Document Number: MPC7447ACE
Rev. 9, 10/2007

Chip Errata for the MPC7447A

This document details all known silicon errata for the MPC7447A. The MPC7447A is a PowerPC™ microprocessor. The MPC7450 has the same functionality as the MPC7447A and any differences are detailed in the document. Table 1 provides a revision history for this chip errata document.

Table 1. Document Revision History

Revision Number	Date	Significant Changes
0	—	Initial release of document.
1	05/26/04	Added Erratum 19.
2	09/02/04	Added Erratum 20and Erratum 21.
3	11/09/04	Revised Erratum 5 and Erratum 21. Reformatted document.
		Added Erratum 22.
4	02/15/04	Added Erratum 24 and Erratum 25.
5	05/19/05	Added Erratum 26.
6	09/07/2005	Erratum 26: added “or 60x bus” to the first sentence in “Description.”
		Erratum 9: updated “Workaround.”
7	02/02/2006	Added Erratum 27.
8	08/16/2007	Added Erratum 28.
9	10/26/2007	Added Erratum 29.

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.

freescale™ semiconductor

Table 2. Revision Level to Part Marking Cross-Reference

MPC7447A Revision	Part Marking	Processor Version Register
1.0	A	8003_0100
1.1	B	8003_0101

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. MPC7447A Summary of Silicon Errata and Applicable Revision

No.	Name	Projected Impact Overview	Errata in Silicon Revision	
			1.0	1.1
1	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*.	
2	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 TAU is not implemented as a feature and is not included in any MPC7450 Family documentation.	
3	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*.	
4	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 hardware specifications as the correct way to configure L1_TSTCLK and L2_TSTCLK.	
5	Earliest transfer of MPX/60x data requirement	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.	The MPC7450 family implementation of the earliest transfer for MPX/60x data will remain as documented.	
6	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

Table 3. MPC7447A Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision	
			1.0	1.1
7	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
8	L2 hardware prefetcher not quiesced before $\overline{\text{QREQ}}$ 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
9	$\overline{\text{MCP}}$ signal assertion with MSR[ME] = 1 does not set SRR1[12]	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.	Y	Y
10	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	N
11	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
12	Multi-cycle $\overline{\text{TS}}$ 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	N
13	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	N
14	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

Table 3. MPC7447A Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision	
			1.0	1.1
15	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
16	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
17	A speculative tlbld instruction may cause an incorrect instruction translation	If system software does not use any tlbld instructions, then the software can avoid this error by guaranteeing that no tlbld instruction will be encountered on a speculative non-taken branch path.	Y	Y
18	Standard IEEE 1149.1 (JTAG) BYPASS / SAMPLE / PRELOAD instructions change the state of the processor	Whenever an emulator tool shifts in a standard JTAG BYPASS/SAMPLE/PRELOAD instruction while normal code is running on any system, an undefined behavior occurs.	Y	Y
19	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
20	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
21	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
22	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
23	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

Table 3. MPC7447A Summary of Silicon Errata and Applicable Revision (continued)

No.	Name	Projected Impact Overview	Errata in Silicon Revision	
			1.0	1.1
26	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
27	Performance Monitor Out (PMON_OUT) not operational	Systems designed to use the PMON_OUT signal as an indication of an internal Performance Monitor event will not have this feature, originally intended as a debug aid.	Y	Y
28	$\overline{\text{MCP}}$ or $\overline{\text{SRESET}}$ signal assertion during transition to Nap may hang processor	Systems where $\overline{\text{MCP}}$ or $\overline{\text{SRESET}}$ can be asserted during entry into Nap mode are at risk. The risk is believed to be extremely small and has never been observed in a MPC74XX system.	Y	Y
29	Unpaired stwcx. may hang processor	The processor will hang if the Load/Store Unit does not report the completion of the stwcx. to the Completion Unit.	Y	Y

Errata No. 1: 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. 2: 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. The TAU is not implemented on the MPC7447A.

