches are trying to allocate into the L2 during the flush.

The MPC7450 performs ‘front-end’ allocation in the L2 cache. If a request misses in the L2, the L2 allocates an entry at the time of the miss to make room for the subsequent reload by setting an ‘allocated’ bit in the L2 cache status for that line. This bit is cleared when the reloaded data is received.

The L2 hardware flush mechanism sequentially performs a tag read on every index, sector, and way in the L2 tag. As it encounters a valid entry, an operation is queued to the L2 to either simply invalidate the line or push the data from the cache if the data is modified.

The failing scenario is as follows: an instruction access to address ‘A’ from the L1 instruction cache (iL1) accesses and misses in the L2. The alternate sector for address ‘A’ is currently allocated in the L2 in way ‘N’ and awaiting a reload from the L3 or external bus. Meanwhile, the victim selection logic has chosen way ‘N’ to evict from the cache if an eviction is necessary. Since address ‘A’ alternate sector is in way ‘N,’ no eviction is necessary. The miss for ‘A’ queues up an internal allocate request to the L2 to allocate into way ‘N.’ During the cycle in which the instruction allocate request arbitrates into the L2, an L2 hardware flush to an unrelated index/sector/way ‘X/Y/Z’ is also arbitrating into the L2. Normally, the allocate request would win. As a result, an internal address mux that chooses the address with which to access the L2 is selected as the allocate address ‘A.’ However, an under-qualified internal retry condition exists for the allocate, where if the victim selection logic points to a way in which the alternate sector is in the allocated state, which it is, then the new allocate request gets retried. In this failing scenario, the L2 hardware flush request wins, since the allocate request gets retried. Meanwhile, the address mux continues to select the allocate address ‘A.’ As a result, the L2 hardware flush routine moves on to the next L2 tag entry without having accessed the current index/sector/way ‘X/Y/Z.’

Projected Impact:

Data corruption will occur in systems whose software misses in the iL1 during an L2 hardware flush routine.

Work-Around:

Set the IONLY and DONLY bits in the L2CR prior to the L2 hardware flush.

Projected Solution:

The workaround has been documented in the *MPC7450 RISC Microprocessor Family User’s Manual* as the correct way to flush the L2 cache.

Errata No. 31:BTIC corruption

Description:

In certain cases of **tlbie** and **icbi** usage, the BTIC can become corrupted (data is written to the wrong entry). If that data is later accessed, the processor will incorrectly use the entries in the BTIC, executing one to four wrong instructions after a taken branch. The processor can get a wrong answer or hang.

The BTIC is a virtual instruction cache and must be flushed any time an I-cache entry is invalidated or instruction translation changes. During normal operation, the BTIC has a write alias condition that occurs if instruction address bits [18:29] of the currently executing branch match (alias) against a currently allocated BTIC entry. If a BTIC reset occurs just when a branch that has a BTIC alias condition is being executed as a taken branch, the BTIC correction update address is corrupted, leading to the wrong entry being written. The update address used can be effectively random, but can be the address of a branch in the instruction buffer that has not been executed yet (and will not be, given that the current branch is taken).

If the wrong BTIC entry is written and is later used by a another branch that happens to match the entry that was written, the processor will incorrectly execute one to four instructions that it receives from the BTIC.

The cases for a BTIC reset that could cause problems are:

1. **tlbie** arrives during a BTIC alias
2. **icbi** arrives during a BTIC alias

All other BTIC reset scenarios are covered by the current logic.

A specific example scenario that can occur is given below:

Currently, BTIC has allocated a branch with instruction address 0x0001000C. Then, the code in Table 4 is executed. The first time that instruction 3 (**bdnz**) executes, there is a BTIC alias condition because both branches match in instruction address bits [18:29] (0b000000000011). If a BTIC reset condition occurs due to arrival of a **icbi** or **tlbie** during the same cycle that this branch is executing the first time, the BTIC write that occurs four cycles later will be to a bad address. While, in general, this bad update address is random, the address will be the same as the instruction address for instruction 5 (an unconditional branch) for the code example below.

In this example, the BTIC will be written with instructions 0–3 (the instructions at the target address of instruction 3), but it will write it to entry that would have been used for instruction 5. If this code loops for several times and exits, instruction 5 will be executed for the first time, sometime after the incorrect BTIC write. Since the BTIC has been incorrectly updated, instruction 5 will hit in the BTIC (when it should have missed), and the BTIC will supply instructions 0–3. The processor does not detect this error and executes instructions 0–3, assuming that these are the instructions starting at address 0x000150000, instead of executing the actual instructions at that address (instruction 6–9). After these first four incorrect instructions, registers R5, R6, R7, R20, R21, R22, and the CTR all have incorrect architectural values. These architectural errors can then propagate to the rest of the processor/memory.

Table 4. Code Snippet for BTIC Alias Example

Instruction No.	Label	Instruction Address	Instruction
0	loop	0x000140000	**add** R5,R6,R7
1		0x000140004	**add** R6,R7,R8
2		0x000140008	**addi** R7,R7,#1
3		0x00014000C	**bdnz** loop
4		0x000140010	**add** R10,R5,R6
5		0x000140014	**b** END
6	END	0x000150000	**add** R20,R21,R22
7		0x000150004	**add** R21,R22,R23
8		0x000150008	**add** R22,R23,R24
9		0x00015000C	**add** R23,R24,R25
10		0x000150010	**add** R24,R25,R26

Projected Impact:

The processor may get the wrong answer or hang if BTIC corruption occurs.

However, for this to happen, three unlikely events have to occur:

1. BTIC alias update occurs
2. **tlbie**/**icbi** occurs on same cycle as the aliased branch is executed
3. The corrupted BTIC entry is actually used later. This would have to have been written to an address that actually has a branch, and that branch is used before the index is replaced by the replacement algorithm, or the BTIC is flushed by some other mechanism.

Workarounds:

Solution (1): Disable the BTIC through HID0. Works for both uniprocessor and multiprocessor but sacrifices performance.

Solution (2): Uniprocessor only, little performance hit, but much more involved.

If an **isync** (or other context serializing instruction) is added after all **tlbsysnc** instructions (which are required after **tlbie** instructions), corruption due to **tlbie** instructions in a uniprocessor system is impossible. **tlbi** instructions are supervisor-only, so this requirement could be feasible.

If an **isync** (or other **csi**) is added after all **icbi** instructions (or groups of **icbi** instructions with no more than one other branch in between, that is, a single **icbi** loop), corruption due to **icbi** instructions in a uniprocessor system is impossible (because at least two different branches are required before execution of corrupt instruction might occur).

However, since **icbi** can be run in user level code, adding the above requirement does not seem feasible, and thus, solution (2) does not seem to fully solve the problem.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 32:IMMU page fault/tlbie collision causes hang

Description:

The IMMU is just finishing the processing of an instruction side hardware tablewalk that results in a page fault. A **tlbie** is snooped from a second processor. The **tlbie** kills the tablewalk transaction, so that the control logic does not restart the tablewalk or report the ISI exception, and the part hangs.

The IMMU is in the process of handling an instruction side hardware tablewalk that should result in a plateful which will force an ISI exception to 0x0400. A **tlbie** is snooped from a second processor on the bus and arrives just as the tablewalk state machine is transitioning from the tablewalk to the exception reporting.

For other types of tablewalks, a **tlbie** snooped at this point will result in cancelling the tablewalk, and then restarting it to ensure that the page table entry is still valid. However, for page fault/ISI, the control logic cancels the tablewalk but does not restart the transaction. The part hangs because the IMMU state machine is still waiting for a tablewalk to complete, but since the IMMU did not restart, the tablewalk will never complete.

Projected Impact:

Hangs in multiprocessor systems.

Workaround:

Use software tablewalks when in MP for parts with this bug.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 33:Software tablewalks may corrupt TLB

Description:

Attempting to use software tablewalks can corrupt the TLB.

Revision 2.0 of the MPC7450 has a problem with supporting software tablewalks in that it can give the incorrect LRU in the TLBMISS register when a d-side software tablewalk exception is taken. Software uses this bit to determine which way of the TLB to write. Therefore, software may inadvertently write the same entry in both ways of an index when it is attempting to update the C-bit on a TLB entry that hit in the TLB. If both ways of an index contain the same PTE data, a lookup to that index will result in a multi-way hit causing the TLB circuit to incorrectly give zeros out as the first 20 bits of the EA.

Projected Impact:

Systems attempting to use software tablewalks will need to use the workaround.

Workarounds:

A way for software tablewalk exception code to avoid this scenario is to only write to way 0 of the TLB. This way, way 1 will never be used, and there can never be a multi-way TLB hit. This has the effect, however, of reducing the TLB to be 64-entry private memory.

Another way for software to avoid this scenario is to use bit 13 of the PTEMISS register as the way to write into the TLB. Bit 13 of the PTEMISS register corresponds to bit 13 of the EA, which is a bit that is not used to index into the TLB array. This allows for the full 128-entry 2-way set associative TLB to be utilized.

Projected Solution:

Fixed in Rev. 2.1.

Errata No. 34: L1_TSTCLK must be pulled low

Description:

Early revisions of the *MPC7450 RISC Microprocessor Hardware Specifications* incorrectly recommended pulling up L1_TSTCLK.

On all previous parts, L1_TSTCLK has been identified as a factory test pin and is recommended to be pulled high. This historical precedent was perpetuated in the *MPC7450 RISC Microprocessor Hardware Specifications*. However, on the MPC7450 Rev. 2.0, pulling this pin high prevents proper operation of the L2 cache and may limit the maximum frequency.

The *MPC7450 RISC Microprocessor Family User's Manual* does not address L1_TSTCLK or L2_TSTCLK.

Revision 1 and prior of the *MPC7450 RISC Microprocessor Hardware Specifications* notes these signals are test input signals for factory use only and must be pulled up to OV_{DD} for normal machine operation.

Projected Impact:

Tying L1_TSTCLK high may cause data corruption in the L2 and may limit the frequency of operation.

Work-Around:

Tie L1_TSTCLK low for proper device operation. It is recommended that L2_TSTCLK be tied to $\overline{\text{HRESET}}$, but other configurations will not adversely affect performance.

Projected Solution:

Documented correctly in all the MPC7450 family hardware specifications.

Errata No. 35: Snoop responses are not generated in PLL-bypass mode

Description:

The earlier members of the MPC7450 family of devices (MPC7450, MPC7451, MPC7441) will not generate snoop responses in PLL-bypass mode.

The MPC7450 never drives a snoop response to the bus when in PLL-bypass mode. With intervention enabled, it will respond with $\overline{\text{DRDY}}$ without asserting $\overline{\text{HIT}}$, resulting in a system hang. With intervention disabled or while in 60x mode, the MPC7450 will perform a snoop push without asserting $\overline{\text{ARTRY}}$, causing coherency to be lost. In either case, the snoop logic on the processor will lock up, causing the processor to hang.

Projected Impact:

The processor will hang and cause memory inconsistencies if it is required to generate a snoop response while operating in PLL-bypass mode. PLL-bypass mode can only be used in systems in which the MPC7450 will not be required to snoop.

Work-Around:

None.

Projected Solution:

Under review.

Errata No. 36:Bus store queue causes hang in 60x and MPX bus modes

Description:

The MPC7450 will encounter a hang condition in some systems in either 60x or MPX bus modes due to the bus store queue gaining and keeping high priority arbitration over the bus load queue.

The MPC7450 has a push queue, load queue, and store queue that arbitrate internally for the external bus. The load queue can contain cacheable/cache-inhibited load requests, cacheable store requests, as well as kills. The store queue can contain cache-inhibited and write-through stores, **eieios**, **syncs**, **tlbsyncs**, flushes, castouts, etc. The push queue only contains pushes due to snoop requests.

The push queue always has highest priority over the load and store queues. The load queue usually has higher priority over the store queue. The store queue can gain higher priority status than the load queue under two conditions described below. The inability of the store queue to relinquish this high priority status may cause a hang in some system configurations. The two conditions under which the store queue can gain high priority are:

Condition 1: The store queue has lost arbitration to the load queue three times consecutively.

Condition 2: An internal load operation behind a queued store request in the store queue collides with the store request address. Since the store queue on the MPC7450 is a nine-entry queue, the internal load operation may collide (cacheline-based collision) against any one of nine entries. If a collision does occur, the oldest entry in the store queue is raised to high priority status even though there may not have been a collision against the oldest entry itself.

In Rev. 2.0*x* of the MPC7450, for both conditions, the store queue entry will keep high priority status in internal arbitration over the load queue until a minimum of one store queue operation has completed successfully on the bus. Retries do not break the high priority status. In Rev. 2.1 of the MPC7450, the processor’s store queue will relinquish high priority status for condition 1 if it is retried, but not for condition 2.

For condition 2, the high priority status is not relinquished by the store queue until all store queue operations older than the address operation collided against, the address operation collided against, and the next three younger operations in the store queue have completed their address tenures successfully on the bus. If a new internal address collision occurs in the meantime with a store queue entry, the cycle repeats itself.

Two examples of potential system hangs are as follows:

Example 1: The processor may make a load request to address ‘A,’ which gets retried. As a result of one of the conditions stated above, the high priority mechanism within the processor engages, and the processor goes to the external bus with a store queue request to address ‘B.’ The system must be able to process the store queue request to ‘B’ (and potentially several other store queue operations as described for condition 2), because the store queue may not relinquish control until ‘B’ (and others) successfully completes its (their) address tenure(s). This is shown Figure 1, where condition 2 has caused the high priority mechanism to be engaged.

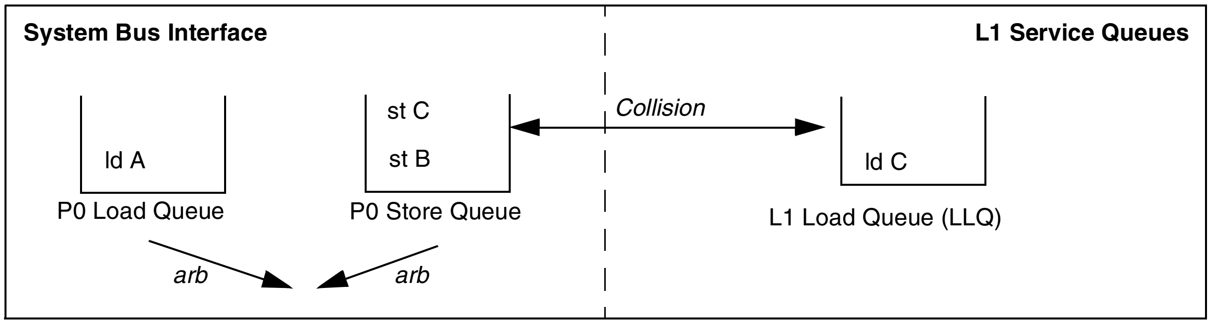
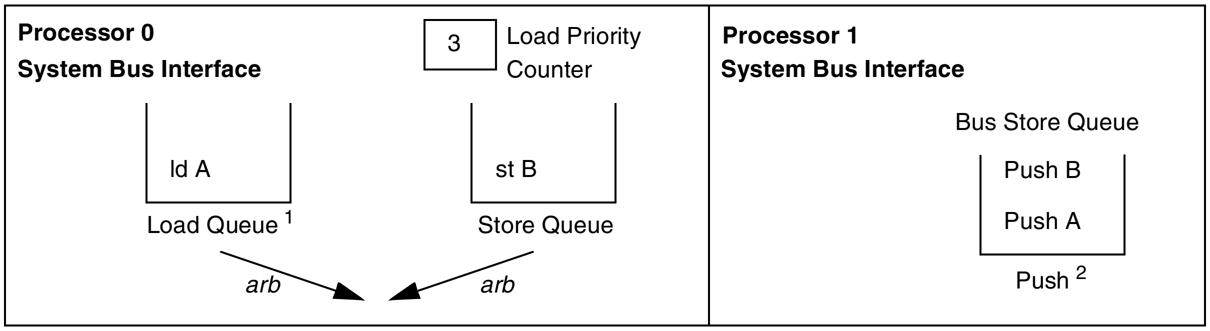


Figure 1. Example 1 with Condition 2

Example 2:In both 60x and MPX modes, the MPC7450 can cause push requests to the external bus, even if external intervention is enabled in MPX mode. In a multiprocessor system, a state can be entered where one processor (processor 0) has a load request to ‘A’ and a store request to ‘B,’ wherein the store request has gained high priority over the load request. A second processor (processor 1) has a push request queued for address ‘A’ followed by a push request to address ‘B.’ Processor 1 retries the store to ‘B,’ requests the window-of-opportunity, and attempts to push address ‘A.’ Although this is legal under the 60x/MPX specification, the MPC7450 is the first PowerPC processor to attempt this. As a result, some system controllers, including the Tundra Tsi106™ PowerPC host bridge or the Tundra Tsi107™ PowerPC host bridge, retry the push to ‘A’ because it requires a push to ‘B,’ since this is the address for which the window-of-opportunity was taken. A system that would not allow the push to ‘A’ to complete, would hang. This is shown in Figure 2, where condition 1 has caused the high priority mechanism to be engaged.



[1] The load queue is an 11-entry queue in the system bus interface.
[2] The push queue shares resources in the bus store queue with the castout queue, such that a combined total of ten pushes and castouts can be queued.

Figure 2. Example 2 with Condition 1

Projected Impact:

Systems with arbitration policies or queue resources that impose a requirement that a retried load be completed before a subsequent store request may hang.

Workaround:

In uniprocessor systems, the loads and stores can be separated by a **sync** instruction. No workaround for multiprocessor systems.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 37: Dropped instruction fetch when disabling instruction address translation

Description:

The instruction fetcher will drop a quad fetch if an **mtmsr** instruction used to clear MSR[IR] and the required **isync** are on different quad fetches, and the **isync** fetch is delayed due to a tablewalk.

The MPC7450 requires serialization instructions in order for certain move-to-special-purpose- register sequences to work correctly. Turning off instruction translation (relocation) by clearing MSR[IR] with an **mtmsr** instruction requires an **isync** to correctly reset the state.

If the **mtmsr** and **isync** are in different fetch groups, that is, quadword alignments, it is possible for the fetching to be halted for the **isync** fetch group due to the processor thinking that it needs to do translation for the **isync** fetch group because it has not yet processed the **mtmsr**.

This can happen in a uniprocessor system if the **mtmsr** is on the last quad of a page, the **isync** is on the first quad of the next page, and the next page is not in the TLB. This can happen in a multiprocessor system if a **tlbie** invalidates the page entry of the current page, where the **tlbie** occurs between the fetch of the **mtmsr** and the fetch of the **isync**. In both cases, a pending tablewalk is created for the **isync** fetch.

When the **mtmsr** is executed, and the expected pipeflush (from the **isync**) does not subsequently occur, the control logic gets into a bad state. The instruction fetcher drops the current quad fetch with the pending tablewalk (the one that includes the **isync**), and instead fetches the next quad and returns it with the effective address of the missing quad.

Projected Impact:

Systems where this occurs can see processes die (ISI or DSI) and may see memory corruption. The likelihood of this occurring is much higher in multiprocessor systems than in uniprocessor systems.

Workarounds:

In the uniprocessor case, avoiding having the **mtmsr** and **isync** on different pages is sufficient. For multiprocessor systems, the **mtmsr** and **isync** need to be in the same quad aligned fetch address. An alternative workaround is to use software tablewalks.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 38:TLB corruption caused by out-of-order I-cache reload and tlbie

Description:

A speculative tablewalk that writes the wrong TLB entry can occur if a tablewalk is pending, and the I-cache is doing an out-of-order reload.

The MPC7450 can handle up to two outstanding I-cache miss requests. If two miss requests are outstanding, such that the first (older) miss request comes back later than the second (newer) miss request, this is termed an out-of-order I-cache reload. One way this can happen is if the first miss goes to the bus, and the second miss hits in the L2 cache. If a **tlbie** arrives while these miss requests are outstanding, and this **tlbie** kills the page entry for the current fetches, a pending tablewalk state will be created for the next fetch.

The I-cache state machine has a bug with out-of-order reloads such that allows this pending tablewalk to start speculatively. In certain scenarios, this speculative tablewalk will cause problems.

One such scenario is if the second miss has a branch to another page that also has a page miss. The taken-branch will reset the state machine, and a new tablewalk will be requested for the branch target. The tablewalk engine returns the data from the first (speculative) tablewalk into the second tablewalk, thus leading the information from the first tablewalk to be written into the TLB entry for the second page miss. Thus, the TLB is corrupted.

Projected Impact:

This bug can only occur in multiprocessor systems. These systems may see problems including ISI, DSI, or memory corruption.

Workaround:

Use software tablewalks.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 39:Use of power management modes causes L3 read errors

Description:

Use of nap or sleep modes on the MPC7450 can cause read errors of the L3 SRAMs under certain conditions when going into the power management modes.

When nap or sleep mode is used on the MPC7450 with the L3 cache enabled, there exist windows of time during which external interrupts, decrementer interrupts, or thermal assist unit interrupts can prematurely abort quiescing of the L3 cache interface logic prior to entering the power savings state. This causes the L3 interface to not properly complete the shutdown required for entering the nap or sleep mode. The interface can then power up in an incoherent state which can cause read errors of the L3. The read errors can cause data corruption or illegal instructions to be executed.

Projected Impact:

L3 read errors can cause data corruption or illegal instructions.

Workaround:

Do not use the L3 if system uses the power management features.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 40:L3 private memory data corruption if L3 cache disabled

Description:

If the entire L3 is operating as private memory space, data in the L3 private memory may be corrupted if L1/L2 external bus reloads or external snoops collide internally with L3 private memory accesses.

If an L3 private memory access collides internally with an L1 or L2 reload request from the external bus, or with an external snoop, a 1-cycle window exists wherein the private memory access will access the external bus instead of the L3.

Projected Impact:

If an internal collision occurs and an unexpected external bus access is created, data corruption to a L3 private memory space or a bus error may result.

The bus error could result in an assertion of $\overline{\text{TEA}}$ or $\overline{\text{MCP}}$ by another device.

Workaround:

For 2-Mbyte private memory-only space, the L3CR can be configured to 2-Mbyte L3-enabled and 2-Mbyte L3 private memory enabled space to create 2-Mbyte private memory-only space; private memory space overrides cacheable space. L3CR[L3IO] and L3CR[L3DO] must be set. No workaround exists for 1-Mbyte private memory-only configurations.

Alternatively, software can be coded such that all other data and instruction fetches hit in the L1 and L2 caches during L3 private memory accesses. Also, external snooping would need to be disabled during L3 private memory accesses.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 41: Multiple bus requests asserted during window-of-opportunity

Description:

The MPC7450 will assert a bus request during another master’s window-of-opportunity if its own window-of-opportunity request has been ignored in systems with three or more bus masters.

The window-of-opportunity is defined as two cycles after $\overline{\text{AACK}}$, wherein the owner of a modified line may request the bus for a push in response to an external snoop. Since there can be only one owner of a modified line, only one bus request should be asserted during the window-of-opportunity.

For the MPC7450, it is highly recommended that when the processor requests the window-of-opportunity, the processor is allowed to perform the push address tenure on the bus before any other master’s tenures are allowed on the bus.

However, systems can, at their own risk, choose not to grant the bus to the processor with the window-of-opportunity push right away. In this scenario, it is possible for the MPC7450 to assert a bus request during another master’s window-of-opportunity.

The scenario is as follows:

1. Processor 0 snoops address on A on the bus, which it has modified.
2. Processor 0 requests the window-of-opportunity by asserting a bus request.
3. The system grants the address bus to a second master. This second master performs an address tenure to address B.
4. A third master has address B modified in its caches, retries master two, and requests the window-of-opportunity.
5. Processor 0, although it does not have address B in its caches, keeps the bus request asserted during this second window-of-opportunity.

This is technically a violation of the bus protocol.

Projected Impact:

Multiprocessor systems that do not let the MPC7540 perform a push after the window-of-opportunity has been requested will see the processor violate the bus specification in the way explained above.

Workaround:

If the MPC7450 requests the window-of-opportunity, a system should give the grant to that processor next. Or, a system can simply handle the assertion of multiple bus requests during the window-of-opportunity if it chooses to ignore the previous statement.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 42:TLBMISS register does not contain page index information

Description:

The data address register (DAR) cannot be easily reverse-engineered during software tablewalks because the MPC7450 does not save page index information in the TLBMISS register.

When the MPC7450 recognizes a software tablewalk, the page address of the fetch that requires a tablewalk is saved in the TLBMISS register. The TLBMISS register also contains a copy of the LRU bit for the addressed TLB set.