Errata No. 3: 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. 4: 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. 5: 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 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 TA for a data transfer does not occur sooner than the last cycle of the snoop response window (cycle after AACK). Note that DBG can still be driven as early as TS. Likewise, the system chipset logic must ensure that 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 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 TA or TEA for a data tenure before or while the AACK for that transaction is driven.

Work-Around:

Devices must not assert TA or TEA for a transaction before that transaction's snoop response window (cycle after 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. 6: 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. 7: 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. 8: 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. 9: $\overline{\text{MCP}}$ 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. 10: 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:

Fixed in Rev. 1.1.

Errata No. 11: 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. 12: 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:

Fixed in Rev. 1.1.

Errata No. 13: 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:

Fixed in Rev. 1.1.

Errata No. 14: 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. 15: 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. 16: 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. 17: A speculative tlbld instruction may cause an incorrect instruction translation

Description:

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

When a PTE match occurs during a hardware tablewalk, a copy of the PTE is written into the on-chip TLB and the R and C bits are updated in memory. If the load/store unit executes a **tlbld** instruction and a hardware tablewalk at the same time, then the hardware tablewalk may corrupt the lower sixteen bits of the PTE entry stored in memory. Since these sixteen bits include page protection bits, WIMG bits, R and C bits, and part of the RPN, if a subsequent page table search operation accesses this PTE entry, the resulting behavior is undefined.

An example code sequence that could cause the error is as follows:

```
.    # MSR[IR]=1; MSR[DR]=1; HID0[STEN]=0
store# store takes a hardware tablewalk
.
bcc  # branch taken
.
tlbld# in speculative non-taken path
```

In this example, (1) instruction and data translation is enabled and software tablewalks are disabled; (2) a load/store instruction takes a hardware tablewalk; (3) a conditional branch is taken, where the target of the branch requires an instruction hardware tablewalk; and (4) the **tlbld** instruction is in a window of less than 16 instructions (the depth of the completion buffer) past the branch.

If the **tlbld** instruction is speculatively executed at the same time as the data hardware tablewalk caused by the store, then the **tlbld** may cause the data hardware tablewalk to corrupt the PTE in memory.

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 **tlbld** instructions, then the software can avoid this error by guaranteeing that no **tlbld** instruction will be encountered on a speculative non-taken branch path.

Note that ‘encountering’ a **tlbld** instruction includes either data, code, or uninitialized memory that decodes to a **tlbld** instruction. It is, therefore, possible to experience errors in any system using hardware tablewalks even if neither software tablewalks nor **tlbld** instructions are ever used.

System software using **tlbld** instructions is more susceptible to this error occurring if a **tlbld** 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 **tlbld** instruction is likewise remote.

Work-Around:

There are several ways to prevent this erratum from occurring.

Any one of the following prevents a speculative **tlbld** from causing an incorrect instruction translation:

- Place a **sync** instruction before the **tlbld** instruction.
- Do not use **tlbld** instructions, even in non-taken branch paths.
- Use software tablewalks only.
- Do not enable instruction translation.

Projected Solution:

Under review.

Errata No. 18: Standard IEEE 1149.1 (JTAG) BYPASS / SAMPLE / PRELOAD instructions change the state of the processor

Description:

Shifting in a JTAG BYPASS/SAMPLE/PRELOAD instruction into TDI changes the state of the microprocessor.

In a system that has multiple IEEE 1149.1 (JTAG) compliant devices connected in a daisy-chain fashion, the emulator tool usually controls only one part at a time. The emulator tool then puts the rest of the IEEE 1149.1 devices in a BYPASS mode. The emulator tool could also use SAMPLE./PRELOAD instructions to access external caches.

When a BYPASS (0xFFFF) or a SAMPLE/PRELOAD (0x00F0) instruction is shifted into the TDI, MPC7447A does the following:

1. Puts a boundary-scan register between TDI and TDO.
2. Resets all latches values to their safe value to avoid signals contention.

Note: When all the latches are reset to their safe value as stated in action 2 above, it changes the state of the processor.