The *MPC7450 RISC Microprocessor Family User's Manual* states that the DAR can be recreated using the TLBMISS register. For MPC7450 Rev. 2.1 and earlier, however, only bits 0–19 of the effective address are saved in the TLBMISS register due to an error. For MPC7450 Rev. 2.3 and beyond, bits 0–30 are saved correctly for software tablewalks initiated by data fetches.

On all MPC7450 revisions, only bits 0–19 are saved for software tablewalks initiated by instruction fetches. The correct instruction fetch address must be retrieved from the SRR0 register.

Note that the TLBMISS register on MPC7450 Rev. 2.3 and later will be halfword accurate but not byte accurate since bit 31 of the TLBMISS register stores the LRU bit.

Projected Impact:

Debuggers in systems using software tablewalks have no easy way to determine the address of the exception. Watchpoints or breakpoints for loads and/or stores will not operate correctly.

Workarounds:

A proper DAR can be created by reverse engineering the instruction that caused the software tablewalk.

On MPC7450 Rev. 2.3 and beyond, no workaround is required for software tablewalks initiated by data fetches.

On all revisions, the correct instruction fetch address must be retrieved from the SRR0 register. No change is planned to correct the TLBMISS register in this scenario.

Projected Solution:

Fixed in Rev. 2.3 for data fetches. For instruction fetches, the correct value is available in the SRR0 register. No further corrective action is planned.

Errata No. 43:Address parity checked during snooped M = 0 operations

Description:

The MPC7450 will check for address parity errors during snooped M = 0 MPX/60x bus operations if HID1[EBA] is set.

The MPC7450 properly ignores all snooped M = 0 MPX/60x bus traffic with regards to coherency. The processor should also ignore all address parity errors when it is snooping M = 0 bus traffic. However, if the M = 0 transaction has incorrect parity, the MPC7450 will either checkstop or take a machine check exception (based on MSR[ME]) if the MPX/60x address parity enable bit, HID1[EBA], is set.

Projected Impact:

Systems with masters that create M = 0 traffic and do not drive correct address parity on the MPX/60x bus will cause the MPC7450 to take unexpected checkstops or machine check exceptions if HID1[EBA] is set.

Workaround:

Disable address parity checking by clearing HID1[EBA] in systems as described.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 44:Push request may hang when retried during doze state

Description:

An MPC7450 window-of-opportunity push in doze power management state will not attempt to re-arbitrate for the external bus a second time if it is retried.

The MPC7450 has three official power management states: full-on, nap, and sleep. Doze state is the intermediate state between full-on and nap or sleep. The processor transitions through doze state in two ways:

1. Software sets MSR(POW) = 1 and either HID0(NAP) = 1 or HID0(SLEEP) = 1. The processor asserts $\overline{\mathrm{QREQ}}$ and awaits a $\overline{\mathrm{QACK}}$ assertion by the system so it can enter nap or sleep state. This waiting period is doze state.
2. The system deasserts $\overline{\mathrm{QACK}}$ while the processor is in nap state so the processor can snoop traffic on the bus. This period is also doze state.

The MPC7450 will not initiate any bus traffic of its own during doze state except snoop response traffic. During doze state, the processor will snoop the external bus and respond with a push or intervention data if it has the snooped address line modified in its caches.

During doze state, if the processor asserts $\overline{\mathrm{ARTRY}}$ due to a snoop and requests the window-of-opportunity, and its subsequent window-of-opportunity push request is retried, the processor will incorrectly not re-assert $\overline{\mathrm{BR}}$ or $\overline{\mathrm{TS}}$ for that push.

When the push is retried, the processor inadvertently de-prioritizes the push internally and puts the push into an idle mode. (Note that the MPC7450 can request a window-of-opportunity push in MPX bus mode and may not always respond hit.) If the system arbiter does not implement a fair arbitration scheme such that the original transaction (that is, the one that caused the processor to assert $\overline{\mathrm{ARTRY}}$ and attempt the push) can again be attempted by the originating device, the system may hang.

However, if the system recognizes the processor’s deassertion of $\overline{\mathrm{BR}}$ and permits the operation that caused the push to be rerun, the processor will re-request the window-of-opportunity correctly and complete the push. Of course, this push will also hang if retried.

Projected Impact:

Systems that permit snooping in doze state and do not perform fair arbitration after a retried window-of-opportunity request may experience hangs.

Workarounds:

Either of the following is sufficient to prevent system hangs:

- Do not retry doze state push requests.
- Perform fair arbitration if a window-of-opportunity push is retried.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 45: PTE changed (C) bit corruption when data L1 cache disabled

Description:

A set PTE changed (C) bit may be cleared when the L1 data cache (dL1) is disabled, L2 hardware prefetching is enabled, and either the L2 and/or L3 caches are enabled.

When the dL1 is disabled, all data accesses except D-side tablewalk load accesses are converted to cache-inhibited requests. D-side tablewalk load accesses remain cacheable, but are marked transient so that they will not be allocated into either the L2 or L3 caches. Neither the L2 nor the L3 allocate transient load misses. As a result, subsequent PTE reference (R) and C bit updates are written as cache-inhibited accesses, and therefore, written directly to memory when the dL1 is disabled.

However, the L2 hardware prefetcher will fetch the alternate sector of a PTE load access and incorrectly allocate that address into the L2 and/or L3 when enabled. This can lead to a lost C bit as described in the following scenario.

A D-side access when the dL1 is disabled causes an initial transient cacheable tablewalk PTE fetch to an address which will miss in the L2 and L3. The alternate sector of this address is fetched by the L2 hardware prefetcher and incorrectly stored in the L2 and L3. If the D-side tablewalk engine performs an R/C store update to a PTE in this alternate sector, the cache-inhibited R/C store will bypass the L2 and L3 and update memory. A subsequent I-side tablewalk load would find a stale PTE in the L2 or L3 with neither of the R/C bits set. The I-side tablewalk engine will then perform a cache-inhibited store of the R bit to memory, overwriting the C bit set by the D-side store. Similarly, if the TLB entry of the original D-side store had been de-allocated from the TLB, a subsequent D-side load would find a stale PTE entry in the L2 or L3 and overwrite the C bit as did the I-side tablewalk.

Projected Impact:

A C bit may be corrupted in systems that keep the dL1 disabled and the L2 and/or L3 caches enabled. Since this may be a transient state for systems flushing the cache hierarchy, L2 hardware prefetching should be disabled when flushing the caches.

Work-Around:

Any of the following is sufficient to prevent corruption of a C bit:

- Enable the L1 data cache when the L2 and/or L3 is enabled.
- Disable the L2 hardware prefetch engine (MSSCR0[L2PFE] = 0) when the L1 data cache is disabled.
- Set L2CR[IONLY] and/or L3CR[IONLY] when the dL1 is disabled.

Projected Solution:

Under review.

Errata No. 46: Earliest transfer of MPX/60x data requirement

Description:

The MPC7450 does not support the transfer of any data on MPX/60x bus before the processor’s snoop response window, defined as the cycle after $\overline{\text{AACK}}$ is asserted.

The MPC7450 implementation of the earliest transfer of data during an MPX/60x bus transaction is more restrictive than previous processors including the MPC7400/MPC7410. This implementation creates a backwards compatibility issue with older system chipsets, including the Tundra Tsi106™ PowerPC host bridge or the Tundra Tsi107™ PowerPC host bridge revisions prior to Rev. 1.4. In addition, any newly designed chipsets must adhere to this restriction.

In an MPC7450 system, the system chipset logic must ensure that the last (or only) assertion of $\overline{\text{TA}}$ for a data transfer does not occur sooner than the last cycle of the snoop response window (cycle after $\overline{\text{AACK}}$). Note that $\overline{\text{DBG}}$ can still be driven as early as $\overline{\text{TS}}$. Likewise, the system chipset logic must ensure that $\overline{\text{TEA}}$ is not asserted before the last cycle of the snoop response window.

The Tsi106 host bridge and the Tsi107 host bridge prior to Rev. 1.4 may transfer read and write data with the processor ending at or before the cycle of $\overline{\text{AACK}}$, which is one cycle before that permitted by the MPC7450. Systems that include either of these two bridge devices, or devices with the same behavior, are susceptible to processor hangs or data corruption as a result of this issue.

Projected Impact:

Any system where a device may assert $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ for a data tenure before or while the $\overline{\text{AACK}}$ for that transaction is driven.

Work-Around:

Devices must not assert $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ for a transaction before that transaction’s snoop response window (cycle after $\overline{\text{AACK}}$), as documented in the *MPC7450 RISC Microprocessor Family User’s Manual*.

Projected Solution:

The MPC7450 implementation of the earliest transfer for data is as documented in the *MPC7450 RISC Microprocessor Family User’s Manual*.

Errata No. 47:dcbz collision with external snoop may cause dcbz to hang

Description:

If an internal **dcbz** operation in the MSS collides with an external snoop to the same address, and that address is only modified in the L3, the **dcbz** may never complete.

A **dcbz**-type operation in the MSS can be created by a **dcbz** or **dcba** instruction. A **dcbz** operation can also be created by cacheable stores to the same cache line that are 'store-gathered' in the LSM to 32 bytes. If the multiple cacheable stores gathered together fill an entire cache line, they are converted into a single **dcbz** that can claim the cache line without requiring a data access.

A **dcbz** operation that does not hit modified in the processor produces a kill operation on the external bus. A **dcbz**-type operation that hits modified in the caches of the MPC7450 does not require an external bus access. A **dcbz** that hits modified in the L3 does not need to access the L3 data interface because data is not required for a **dcbz**.

The MSS can only process one **dcbz** at a time. A **dcbz**-type operation in the MSS takes four cycles to access the L2 and L3 tag arrays in parallel and hit modified in the L3. During this access, there exists a protected window where an external snoop to the same address as the **dcbz** will internally retry the **dcbz** and externally retry the snoop. The **dcbz** should normally be internally re-attempted, and the dL1 reloaded from the L3. When the snoop returns, it will hit in the dL1.

However, due to a logic problem, the **dcbz** request's internal retry bit does not get cleared, and the **dcbz** enters a stalled state where it is continuously internally retried every six cycles. If the snoop does not return, and the L3 remains modified, the **dcbz** will never complete, and the processor will eventually hang. In a typical system, the external snoop will return and change the state of the line in the L3 to either shared or invalid. The stalled **dcbz** will be allowed to complete, and the processor will not hang.

The chances of this scenario occurring depends on the frequency with which an external snoop can collide with a **dcbz** operation in a given system, and whether a snoop is guaranteed to return once retried. Systems that limit these types of collisions and guarantee the return of retried snoops will not likely encounter problems due to this erratum.

For systems in which snoop/**dcbz** collisions are likely, a complicating factor exists for all core:bus multipliers such that a returning external snoop may not be permitted by the stalled **dcbz** to access the L3 and clear the hanging condition.

Within the six-cycle stalled state of the **dcbz**, there is a one-cycle protected window in which a snoop to the address of the **dcbz** will again be retried. This protected window occurs once every six cycles since the **dcbz** is in its six-cycle stall-retry loop. If this protected one-cycle **dcbz** window is aligned with the first processor cycle of the system clock in which the snoop returns, the snoop is guaranteed to be retried again.

For a 6:1 core:bus multiplier, if these cycles do align, the **dcbz** protected window will always be aligned with the snoop (due to the bus clock being six core clocks in length), and the system is guaranteed to hang due to a harmonic livelock.

For other core:bus multipliers, a harmonic livelock can also be entered if the snoop always returns every 'N' number of system clocks (or any integer multiple of 'N') where 'N' is defined as the least common multiple of the core:bus ratio and six (number of **dcbz** stalled states), divided by the core:bus ratio. For example, for 6:1, N = LCM[6:1,6]/6:1 = 6/6 = 1 system clock. So, if a snoop returns on any multiple of one system clock (effectively every possible system clock), the system will hang. The harmonic livelock frequency for each core:bus ratio is shown in Table 5 as the snoop harmonic.

Table 5. Chance of Livelock Condition

Bus Multiplier	Snoop Harmonic	Minimum System Harmonic [1]
6 to 1	1,2,3,4, ...	1
3 to 1	2,4,6, ...	2
2, 4, or 8 to 1	3,6,9,12, ...	3
4.5 or 7.5 to 1	4,8,12, ...	4
5 or 7 to 1	6,12,18,24, ...	6
2.5, 3.5, 5.5, or 6.5 to1	12,24,36, ...	12

[1] Measured in system bus cycles.

Other requests to the MSS will produce bubbles in the MSS pipeline, and shift the retried **dcbz** instruction's protected window. So, instruction fetches, load requests, castouts, hardware prefetches, bus reloads, or snoop requests to other addresses may move the alignment of the protected window with the snoop, and permit the snoop to not be retried, change the state of the L3, and allow the stalled **dcbz** to complete.

Projected Impact:

Systems with a 6:1 core:bus multiplier, **dcbz** instructions and store-gathered accesses to snoopable addresses are most at risk. Systems that can return a snoop quicker than the minimum system harmonic shown in Table 5 are not susceptible.

Workarounds:

The workarounds for instruction **dcbz**-types and store-gathered **dcbz**-types are as follows:

1. **dcbz** (**dcbz**/**dcba** instruction): Do not snoop **dcbz** address space during **dcbz** instruction executions.
2. **dcbz** (store-gathered): Keep all store-gathering (cache-inhibited, write-through, cacheable) disabled by not setting HID0[SGE]. An alternate solution is to set bit 18 of the LDSTDB register. This will permit all store gathering, but will perform a 32-byte cacheable store instead of a **dcbz**.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 48:Snooped kill followed by snoop to same address hangs processor

Description:

If a kill-type external snoop hits modified data in the L1 data cache (dL1) and is followed by an MPX-style address streamed snoop to the same address as the kill, the MPC7450 may hang.

An MPX kill-type snoop is defined as either a kill or a write-with-kill operation. The subsequent snoop type can be any one of the following: clean, write-with-flush, flush, write-with-kill, read, kill, read-with-intent-to-modify, write-with-flush-atomic, read-atomic, or read-with-no-intent-to-cache.

For a kill-type snoop to address ‘A,’ where ‘A’ is modified in the dL1, the MPC7450 queues the modified dL1 data into a push buffer to await the address retry window results for the snoop.

If the kill-type operation is not retried on the bus by any device, the modified data is simply invalidated. If the kill-type operation is retried on the bus, the modified data is turned into a push (but not a window-of-opportunity push).

In the failing scenario, a kill-type snoop is not retried, so the MPC7450 properly invalidates the killed data. Before it is invalidated, however, an MPX-style address streamed operation collides against the push buffer. Detecting the collision, the snooper incorrectly creates a window-of-opportunity push for the modified data in the push buffer, even though it is being invalidated.

The snooper tells the BIU to expect a window-of-opportunity push, but since the modified data gets invalidated, there is no longer any data to push, so the MPC7450 hangs.

Projected Impact:

Devices that can master a kill-type operation followed by a second address streamed snoop to the same address in MPX bus mode may hang.

Workaround:

Do not produce the snooping sequence described above, or place a dead cycle between the first and second bus operations.

MPC7450 masters never create bus traffic of this type.

Projected Solution:

Fixed in Rev. 2.3.

Errata No. 49: Missed snoop transaction due to ignored $\overline{\text{QACK}}$ deassertion during transition to low power mode

Description:

A deassertion of $\overline{\text{QACK}}$ will be ignored for a certain number of cycles after entering nap mode. A snoop transaction presented during this time will not be serviced and may cause memory incoherency or processor hangs.

Upon entering a low power mode (nap or sleep), the L3 interface is powered down. However, read errors can result if the system attempts to wake the processor before the L3 has been properly powered down (see Error 39 in the *MPC7450 Family Chip Errata for the MPC7451, MPC7450, and MPC7441*). To prevent these errors, the processor enters a protected time window during which it ignores the state of the $\overline{\text{QACK}}$ signal while it powers down the L3. This window exists when L3CR[L3CLK] is set and does not depend on L3CR[L3E]. A consequence of this protected window is that the processor will remain in the low power mode for a minimum period of time. During this time, the processor cannot be awakened by deasserting $\overline{\text{QACK}}$ and is unable to snoop bus transactions. The processor will recognize the deassertion of $\overline{\text{QACK}}$ at the end of the protected period, however, if $\overline{\text{QACK}}$ is held deasserted.

Note: There is an eight-bus cycle period required after the deassertion of $\overline{\text{QACK}}$ for the processor to transition from nap or sleep mode to doze mode (see the *MPC7450 RISC Microprocessor Family User's Manual*). This requirement is separate from the minimum time period described herein, therefore, $\overline{\text{TS}}$ must not be asserted until eight cycles after the expiration of that period. Asserting $\overline{\text{TS}}$ for a snoop transaction too early will cause the transaction to be ignored, possibly resulting in memory incoherency or a hang condition.

Projected Impact:

Systems using the L3 interface and power management modes may experience memory corruption or hangs.

Work-Around:

Do not issue a transaction the processor must snoop until eight bus cycles after the protection window expires. The length of the protected window is measured in bus cycle and is determined by both the bus:core clock ratio and the core:L3 clock ratio, as shown in Table 6.

Table 6. Protection Window Duration (Bus Cycles)

Bus:Core Ratio	Core:L3 Ratio						
	2	**2.5**	**3**	**3.5**	**4**	**5**	**6**
2	25	33	31	41	38	45	52
2.5	20	27	25	33	30	36	42
3	17	22	21	28	25	30	35
3.5	14	19	18	24	22	26	30
4	13	17	16	21	19	23	26
4.5	11	15	14	19	17	20	23
5	10	14	13	17	15	18	21

Table 6. Protection Window Duration (Bus Cycles) (continued)

Bus:Core Ratio	Core:L3 Ratio						
	2	2.5	3	3.5	4	5	6
5.5	9	12	12	15	14	17	19
6	9	11	11	14	13	15	18
6.5	8	11	10	13	12	14	16
7	7	10	9	12	11	13	15
7.5	7	9	9	11	10	12	14
8	7	9	8	11	10	12	13
9	6	8	7	10	9	10	12
10	5	7	7	9	8	9	11
11	5	6	6	8	7	9	10
12	5	6	6	7	7	8	9
13	4	6	5	7	6	7	8
14	4	5	5	6	6	7	8
15	4	5	5	6	5	6	7
16	4	5	4	6	5	6	7

Projected Solution:

None. This erratum will not be fixed in any subsequent revisions of the MPC7450 family.

Errata No. 50: AltiVec lvxl/stvxl instructions allocate lines in the L2/L3 caches

Description:

The **lvxl**/**stvxl** AltiVec instructions are defined to be transient and not allocate into the L2/L3 caches. However, allocation does occur for these instructions.

On the MPC7450, a typical cacheable load-type instruction allocates data simultaneously in the L1 data cache (dL1), L2, and L3 caches. In the dL1, the line is allocated in the most-recently-used state. If the line is deallocated from the dL1, then the data may still reside in the L2 or L3, thus preventing an access to main memory.

This method works efficiently for addresses that are frequently accessed, but is inefficient for addresses that are utilized just once.

For the latter type of addresses, the AltiVec architecture defines the **lvxl** and **stvxl** instructions that leave dL1 cache entries in least-recently-used state instead of most-recently-used state. In addition, for the MPC74xx and MPC75xx family of parts, these load-type operations are defined as transient, and therefore, should not allocate data in the L2 (and L3) cache(s). As a result, addresses that are accessed just once will not occupy valuable L2 (or L3) cache real estate.

For the **lvxl**/**stvxl** instructions, the MPC7450 correctly implements the least-recently-used feature in the dL1, but does not implement the transient behavior for the L2 and L3.

Projected Impact:

A loss of performance may occur since memory accessed by **lvxl**/**stvxl** instructions that is known to have little re-use will be allocated in the L2/L3 caches of the MPC7455, taking cache entries from other data which may be more frequently accessed.

In addition, systems implementing software that assumes data accessed by the **lvxl**/**stvxl** instructions will never be resident in the L2 or L3 will not function properly.

Work-Around:

The **lvxl**/**stvxl** instructions can be forced into transient mode by setting bit 31 of every **lvxl**/**stvxl** instruction opcode. Other PowerPC instruction set architecture compliant processors will ignore this bit of the opcode since the bit is defined as reserved.

Projected Solution:

Under review.

Errata No. 51: tlbld instruction can cause incorrect instruction translation

Description:

An executed or speculatively executed **tlbld** instruction may cause the hardware tablewalk engine to return an incorrect result for an instruction tablewalk.

If a **tlbld** instruction is executed by the load/store unit (or is speculatively executed on the other side of a not taken branch), and the next operation executed by the load/store unit is an instruction hardware tablewalk, then the instruction tablewalk may fetch an incorrect result. The subsequent look-up of the i-side TLB may then cause the instruction fetch to incorrect or invalid addresses.

The typical scenario in which this issue would be seen is when software tablewalks are enabled (HID0[STEN] = 1), a **tlbld** instruction is executed, then software tablewalks are disabled (HID0[STEN] = 0), and a hardware tablewalk occurs for an instruction fetch. If no load or store operations are performed between the **tlbld** and disabling the software tablewalks, then the iTLB may contain bad data.

Projected Impact:

Systems using **tlbld** instructions may cause the hardware tablewalk engine to return an incorrect result for an instruction tablewalk.

Work-Around:

Any one of the following is sufficient to prevent the failure condition:

1. Do not use **tlbld** instructions, even in non-executed code on the other side of an always taken branch.
2. Put **tlbld** instructions on a different page than any code requiring hardware tablewalks. Then, before returning to hardware tablewalk mode (HID0[STEN] = 0), run a benign instruction such as a **sync** or **dss** in order to clear out any **tlbld** residue from the load/store unit.
3. Use software tablewalks only.

Projected Solution:

Under review

Errata No. 52: Variable size memory accesses via COP cannot be performed using the service bus

Description:

Variable size memory accesses must be performed using the LSRL instead of the service bus, causing these accesses to take considerably longer.

The current service bus architecture does not allow variable size memory accesses. All memory accesses using the service bus must be the size of one cache-line (32 bytes). As a result, the LSRL must instead be used to perform variable size accesses, causing the performance of emulators and debug tools to suffer dramatically.

Projected Impact:

Emulators and debug tools will experience reduced performance while performing operations that require variable size memory accesses.

Work-Around:

Use LSRL to perform variable size memory accesses.

Projected Solution:

None. This erratum will not be fixed in any subsequent revisions of the MPC7450 family.

Errata No. 53: L2 hardware prefetching occurs for transient requests

Description:

The L2 hardware prefetch engine will incorrectly fetch the alternate sector of a transient load/store request and allocate it into the L2 and L3 caches.

The **dstt**, **dststt**, **lvxl**, and **stvxl** instructions are defined to be transient, and data accessed by these instructions should not be allocated into the L2 or L3 cache. If the L2 hardware prefetch engine is enabled (MSSCR0[L2PFE] $\neq$ 0), then a transient request will trigger a hardware prefetch of the alternate sector of the transient request.

Software expecting the alternate sector of a transient address to also be transient and not be allocated in the L2 and L3 caches may see a performance loss.

Also, as noted in Error 50, **lvxl** and **stvxl** are not considered transient, so both the address accessed by the **lvxl** or **stvxl** instructions and the alternate sector will be allocated into the L2 and L3 caches.

Also note that the **dststt** instruction is not recommended for use on the MPC7450.

Projected Impact:

Software expecting the alternate sector of a transient address to also be transient and not be allocated in the L2 and L3 caches may see a performance loss.

Work-Around:

Disable hardware prefetching when the performance gain expected by declaring data as transient will outweigh the benefits of the hardware prefetching of other data accesses.

Projected Solution:

Under review.

Errata No. 54: L2/L3 data parity error forces checkstop instead of machine check

Overview:

An L2 or L3 data parity error should cause a machine check exception if data parity checking is enabled and if MSR[ME] is set. The MPC7455 incorrectly checkstops for these errors.

Description:

An L2 or L3 data parity error should cause a machine check exception if data parity checking is enabled and if MSR[ME] is set. The MPC7455 incorrectly checkstops for these errors.

L2 data parity and L3 data parity errors can be set to trigger either a checkstop or a machine check exception if the L2CR[L2PE] or L3CR[L3PE] bits are set, respectively. If MSR[ME] is not set, then the processor will checkstop. If MSR[ME] is set, then the processor should take a machine check exception. The MPC7455 incorrectly checkstops instead of taking the machine check exception.

For the MPC7455, L2 and L3 data parity errors incorrectly cause a sticky state within the processor that continually signals a parity exception to the exception processing unit. Therefore, while the exception processing unit clears the MSR[ME] bit and starts to request the machine check exception vector for the initial parity error, the sticky parity error state within the processor signals new non-existent parity errors to the exception processing unit. Since MSR[ME] is no longer set, the MPC7455 will checkstop.