Projected Impact:

Whenever an emulator tool shifts in a standard JTAG BYPASS/SAMPLE/PRELOAD instruction while normal code is running on any system, an undefined behavior occurs.

Work-Around:

Emulator tool vendors should use 0x00FF instead of 0xFFFF for the standard JTAG BYPASS instruction. There is no known workaround for JTAG SAMPLE/PRELOAD instruction.

Projected Solution:

Under review.

Errata No. 19: 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. 20: Data parity errors not checked during L2 hardware flush

Description:

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. 21: Processor may hang when mtmsr/isync transitions MSR[IR] from 1->0 and isync instruction resides in unmapped page

Description:

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. 22: Scanning MSS_NRM chain via COP during softstop may cause unexpected interrupts upon resumption of normal operation

Description:

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. 23: tlbie snoop during Doze state may cause processor hang

Description:

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. 24: Data corruption due to write-through aliasing in the MMU

Description:

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. 25: HID0[SPD]=1 does not prevent instruction fetches past unresolved branches when MSR[IR]=0

Description:

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. 26: Unexpected instruction fetch may occur when transitioning MSR[IR] 0->1

Description:

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.

Errata No. 27: Performance Monitor Out (PMON_OUT) not operational

Overview:

The Performance Monitor Out output signal on the MPC7450 will not be asserted by the Performance Monitor unit as expected. PMON_OUT is defined to be asserted on the external interface when either a PMCn register overflow condition or a timebase event occurs. However, when either condition is detected, the processor incorrectly drops the event in most cases.

Projected Impact:

Systems designed to use the PMON_OUT signal as an indication of an internal Performance Monitor event will not have this feature, originally intended as a debug aid.

Work Arounds:

No known general workaround exists for this erratum.

The following two examples are the only known workarounds and are specific to 6:1 and 8:1 processor: bus ratios and the timebase unit.

In 6:1 processor:bus ratio, a time-base event with MMCR0[TBSEL]=00 (TBL[31]) can be used to force PMON_OUT to be asserted. PMON_OUT in this case would be asserted maximum once every eight system clocks.

In 8:1 processor:bus ratio, an overflow condition of event PMC1-4[3] with MMCR0[TBSEL]=00 (TBL[31]) can be used to force PMON_OUT to be asserted.

Projected Solution:

Under review.

Erratum No. 28: $\overline{\text{MCP}}$ or $\overline{\text{SRESET}}$ signal assertion during transition to Nap may hang processor

Overview:

If the machine check signal ($\overline{\text{MCP}}$) or soft reset signal ($\overline{\text{SRESET}}$) are asserted in a window after MSR[POW] has been set but before the processor has asserted the quiesce request signal ($\overline{\text{QREQ}}$), then the processor may encounter a hang condition.

The nap entry sequence in the MPC74XX User's Manual is as follows:

```
loop:   sync
        mtmsr POW
        isync
        b loop
```

Legacy software entering nap using this entry sequence are not affected:

```
        sync
        mtmsr POW
        isync
loop:   b loop
```

The fail happens when an outstanding out-of-order instruction fetch that occurs before the mtmsr—but not having received data and the results of which are no longer needed for execution—extends the window between the setting of POW and the assertion of $\overline{\text{QREQ}}$ by the processor. If the Branch Target Instruction cache (BTIC) is enabled, and the Nap entry code is in the BTIC due to a previous execution of this code, then the instructions from the sequence can be loaded into the completion buffer the cycle after execution has halted. If the $\overline{\text{MCP}}$ or $\overline{\text{SRESET}}$ signals are asserted at this point, they will void the entry into Nap as architected. However, both events require the completion buffer to be empty for their exceptions to be processed. Completion is halted, though, so no exceptions are processed and the hang occurs. $\overline{\text{SMI}}$ and $\overline{\text{INT}}$ signal assertions do not require the completion buffer to be empty and therefore do not cause a hang.

Projected Impact:

Systems where $\overline{\text{MCP}}$ or $\overline{\text{SRESET}}$ can be asserted during entry into Nap mode are at risk. The risk is believed to be extremely small and has never been observed in a MPC744X system.