This problem does not exist for either L2 tag or L3 tag parity errors.

Projected Impact:

Because the MPC7455 will checkstop, systems that enable L2 and/or L3 data parity checking will not be able to recover from these exceptions.

Work-Around:

If a checkstop is unacceptable behavior when an L2 or L3 data parity error occurs, then do not enable L2 or L3 parity checking.

Projected Solution:

Under review.

Errata No. 55: Instructions following a dcba mapped T = 1 may not be executed

Description:

A single instruction that is one, two, or three instructions following a **dcba** instruction with an address mapped using Direct-Store Segment Address Translation (T = 1) may not be executed.

Direct-Store Segment Address Translation has been removed from the architecture and is not supported on the MPC750, MPC7400, or MPC7450 processors. However, the following cache operations, when mapped T = 1, should still no-op as defined in the original architecture: **dcba**, **dcbz**, **icbi**, **dcbst**, **dcbi**, **dcbf**, **dcbtst**, and **dcbt**.

On the MPC7450, the **dcba** instruction will correctly no-op when T = 1, but will incorrectly cause subsequent instructions to not be executed. Specifically, one of the instructions in a window of three instructions immediately following the **dcba** may not be executed. All three instructions may be executed correctly, but a maximum of one may not be executed. Whether or not this one instruction is executed is highly dependent upon internal timings.

Note that if branch folding is enabled (HID0[FOLD] = 1), and one or multiple branch instructions following the **dcba** are folded, then these branch instructions do not count towards determining the window of three instructions following the **dcba**.

Projected Impact:

Any code that executes **dcba** instructions with T = 1 effective addresses may not see all instructions executed. Since Direct-Store Segment Address Translation was originally included in the architecture to support legacy POWER architecture I/O devices that used this interface, the impact to customers should be minimal.

Work-Around:

A workaround can be achieved by doing one of the following:

1. Do not map **dcba** instructions to T = 1 space.
2. Follow each **dcba** with three no-op instructions.
3. Follow each **dcba** instruction with an **isync** instruction.

Projected Solution:

None. This erratum will not be fixed in any subsequent revisions of the MPC7450 family.

Errata No. 56: $\overline{\text{CKSTP\_OUT}}$ asserted incorrectly and output signals not disabled

Detailed Description:

The $\overline{\text{CKSTP\_OUT}}$ signal is driven asserted for a period of time less than one system clock for some checkstop conditions. Also, during these checkstop conditions, all non-test output signals are not disabled.

If MSR[ME] = 0 and a machine check condition occurs, the processor enters the checkstop state. The machine check conditions are as follows:

1.Assertion of the machine check signal ($\overline{\text{MCP}}$) with HID1[EMCP] = 1.

2.MPX/60x bus error ($\overline{\text{TEA}}$).

3.MPX/60x address bus parity error with HID1[EBA] = 1.

4.MPX/60x data bus parity error with HID1[EBD] = 1.

5.Instruction cache parity error with ICTRL[EIEC] = 1 and ICTRL[EICP] = 1.

6.Data cache parity error with ICTRL[EDCE] = 1.

7.L2 tag/data parity error with L2CR[L2PE] = 1.

8.L3 tag/data parity error with L3CR[L3PE] = 1.

For conditions 2 through 8, the MPC7455 incorrectly asserts $\overline{\text{CKSTP\_OUT}}$ for a period as short as one processor clock. In addition, all non-test output signals should be disabled during the checkstop. However, the processor only disables all non-test output signals for the same short period of time for conditions 2 through 8. Checkstops caused by assertion of the $\overline{\text{CKSTP\_IN}}$ signal behave correctly and are not affected by this erratum.

Note: For devices affected by Error No. 14 (see Table 6), L2/L3 data parity errors do cause the MPC7455 to fully enter the checkstop state and assert $\overline{\text{CKSTP\_OUT}}$ continuously; see Error No. 14 for more information.

Projected Impact:

Editors note: due to the special fonts, the paragraph below can not be cross-referenced to the “Summary of Silicon Errata” tables, if the projected impact description is modified below, make sure to update the descriptions in the tables listed below as well:

Table 3, —MPC7455 Summary of Silicon Errata

The assertion of the $\overline{\text{CKSTP\_OUT}}$ signal may not be recognized by external logic, especially if the logic is synchronous or not edge-sensitive to changes in this signal.

Contention may occur on the output pins if an external device recognizes the checkstop and, expecting the MPC7455 outputs to be disabled, drives any non-test signals also being driven by the MPC7455.

Work-Around:

External logic monitoring $\overline{\text{CKSTP\_OUT}}$ must be designed to be edge-sensitive in order to detect the falling edge when the signal is asserted. Because of the high core frequencies of the MPC7455, this may not be a viable workaround in many designs because external logic may not be able to recognize a pulse that lasts one core clock, or the pulse width may be too short for the $\overline{\text{CKSTP\_OUT}}$ signal to transition below the logic’s low input threshold voltage.

Projected Solution:

Under review.

Errata No. 57: L2 hardware prefetcher not quiesced before $\overline{\text{QREQ}}$ asserted

Description:

The L2 hardware prefetcher may have operations outstanding when the processor asserts $\overline{\text{QREQ}}$ as a prelude to entering nap/sleep.

Before asserting $\overline{\text{QREQ}}$ due to the setting of MSR[POW], the MPC7450 waits until all outstanding instruction-initiated load operations and all store-type operations have completed in the internal caches or on the external bus.

However, outstanding L2 hardware prefetches that have not yet successfully completed their address tenures do not factor into the quiesce logic. As a result, up to three L2 hardware prefetches may be pending in the processor's load queue after MSR[POW] has been set. These prefetches will assert bus request, and will begin an address tenure if given a bus grant.

Only prefetches that have successfully completed their address tenures prior to setting MSR[POW] will have been visible to the quiesce logic and will complete before $\overline{\text{QREQ}}$ is asserted.

Projected Impact:

The issues resulting from the pending prefetches are dependent on how a particular system controller handles $\overline{\text{BR}}$ and/or $\overline{\text{TS}}$ assertions after $\overline{\text{QREQ}}$ has been asserted. If $\overline{\text{QACK}}$ is asserted while $\overline{\text{BR}}$ is asserted, $\overline{\text{BR}}$ may be held asserted until the processor is awakened, potentially confusing the system bus arbiter. If the bus arbiter issues a bus grant to the processor in response to the assertion of $\overline{\text{BR}}$ after $\overline{\text{QREQ}}$ is asserted, the transaction must be allowed to complete.

In general, a system that meets either of the following criteria will not be impacted by this erratum:

- The bus arbiter ignores the $\overline{\text{BR}}$ signal from any processor that has its $\overline{\text{QREQ}}$ signal asserted. The arbiter must continue to ignore $\overline{\text{BR}}$ until the processor is awakened and could issue a legitimate bus request. For example, a window-of-opportunity bus request in response to a transaction the processor was awakened to snoop.
- The bus arbiter does not ignore $\overline{\text{BR}}$ but the logic controlling $\overline{\text{QACK}}$ does not assert it until at least two bus cycles after $\overline{\text{QREQ}}$ is asserted, ensuring that no prefetches are pending. If $\overline{\text{BR}}$ or $\overline{\text{TS}}$ is asserted by the processor, the controller must delay asserting $\overline{\text{QACK}}$ until the prefetch has completed and the processor issues no more bus requests. While $\overline{\text{QREQ}}$ is asserted, $\overline{\text{QACK}}$ may be safely asserted if two bus cycles have passed where the processor did not assert $\overline{\text{BR}}$ or $\overline{\text{TS}}$.

Work-Around:

Apart from never enabling prefetching, no absolute method exists to ensure that all L2 hardware prefetches are complete before setting MSR[POW]. However, by using the recommended code sequence below, the chance that a prefetch will exist in the processor after $\overline{\text{QREQ}}$ is asserted will be limited to certain corner cases.

As a software workaround, the following psuedo-code sequence is recommended before entering nap/sleep:

```
mtspr msscr0 # disable-prefetcher
sync
isync
<flush the pipeline># see below
sync
```

```
mtmsr # msr value w/ POW set
isync
```

If L2 hardware prefetching has been enabled in a system (MSSCR0[30–31]), then it must be manually disabled by a mtspr instruction before setting MSR[POW]. After disabling the prefetcher, there will be a maximum of three prefetches queued in the bus load queue awaiting an address tenure. Disabling of the prefetcher must then be followed by 'flush-the-pipeline' code.

Any load operation that is guaranteed to access the external bus (a cache-inhibited load, for example), executed after prefetching is disabled and before MSR[POW] is set, can be used to improve the odds that the L2 hardware prefetches have completed their address tenures. Since the load queue is an ordered queue, the sync that precedes setting of the MSR[POW] will ensure that the pipeline-flushing load has completed before the sync can complete. Thus, the L2 hardware prefetches will be completed before the pipeline-flushing load.

The sync instruction alone is not of use as a pipeline-flushing operation in this context because a sync always has higher internal bus arbitration priority than L2 hardware prefetch operation. Additionally, the prefetches are not required to be completed as a condition for completing the sync instruction.

Other store-type operations can be used as a pipeline-flushing operation, however. For example, a dcbf instruction executed after disabling L2 hardware prefetching will be queued in the store queue. Because the store queue normally has lower internal bus arbitration priority than the load queue, the prefetches in the load queue will win arbitration to the system bus first. The caveat with using a store-type operation as a pipeline-flushing operation is that the store queue is guaranteed to win arbitration to the external bus once it has been bypassed by three store operations. This count is initialized only at reset and, because it is impossible to determine how many times the store queue may have lost arbitration to the load queue, it may be in any state when performing a single pipeline-flushing operation. If three loads have already bypassed stores, for example, the dcbf may immediately win arbitration over any queued L2 hardware prefetches. Therefore, four dcbf instructions are recommended because, barring retries, this will guarantee that all L2 hardware prefetch operations have completed their address tenures.

As mentioned, however, corner cases do exist. If an L2 hardware prefetch bus request is retried, then the retried prefetch will be queued behind the pipeline-flushing load, perhaps stranding the prefetch. Also, if a prefetch request is retried, then this is considered an arbitration attempt for the purposes of counting the number of times that loads have bypassed stores. Therefore, the store unit may be given higher priority in arbitrating for the system bus (if the load queue has lost arbitration to the store queue three consecutive times), allowing the store-type instruction (for example, **dcbf** from previous example) to access the bus before a prefetch. As a result, a pathological retry sequence can be imagined such that no combination of store-type requests can guarantee that the L2 hardware prefetches have completed their address tenures before MSR[POW] is set.

Additional caution should be exercised when using any store-type operation as a pipeline-flushing operation when either M = 0, or HID1[ABE] and/or HID1[SYNCBE] are cleared, as these may not cause an actual address tenure to be performed and the pipeline to be flushed.

Projected Solution:

Under review.

Errata No. 58: $\overline{\text{MCP}}$ signal assertion with MSR[ME] = 1 does not set SRR1[12]

Description:

When a machine check exception is taken due to the assertion of the external $\overline{\text{MCP}}$ signal, SRR1[12] should be set to indicate that $\overline{\text{MCP}}$ was the cause of the exception. It is not set.

SRR1[0–15] record the result of a machine check exception when that exception is taken and machine check exceptions are enabled via the MSR register (MSR[ME] = 1). The setting of the attribute bits lets an exception handler either attempt to recover from the exception or log the error type. If MSR[ME] = 0, these attribute bits do not get set. This is normal behavior.

SRR1[12] should be set by the hardware if the external signal $\overline{\text{MCP}}$ has been asserted for a minimum of two cycles with machine checks enabled. However, the processor does not set this bit.

Projected Impact:

Systems using the external $\overline{\text{MCP}}$ signal and utilizing the software that expects SRR1[12] to be set as a result of the assertion of the $\overline{\text{MCP}}$ signal will not function correctly.

Work-Around:

All other machine check types have a unique attribute bit in SRR1[0:5,7:15]. If SRR1[0:5,7:15] is zero after a machine check occurred with MSR[ME] = 1, then the exception must have been caused by the assertion of the MCP pin.

Note that SRR1[6] is loaded with the equivalent MSR[VEC] bit. For forward compatibility, if MSR[VEC] is enabled, Boolean AND SRR1 with 0xfdf7000. If result is zero, then the exception must have been caused by the assertion of the MCP pin.

Projected Solution:

Under review.

Errata No. 59: A tlbli instruction may cause an incorrect instruction translation

Description:

A **tlbli** instruction may cause the hardware tablewalk engine to return an incorrect result for a subsequent instruction hardware tablewalk.

If a **tlbli** instruction is executed either non-speculatively or speculatively by the load/store unit, and the next operation executed by the load/store unit is an instruction hardware tablewalk, then the instruction hardware tablewalk may fetch an incorrect result. The incorrect result will be loaded into the instruction TLB (iTLB). The subsequent look-up of the iTLB may cause either an instruction fetch to an unknown address, or an ISI exception due to unknown page access control bits.

For the non-speculative execution case, an example code sequence that could cause the error is as follows:

```
.   # Initial setup: MSR[IR]=0; HID0[STEN]=1
tlbli # Load the itlb
mtspr hid0# Disable software tablewalks

mtspr srr0# address of page requiring hardware tablewalk

mtspr srr1# prepare to set MSR[IR]=1 (enable translation)
rfi # return-from-interrupt w/ hardware tablewalks
```

In this example, (1) MSR[IR] is initially cleared and HID0[STEN] is set, which is a typical scenario for when **tlbli** is being used; (2) the **tlbli** is executed correctly, but a residual state is left in the processor that may corrupt a subsequent instruction hardware tablewalk; (3) software tablewalks are then disabled; and (4) the state of the machine is then reloaded via **rfi** such that instruction translation is enabled and hardware tablewalks are enabled.

If the target of the **rfi** requires an instruction hardware tablewalk, then that instruction hardware tablewalk may be corrupted due to the residual state left by the **tlbli**. Note that this scenario is unlikely since software and hardware tablewalks are rarely mixed in this manner.

For the speculative execution case, an example code sequence that could cause the error is as follows:

```
.   # MSR[IR]=1; HID0[STEN]=0
bcc # branch taken

.
tlbli# in speculative non-taken path
```

In this example, (1) instruction translation is enabled and software tablewalks are disabled; (2) the branch, either conditional or unconditional is taken, where the target of the branch requires an instruction hardware tablewalk; and (3) the **tlbli** instruction is in a window of less than 16 instructions (the depth of the completion buffer) past the branch.

If the **tlbli** instruction is speculatively executed before the instruction hardware tablewalk caused by the branch, then the **tlbli** may leave a residual state, as with the non-speculative execution case, that will cause the instruction hardware tablewalk to fail.

Other detailed architectural timings in the load/store unit must align properly for this scenario to fail and the risk to a general system is considered to be small.

Projected Impact:

If system software does not use any **tlbli** instructions, then the software can avoid this erratum by guaranteeing that no **tlbli** instruction will be encountered on a speculative non-taken branch path. Note that ‘encountering’ a **tlbli** instruction includes either data, code, or uninitialized memory that decodes to a **tlbli** instruction. It is, therefore, possible to experience errors as a result of this erratum in any system using hardware tablewalks, even if neither software tablewalks nor **tlbli** instructions are ever used.

System software using **tlbli** instructions is more susceptible to this erratum if a **tlbli** instruction is near where a hardware tablewalk may occur. The probability of either of the previous code scenarios appearing in software is low. The probability of a random data or uninitialized memory decoding to a **tlbli** instruction is likewise remote.

Work-Around:

Any one of the following is sufficient to prevent the failure condition:

- Do not use **tlbli** instructions, even in non-taken branch paths.
- Place two **eieio** or two **sync** instructions before, and two **eieio** or two **sync** instructions after the **tlbli** instruction.
- Use software tablewalks only.
- Do not enable instruction translation.

Projected Solution:

Under review.

Errata No. 60: AltiVec™ vmaddfp or vnmsubfp can incorrectly return zero

Description:

Execution of either the Multiply-Add Float (**vmaddfp**) or the Vector Negative Multiply-Subtract Float (**vnmsubfp**) instructions will provide only 45 bits of intermediate precision, instead of the expected 46 bits of precision, when encountering certain combinations of floating-point mantissas. This loss of precision will cause an unexpected zero result.

The AltiVec instructions **vmaddfp** and **vnmsubfp** produce a 46-bit intermediate significant prior to rounding it to a 24-bit significant. Due to a problem with the normalizer unit, the 46-bit intermediate significant only achieves 45-bit precision for certain combinations of operands.

For the fail to occur, the terms presented to the AltiVec floating-point adder must meet certain conditions. Specifically, all of these conditions must occur:

1. The two terms for the adder, B and $(A \cdot C)$, must have opposite signs, hence, causing an effective subtraction.
2. The two terms must have the same exponent.
3. The absolute value of term B must be slightly larger than the absolute value of term $(A \cdot C)$; that is, $|B| > |A \cdot C|$.
4. The effective subtraction of the two terms must result in the smallest non-zero number possible.

An example failing scenario is as follows:

```
vmaddfp    V3,V0,V1,V2
```

where:

```
V0 = 3FBFFFFF3FBFFFFF3FBFFFFF3FBFFFFF
V1 = 3FC000013FC000013FC000013FC00001
V2 = C0100000C0100000C0100000C0100000
```

will result in:

```
V3 = 00000000000000000000000000000000
```

when the result should be:

```
V3 = A8800000A8800000A8800000A8800000
```

When this case occurs, the amount of the result error will always be $2^{(B-46)}$. The worst case is 2^{81}, corresponding to the case when the B exponent is at its maximum value of 2^{127}.

Note that even with the lost precision due to this erratum, the **vmaddfp** and **vnmsubfp** instructions provide no less precision than if the multiply and add/subtract instructions were executed as two discrete instructions.

Projected Impact:

Applications using either the **vmaddfp** or **vnmsubfp** instructions will only generate 45 bits of precision for the intermediate $(A \cdot C)$ product in some cases.

Work-Around:

For applications that require 46 bits of precision for the vector multiply-add instructions, the equivalent non-vector floating-point instructions must be used.

Projected Solution:

Under review.

Errata No. 61: Back-to-back L2 allocation causes lost data

Description:

If two L1 data cache (dL1) load miss requests differing by only physical address bit 30 and a dL1 castout to the same address as one of the load requests arbitrate for the L2 cache in a unique time sequence, the dL1 castout data may be lost.

The sequence of L2 arbitration requests required for failure is as follows:

1. Cycle N: a dL1 load miss request to address A arbitrates for the L2 cache. The request for A misses in the L2 cache and chooses a victim way with no sectors modified. The read may either hit in the L3 (if enabled) or be sent to the bus (if L3 miss or L3 disabled).
2. Cycle N+2: a dL1 load request to A's alternate sector arbitrates for the L2. The request for the alternate sector misses in the L2 cache and chooses a victim way with one or two sectors modified. The alternate sector read may also either hit in the L3 (if enabled) or be sent to the bus (if L3 miss or L3 disabled).
3. Cycle N+3: the first load miss allocates into the L2 cache.
4. Cycle N+4: a dL1 castout to address A arbitrates for the L2, hits in the allocated way, and writes modified data into the L2 cache.
5. Cycle N+5: an L2 castout caused by the second load miss arbitrates for the L2 pipeline going into the L2 castout queue.
6. Cycle N+6: a second L2 castout caused by the second load miss may arbitrate for the L2 pipeline going into the L2 castout queue.)
7. Cycle N+6 or N+7: the second load miss allocates in the L2 cache. The load miss also mistakenly sets to invalid all sectors of the matching L2 line, thereby invalidating the modified state of address A and causing A's data to be lost.

Projected Impact:

The alignment of internal arbitration timings required to cause a failure is extremely rare but possible, and has occurred in a system.

Work-Around:

The following software workarounds have been identified, applicable to all MPC7450-family devices:

1. If instruction and data address spaces do not overlap, setting the L2CR[L2IO] bit will prevent modified data from being lost by preventing data from being allocated in the L2.
2. If instruction and data address spaces do not overlap, setting bit 6 of SPR 1012 to 1 and disabling the L2 prefetch engine (MSSCR0[L2PFE] = 0b00), will prevent modified data from being lost by serializing load and cacheable store requests to the MSS while still allowing pipelined dL1 hits. Note that SPR 1012 is an undocumented, factory-use only register and other bits in this register must not be modified.
3. Setting a page to write-through-required (W = 1) or cache-inhibited (I = 1) will prevent modified data from that page from being lost by preventing the data from using the modified cache state.
4. Inserting a **sync** instruction between stores and subsequent loads to the same cache line will prevent modified data from the store from being lost by preventing the loads from requesting L2 arbitration while a dL1 castout is pending.

Projected Solution:

Under review.

Errata No. 62: Six outstanding miss requests may stall the processor

Description:

A processor configured with 2M of L3 cache that executes a sequence of heavy castout traffic to the external bus, followed by six cacheable L1 cache miss requests (instructions or data) that each require victimizing four modified sectors from the L3, may enter a hang state until an external snoop clears the hang.

The sequence of events required for this fail are as follows:

1. The processor is configured with 2M of external cache.
2. Heavy internal castout traffic occurs such that the 9-entry L3 castout queue is full. No load miss operations are pending.
3. Five new cacheable load miss or instruction fetch requests and one cacheable store miss request all miss in both the L2 and L3 caches while the castouts to the bus are pending. All six are retried internally due to the castout queue full condition.
4. Each of the six miss requests randomly choose a victim way in the L2 which requires no castout. That is, no sectors of any of the 6 chosen victim L2 cache lines (a total of 12 sectors altogether) are modified.
5. Each of the six miss requests choose a victim way in the L3, which require four castouts, one for each of the four L3 sectors in the 2-Mbyte mode. That is, every sector of the 6 chosen victim lines (a total of 24 sectors altogether) are modified.
6. The miss requests enter an arbitration harmonic, wherein even after the L3 castout queue drains, allowing the misses to arbitrate for the bus, the miss requests continue to be retried due to an L3 SRAM interface resource counter which incorrectly signals a queue full condition.

The resource counter will continue to report the queue full condition until an external snoop gives the counter the one cycle it requires to clear. Note that the above sequence of events must occur without interruption; an unrelated load or store miss (that is, one that does not meet the criteria stated above) will cause the processor to exit the sequence and prevent the stall condition from occurring. As a result, the stall condition is extremely rare but has been observed in a system.

Projected Impact:

Systems will typically only enter this mode if the system bus becomes congested with store traffic. The stall condition is not a permanent hang state unless snoops never occur once the condition is encountered.

Work-Around:

Two software workarounds and one hardware workaround exist:

- Operate in 1 Mbyte L3 cache-only mode or 1 Mbyte cache + 1 Mbyte private memory mode.
- Operate the L3 in instruction-only mode (L3CR[L3IO] set)
- Design the system such that periodic snoop traffic is guaranteed to occur.

Projected Solution:

Under review.

Errata No. 63: dcbt or dcbtst to protected space may not no-op

Description:

A **dcbt** or **dcbtst** instruction with an address translation that is mapped as protected in either the DBATs or the DTLB should no-op. A one-cycle window exists where the instruction does not no-op, and a request is incorrectly sent from the load/store unit to the memory subsystem.

The **dcbt** and **dcbtst** instructions are defined as being treated as a no-op condition for the following cases:

- The access causes a protection violation
- The page is mapped cache-inhibited or direct-store (T = 1)
- The cache is locked or disabled
- Software table searching is enabled (HID0[STEN] = 1) and the access misses in the DBATs and dTLB
- HID0[NOPTI] = 1

For the protection violation case only, a one-cycle window exists where if an unrelated data reload occurs to the dL1 from the memory subsystem, and the address of that reload is to the same cache index as the **dcbt**/**dcbtst**, then the **dcbt**/**dcbtst** will not correctly no-op. Instead, a touch (for **dcbt**) or touch-for-store request (for **dcbtst**) will be issued to the memory subsystem using the address from the DBAT or DTLB translation. If the request misses in the L2 and L3, either a READ or an RCLAIM request will be issued on the external bus. The data to the protected address will then be reloaded into the L1 data cache.

For all other listed cases, **dcbt** and **dcbtst** instructions will be correctly treated as no-op conditions.

Note: The *MPC7450 RISC Microprocessor Family User's Manual* states that a **dcbtst** instruction will be treated as a no-op if the target address of the **dcbtst** is mapped as write-through. This is an incorrect statement. A write-through **dcbtst** will result in an RCLAIM (or a READ in 60x bus mode) on the external bus if it misses in the cache hierarchy.

Projected Impact:

Systems executing **dcbt** or **dcbtst** instructions with addresses that map to protected space using the DBATs or the DTLB may see accesses to undesired addresses on the external bus.

Work-Around:

Two workarounds exists:

- Do not execute a **dcbt** or **dcbtst** instruction to an address space that is protected; *or*
- Set HID0[NOPTI], which correctly no-ops all **dcbt** and **dcbtst** instructions regardless of address translation.

Projected Solution:

Under review.

Errata No. 64: PMC2[32] does not increment correctly

Description:

PMC2[32] should count the number of cycles when the number of valid entries in the 16-entry completion queue is greater than or equal to a programmed threshold. The counter does not increment correctly any time the number of valid entries is 8 or 16.

PMC2[32] should count the number of cycles when the number of valid entries in the 16-entry completion queue is greater than or equal to the value programmed in MMCR0[THRESHOLD].

The performance monitor does not correctly increment if the number of entries in the completion queue is exactly eight. No increment is performed. Therefore, if MMCR0[THRESHOLD] is set to any value between zero and eight inclusive, the counter will be *X* less than the correct value, where *X* is equal to the total number of cycles in which there were exactly eight valid completion queue entries.

The performance monitor also does not correctly increment if the number of entries in the completion queue is exactly 16 (full). Sixteen entries appears to the counter logic as eight entries. Therefore, if MMCR0[THRESHOLD] is set to any value between 9 and 16, the counter will be *Y* less than the correct value, where *Y* is equal to the total number of cycles in which there were 16 valid completion queue entries.

Projected Impact:

Performance analysis using PMC2[32] will receive low counts equal to the number of cycles in which the number of valid completion queue entries was either 8 or 16.

Work-Around:

None

Projected Solution:

Under review.

Errata No. 65: Multi-cycle $\overline{\text{TS}}$ assertion in COP soft-stop debug mode

Description:

During soft-stop debug mode, accessible via the common on-chip processor (COP), the processor may assert $\overline{\text{TS}}$ on the system bus for multiple bus clocks, thereby violating the 60x and MPX bus protocols.

Emulator tools use soft-stop regularly for debugging purposes. Soft-stop is configured via the COP service register interface.

The COP can request that the processor enter soft-stop either by **(A)** issuing a soft-stop command, or by **(B)** informing the processor that it should soft-stop due to any of the following four conditions:

1. An instruction address breakpoint register (IABR) match.
2. A data address breakpoint register (DABR) match.
3. A trace interrupt condition caused by setting either MSE[SE] or MSR[BE].
4. A performance monitor interrupt condition.

When a softstop condition occurs, the processor will wait for all memory subsystem traffic to quiesce, the processor will inform the COP that traffic is quiesced, and the COP will then shut down clocks to the processor. After clocks have been shut down, the COP can perform activities such as dump the internal caches or check the state of the architected registers.

The processor will perform type "**(A)**" soft-stop actions correctly. However, soft-stop type "**(B)**" actions do not always correctly wait for instruction fetches to complete. As a result, during soft-stop sequencing, $\overline{\text{TS}}$ can be asserted on the external bus to initiate an instruction fetch and the processor clocks can be shut down during the $\overline{\text{TS}}$ assertion. The processor does recognize that $\overline{\text{TS}}$ is asserted and the fetch is in progress, and the clocks are re-started, but not before $\overline{\text{TS}}$ is asserted for multiple bus clocks, which is a violation of the 60x and MPX bus protocols.

If the system controller ignores the protocol violation and returns instruction data for the instruction fetch, the processor will then correctly enter soft-stop mode and the clocks will be shut down as expected. If the processor's protocol violation produces no unexpected system controller behavior, debug using soft-stop may be possible. However, special care must be taken in the cases of system controllers that park $\overline{\text{DBG}}$ to the processor and negate $\overline{\text{DBG}}$ the cycle following the detection of $\overline{\text{TS}}$ being asserted. The processor normally begins sampling $\overline{\text{DBG}}$ during the cycle $\overline{\text{TS}}$ is asserted. However, because the processor's clocks are disabled during all but the last cycle of the multi-cycle $\overline{\text{TS}}$ assertion, the processor will not be able to sample $\overline{\text{DBG}}$ correctly until the final cycle of $\overline{\text{TS}}$ assertion. Therefore, a system controller parking $\overline{\text{DBG}}$ in this manner must drive it for the entire multi-cycle assertion of the processor's $\overline{\text{TS}}$.

Only soft-stop debug mode via the COP is affected. Using the IABR/DABR and taking a trace or performance monitor interrupts work as expected in functional mode. The $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ nap/sleep protocol also works as expected.

Projected Impact:

Emulators that utilize the COP soft-stop feature may not work as intended since the processor's violation of the system bus protocol may cause unexpected behavior on the part of the system controller.

Work-Around:

None.

Projected Solution:

Under review.

Errata No. 66: Snoop during L2 hardware flush can lose modified data

Description:

An external snoop during an L2 hardware flush that collides with a cache line that is being flushed from the L2 may not receive a retry response on the external bus, leading to possible data corruption.

If a line exists as modified in the L2, and is not modified in either the L1 data cache or L3, and the L2 hardware flush engine internally casts out the modified data (moving it from the L2 cache to the L2 Castout Queue) the same cycle in which internal queues are checked against a pending external snoop, the processor may not recognize the address collision and will not correctly assert address retry on the system bus. Meanwhile, because the processor did not respond with an address retry response, the snooping entity may assume ownership of the line and modify the data. When the processor eventually writes the data to the bus, the cache line is corrupted with the stale data from the processor and the data modified by the snooping entity is lost.

This issue exists in both 60x and MPX bus modes.

Neither the L2 hardware invalidate or the L3 hardware flush/invalidate operations are impacted by this erratum.

Projected Impact:

Systems with snooping activity while a processor is performing an L2 hardware flush may see data corruption.

Work-Around:

Avoid issuing transactions that must be snooped by the processor ($\overline{\text{GBL}}$ asserted) during the L2 hardware flush.

Projected Solution:

Under review.

Errata No. 67: Processor does not correctly single step lmw/stmw/lsw/stsw instructions in COP debug mode

Description:

During softstop debug mode, which is accessible through the COP, the processor does not correctly single step a multiple or string word operation type.

All segments of a Load Multiple Word (**lmw**), Store Multiple Word (**stmw**), Load String Word (**lsw**), or Store String Word (**stsw**) instruction should be performed before a softstop is taken for that instruction type during COP single-step mode. The program counter (PC) register should point to the next instruction.

The processor incorrectly stops executing the instruction after performing only the first segment of the operation. The PC remains as the address of the instruction itself.

During normal functional mode, all segments of these instruction types will be correctly executed before the trace exception is generated when MSR[SE] = 1.

Projected Impact:

Emulators that utilize the COP soft-stop feature may not work as intended when attempting to single step a multiple or string word operation.

Work-Around:

An emulator should not attempt to single step an operation of this type. A breakpoint can be successfully set to the address of the instruction following the multiple or string word operation.

Projected Solution:

Under review.

Errata No. 68: SRR0/SRR1/PC values incorrectly updated if instruction at breakpoint in COP debug mode causes an exception

Description:

During softstop debug mode, which is accessible through the COP, the processor corrupts the values of the SRR0/SRR1/PC if an instruction that matches IABR causes an exception of lower priority than a breakpoint exception.

If the instruction address breakpoint register (IABR) is set and enabled for an instruction address in COP softstop mode, the processor should halt operation and stop internal clocks before the instruction is executed. The PC should point to that instruction's address.

However, the processor may incorrectly update the PC if the instruction at that address causes an exception of lower priority than an IABR breakpoint. The SRR0 is incorrectly updated to the address of the instruction itself, and the SRR1 is modified.

For example, if the IABR points to an instruction that performs a data access to an address space that is mapped no-access in a DBAT, the PC is updated to the DSI exception handler address, and the SRR0 is updated to the address of the excepting instruction.

Other exception types that cause this failure are illegal instruction exceptions, traps, floating-point or Altivec unavailable exceptions, privilege violations, and alignment exceptions.

Data translation that hits in the on-chip DTLB as an access violation will also show this problem. Translation that misses in the DTLB and requires a tablewalk will not show this problem because the tablewalk will be correctly terminated before any page table results return.

Instruction address breakpoints set through software during normal functional mode stop correctly before any PC/SRR0/SRR1 update occurs.

Projected Impact:

Emulators that utilize the COP soft-stop feature may not work as intended when attempting to set a breakpoint at an instruction that causes an exception.

Work-Around:

None.

Projected Solution:

Under review.

Errata No. 69: Data corruption with M=0 and L2 prefetching enabled.

Description:

If memory is mapped as non-coherent (M=0), and L2 hardware prefetching is enabled, data corruption may result. Corruption may also result with memory mapped non-coherent if data and instruction address spaces overlap or are adjacent.

The L2 hardware prefetcher performs an internal snoop to the L1 data cache (dL1) before making a request to the L3 or the external bus for an address. If the dL1 has the cache line for that address modified, the dL1 cache state for that address transitions to a shared state, and an internal operation is created to transfer the data to the L2 for the prefetch request. This internal transfer, or 'local intervention', requests arbitration for the L2 like all other requests, and can be stalled, for example, if the L2 castout queue is full.

If, for non-coherent memory, a cacheable store hits in the dL1 to this address in the shared state, the data in the dL1 will be updated, and the cache state will transition from shared to modified state. If this modified data is subsequently evicted from the dL1 either due to a reload replacement or a flush operation, the resulting castout will also arbitrate for the L2 cache.

The local intervention with stale data may be stalled awaiting access to the L2 cache when the castout with modified data also requests access to the L2. Since the arbitration policy between these two entities is round-robin, the castout may actually win arbitration to modify the L2 cache before the local intervention (or send data to the bus or L3 if the L2 is disabled). The local intervention will later win L2 arbitration and write stale data to the L2 (or the L3 or external bus if the L2 is disabled).

Memory coherent (M=1) accesses will not cause a fail because the store to the dL1 will not incorrectly allow the store to transition the dL1 state to modified. Instead, it will make a cacheable store request to the L2 cache. Cacheable load or store requests are not allowed to bypass a castout to the same address, guaranteeing no loss of data.

The loss of data can also occur if instructions and data share the same or adjacent cache lines that are also mapped as M=0. Instruction fetches snoop the dL1 in a similar fashion as prefetches. If the data can be in a modified state, the same data loss situation may occur.

Note: This erratum will occur whether the L2 is enabled or disabled because the L2 prefetch engines are enabled independently of the L2.

Projected Impact:

Systems with software mapping memory as non-coherent and enabling L2 hardware prefetching may see loss of data.

If instructions and data share the same or adjacent cache lines when mapped M=0, loss of data may occur.

Work-Around:

Disable L2 hardware prefetching when using non-coherent memory.

Also, do not permit instructions and data to share the same or adjacent cache lines if mapped M=0.

Projected Solution:

Under Review.

Errata No. 70: Cacheable load/store during L2 hardware flush can lose modified data

Description:

In the failing scenario, a flush of the caches starts as described in the section “Flushing of L1, L2, and L3 Caches,” of the “L1, L2, and L3 cache operation” chapter in the *MPC7450 RISC Microprocessor Family User’s Manual*. While the L2 hardware flush is being performed, a cacheable load or store misses in the L1 data cache and requests data from the L2 cache. If this occurs during the same cycle in which the same address is transferred to the L2 castout queue (as a result of the L2 hardware flush), the miss request will incorrectly initiate a read or read-with-intent-to-modify (rwitm) operation on the external bus. If the read or rwitm operation accesses memory before the castout of the modified data is stored to memory, data corruption may result.

Projected Impact:

If the recommended code loop for flushing the L2 cache is used, it keeps the cacheable loads or stores in a sequential code stream from requesting access to the L2 during the flush. However, an interrupt or exception taken during the L2 hardware flush may have an interrupt handler that makes a data miss request. If the recommended L2 hardware flush routine is not used, or a system encounters interrupts during this routine, data corruption could occur.

Because the L2 hardware flush engine has higher internal arbitration priority to the L2 cache than either L1 instruction or data cache miss requests, miss requests rarely win L2 arbitration during this time. (Windows are opened into which miss requests can arbitrate for the L2 cache only when the L2 and L3 castout queues fill with modified data flushed from the L2 and the external bus can not drain the queues as quickly as the lines are flushed.) As a result, an interrupt handler is not likely to make much progress, and encountering this error is considered unlikely.

Note: Neither the L2 hardware invalidate nor the L3 hardware flush/invalidate operations are impacted by this erratum.

Work-Around:

To guarantee no data corruption during an L2 hardware flush sequence, interrupts must be disabled and the following software flush routine must be used. The software routine, or a variant, waits for the flush to finish and does not perform other loads/stores during this time. This sequence flushes only the L2 but does conform to the requirements recommended in the section “Flushing of L1, L2, and L3 Caches,” of the “L1, L2, and L3 cache operation” chapter in the *MPC7450 RISC Microprocessor Family User’s Manual*:

```
# Flush the L2
     mfspr   r1,l2cr
     oris    r1,r1,0x0011    # Make it IONLY/DONLY
     ori     r1,r1,0x0800    # Set the flush bit
     mtspr   l2cr,r1
```

```
# Wait for the invalidate to finish
poll_l2:
      mfspr   r1,l2cr
      andi.   r1,r1,0x0800    # Check to see if we’re done
      bne     poll_l2           # try again
      sync
      isync
```

NOTE

Because software flush routines are generally ineffective for the L2 cache due to the random replacement algorithm., the L2 hardware flush mechanism is the only reliable way to flush the L2 cache.

Projected Solution:

Under review.

Errata No. 71: Data parity errors not checked during L2 hardware flush

Overview:

During an L2 hardware flush, every index, sector, and way of the L2 cache is sequentially invalidated. All modified lines are flushed from the cache as individual castout operations.

If the modified data and parity read from the L2 cache during the flush do not match, an L2 data parity error should be flagged but is not.

L2 data parity is correctly checked for all other L2 read accesses including load hits, castouts due to either replacement or a dcbf, and pushes or interventions due to external snoops.

The L2 tag is correctly checked for parity errors during an L2 hardware flush. The L3 tag and L3 data are correctly checked for parity errors during an L3 hardware flush.

Projected Impact:

Any L2 data cache lines with an incorrect bit that would normally be flagged as either a machine check exception or a checkstop will be passed to the L3 or external bus during an L2 hardware flush. L2 parity errors are not expected to occur during normal system operation.

Work Arounds:

None.

Projected Solution:

Under review.

Errata No. 72: Processor may hang when mtmsr/isync transitions MSR[IR] from 1->0 and isync instruction resides in unmapped page

Overview:

If the mtmsr instruction is used to disable instruction translation, and the subsequent isync instruction resides on a different page than the mtmsr instruction, and the isync instruction’s page causes a page fault exception, the processor will hang before taking the page fault exception. Once in the hang state, the processor will not recover and hard reset must be asserted.

An alternate failing scenario can exist if the mtmsr and isync reside on the same page but in different quadwords. If the mapping for that page exists in the iTLB, but not the page table, and a tlbie snoop occurs between the mtmsr and isync that invalidates the iTLB entry, then the processor will hang before taking the page fault exception.

If the isync instruction address is guaranteed to have a valid page table mapping resident in the memory hierarchy, then neither scenario can occur.

Projected Impact:

Any systems that disable instruction translation using mtmsr, and for which the required isync may reside in a different page whose page table entry is not available in the memory hierarchy may hang.

Any systems mapping the mtmsr/isync code with the IBATs will not fail due to this issue. Any systems not demand-paging their supervisor-level code will not fail due to this issue.

Work Arounds:

Any of the following work-arounds will avoid the processor hang:

- mtmsr/isync instruction pairs should reside within the same quadword.
- disable instruction translation by initializing SRR0/SRR1 and executing rfi.
- map code which disables instruction address translation with the IBATs.

Projected Solution:

Under review.

Errata No. 73: Scanning MSS_NRM chain via COP during softstop may cause unexpected interrupts upon resumption of normal operation

Overview:

If the MSS_NRM scan chain is accessed via COP during softstop and MSR[EE] is set, then an unexpected decrementer (0x0900) or thermal management (0x1700) interrupt may occur when the processor resumes from softstop.

The following special purpose registers reside in the MSS_NRM scan chain:

1. HID1
2. L2CR
3. L3CR/L3PM/L3ITCR[0-3]/L3OH (L3ITCRs[1-3]/L3OH exist in MPC7457 only)
4. MSSCR0
5. MSSSR0
6. TBL/TBU
7. DEC
8. THRM1/THRM2/THRM3 (exist in MPC7451/MPC7445/MPC7455 only)

Accessing any of the above registers may trigger the unexpected interrupt.

The unexpected thermal management interrupt will only occur in the MPC7451 and MPC7455 processors.

The unexpected decrementer interrupt may occur in all processors of the MPC7450 family.

Projected Impact:

Emulators and debug tools accessing the MSS_NRM scan chain via COP during softstop.

Work Arounds:

Use LSRL to access any elements in the MSS_NRM scan chain including the SPRs listed above.

Projected Solution:

Under review.

Errata No. 74: tlbie snoop during Doze state may cause processor hang

Overview:

If the instruction TLB entry for the code that caused entry into Nap mode is invalidated while in Doze state via a snooped tlbie, an instruction tablewalk will be immediately triggered for that instruction address. Taking the tablewalk is an incorrect action for the processor because no code is being executed during Doze state.

A processor hang can occur due to the unintentional tablewalk if the tablewalk results in an ISI exception, most likely due to a page fault. When the page fault is detected, the SRR0, SRR1, and MSR registers are updated as for a normal ISI exception except that the ISI handler code is not fetched because of the Doze state. Since the MSR[EE] External Interrupt Enable bit is cleared due to the ISI exception, neither an external interrupt or a decrementer interrupt can wake the processor from Nap mode. A processor hang results.

If the unintentional instruction hardware tablewalk had completed without an exception, the MSR[EE] bit would not be cleared and the processor can be woken from Nap mode by an interrupt. However, system designers should use caution since the processor is only ever expected to provide snoop responses and/or push or intervention data during Doze state. Due to the unintentional tablewalk, up to four tablewalk read requests may occur to the external bus. The system would either have to service these read requests before re-entering Nap mode, or somehow ignore pending bus request assertions for these reads.

Projected Impact:

Systems performing tlbie snoops during Doze state where the index of the tlbie matches that of the instruction code that caused entry into Nap mode may hang.

Systems using the iBATs to map the Nap mode code are not affected.

Since a tlbie snoop will only likely occur in multiprocessor systems, most uniprocessor systems should not be affected.

Work Arounds:

Any of the following work arounds are acceptable:

1. Do not perform tlbie snoops during Doze state.
2. Disable instruction address translation before entering Nap mode.
3. Map the code that causes entry into Nap mode using the IBATs.
4. Ensure that the page table entry mapping for the Nap mode code remains valid during Nap mode, perhaps via an OS locking mechanism. Note that the system would still have to handle potential tablewalk read accesses during Doze state.

Projected Solution

Under review.

Errata No. 75: Data corruption due to write-through aliasing in the MMU

Overview:

If the memory management unit (MMU) on the MPC7450 processor is programmed such that the same physical block or page has storage access attributes mapped as both cacheable in one memory mapping and write-though in another, data corruption may result.

Accesses to the same storage location using two effective addresses for which the write-through mode (W-bit) differs meet the memory coherence requirements of a MPX744X processor if the accesses are performed by a single processor. Since the usage of the W-bit is mixed for the same physical address, a store addressed to a write-through page may find the addressed cache block modified in either the dL1 or L2 caches. In this case, the write-through store will update the location in both the cache block and main storage. The cache block remains in the modified state. If a block or page is mapped as write-though only, the cache line will never be modified.

In the failing scenario, address “A”, which is mapped both as cacheable and write-through resides as modified in the dL1 cache. A unrelated dL1 reload to the same set as address “A” occurs. The reload chooses address “A” for eviction due to the pseudo-LRU replacement algorithm. As a castout is being created of address “A”, a write-though store to address “A” erroneously bypasses the castout without updating the data. As a result, the castout with stale data will be written to memory after the write-through store.

Projected Impact:

Systems performing write-through aliasing of the same physical page may encounter data corruption.

Work Arounds:

Do not permit two effective addresses to map to the same physical address where one effective address is mapped as write-though and the other is mapped as cacheable. If write-through aliasing is unavoidable, the dL1 cache must be flushed of its contents by the system when switching from cacheable to write-through accesses.

Projected Solution:

Under review.

Errata No. 76: HID0[SPD]=1 does not prevent instruction fetches past unresolved branches when MSR[IR]=0

Overview:

An out-of-order instruction fetch is defined as an instruction fetch made before it is known whether the instructions are required by the sequential execution model. When instruction translation is disabled (MSR[IR]=0), the accesses are guarded. In general, guarded storage is not accessed out-of-order. Three exceptions to this rule, as defined by the architecture, may cause out-of-order instruction fetches:

1. The instruction is in the instruction cache.

2. The instruction resides in the same physical page as an instruction that is required by the execution model.

3. The instruction resides in the next sequential physical page after the page of an instruction that is required by the execution model.

Therefore, guarded out-of-order instruction fetches may occur to a page for which there are no executable instructions as long as the previous page contained instructions required by the sequential execution model.

Typically, system software will contain a branch near the end of the second to last page of a memory region and leave the last page empty but valid. For example, a hypothetical system with one megabyte of memory will place no executable code in the last 4096 bytes of the one megabyte of memory space to avoid instruction fetches on the external bus interface beyond the one megabyte of valid physical memory.

The HID0[SPD], “Speculative data cache and instruction cache access disable”, is designed to prevent out-of-order instruction cache fetches beyond an unresolved branch and into a subsequent page from propagating to the external bus interface. By definition, setting HID0[SPD]=1 should permit a final branch instruction to exist in the very last word of a physical memory region.

However, the MPC7450 processor with HID0[SPD]=1 will incorrectly fetch past an unresolved branch as with HID0[SPD]=0, and perhaps create an external bus read operation to an invalid region of memory.

HID0[SPD] works as designed for load operations.

Projected Impact:

Systems executing code with instruction translation disabled (MSR[IR]=0) and HID0[SPD]=1 may see instruction fetches propagate to the external bus unexpectedly.

Work Arounds:

System software should not place the last branch instruction at the end of a physical memory region in the last two cache lines of that memory region when MSR[IR]=0 and HID0[SPD]=1. If the branch is placed in the third to last cache line, accesses may still occur to the external bus for the final two cache lines, but no accesses will occur to the subsequent page.

The same restriction actually also holds when HID0[SPD]=0. Although architecturally the processor is permitted to access instructions out-of-order and into the next page past what is required by the sequential execution model, the processor will not do so if the final branch is not in the last two cache lines of the last page required by the sequential execution model.

Projected Solution:

Under review.

Errata No. 77: Unexpected instruction fetch may occur when transitioning MSR[IR] 0->1

Overview:

An mtmsr/isync pair followed by a blr, bctr, or branch absolute instruction, where the target effective address is not equal to the target physical address, may cause an unexpected instruction fetch on the MPX or 60x bus using the effective address.

The effect of the unexpected instruction fetch will be system dependent. If physical memory exists at the unexpected address, and read accesses are permitted, the processor will simply discard any returned instructions after the data tenure is complete, and no fail will occur. If a TEA_ is asserted for the data tenure the processor will take either an unexpected machine check exception or checkstop.

The unexpected instruction fetch issue is independent of translation method, and thus can occur for either iBATs or page tables.

Projected Impact:

Systems executing code as described above may encounter unexpected machine checks or system hangs.

Work Arounds:

The recommended workaround is for any target of an unconditional branch in the sixteen instructions following an mtmsr/isync pair where instruction translation is being enabled to have a translation mapping of effective address = physical address.

If such a mapping is not possible, the issue can be avoided if the effective address target matches a valid region of physical memory in the system. As with the previous workaround, the unexpected instruction fetch would still occur, and the instructions would be discarded by the processor.

A third workaround is to only enable instruction translation by initializing SRR0/SRR1 and executing rfi. A fourth workaround that would prevent the unexpected fetch altogether would be to guarantee in software that there were no unconditional branches in the sixteen instructions following the mtmsr/isync pair.

Projected Solution:

Under review.

How to Reach Us:

Home Page:
www.freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130

Europe, Middle East, and Africa:
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120-191014
+81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
852-26668334

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150

Learn More: For more information about Freescale Semiconductor products, please visit
www.freescale.com

MPC7451CE
Rev. 20
9/2005