Work Arounds:

Systems can implement any one of the following changes to work around this issue:

1. Use the legacy code entry sequence to enter Nap.
2. Disable the BTIC before entering Nap.
3. Invalidate the Nap entry code using an icbi instruction after taking the exception (a decrementer exception for example) that awakens the processor from Nap state.

Projected Solution:

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

Errata No. 29: Unpaired stwcx. may hang processor

Overview:

In general, lwarx and stwcx. instructions should be paired, with the same effective address used for both. The only exception is that an unpaired stwcx. instruction to any (scratch) effective address can be used to clear any reservation held by the processor.

When a stwcx. is unpaired, the e600 core may encounter an unexpected hang condition if each of the following is true:

1) Initial condition

```
RESERVE bit = 1 (due to previously executed lwarx)
Reservation = address A
Instruction executed = stwcx. to address B, resident in dL1 in Modified or
   Exclusive state
dL1 modified entry = address C
```

2) An external snoop to address A of a type shown in Table 4 occurs while the stwcx. is being processed by the Load/Store Unit.

Table 4. Snooped store types that cancel a reservation

Command	TT
Write-with-flush	0x02
Write-with-kill	0x06
Kill block	0x0c
Read-with-intent-to-modify	0x0e
Read claim	0x0f
Read-with-intent-to-modify-atomic (stwcx.)	0x1e

3) An external snoop to address C follows the snoop to address A while the stwcx. is being processed by the Load/Store Unit.

Normally, the snoop to address A would clear the RESERVE bit, and the snoop to address C would initiate a snoop push of the modified line from the dL1. The stwcx. may succeed or fail depending on when the snoop to A cleared the RESERVE bit, but the completion of the stwcx. should be reported regardless.

A one cycle window exists wherein the above sequence will cause the Load/Store Unit to not report the completion of the stwcx. to the Completion Unit. As a result, the processor will hang. The hang can only be cleared by asserting hard reset.

Projected Impact:

The processor will hang if the Load/Store Unit does not report the completion of the stwcx. to the Completion Unit. Paired lwarx/stwcx. instructions are not affected by this erratum. Most operating systems include an unpaired stwcx. at the end of exception handler code and context switch code to clear the RESERVE bit before returning to normal execution. Note that stwcx. is a user level instruction.

Non-SMP environments are less susceptible to this erratum due to the requirement of an external snoop to address A which holds the reservation from a previous lwarx instruction. Most operating systems do not allow a snoop to be initiated by an external peripheral to an address that the core wants to reserve in a non-SMP environment. If the non-SMP environment's operating system does not allow such snoops, then the environment will not be affected by this erratum.

This erratum effects the core:bus ratios shown in Table 5.

Table 5. Core:Bus Ratios Affected

Part	2:1	5:2	3:1	7:2	4:1	9:2	5:1	11:2	6:1	13:2	7:1	15:2	8:1	17:2	9:1
MPC7447A	yes	yes[1]	yes	yes[1]	yes	yes[1]	no	no	no	no	no	no	no	no	no

1 This ratio is reachable only during Dynamic Frequency Switching (DFS).

Work Arounds:

Any individual one of the following steps can be taken to work around this issue:

Option 1: Place a lwarx instruction to the same scratch address as the stwcx. immediately before the stwcx., or

Option 2: Place a dcbf instruction to the same scratch address as the stwcx. immediately before the stwcx., or

Option 3: Do not permit an external snoop to the address of the reservation address.

Interrupts must be disabled (MSR[EE] = 0) during the instruction sequences for the first two options. In most operating systems, interrupts are already disabled when an unpaired stwcx. is executed.

If the workarounds are not possible, then the number of core:bus ratios affected in some products may be reduced (as shown in Table 6) by placing the L2 cache in L2 data only mode (L2CR[L2DO] = 1).

Table 6. Core:Bus Ratios Affected if L2CR[L2DO] = 1

Part	2:1	5:2	3:1	7:2	4:1	9:2	5:1	11:2	6:1	13:2	7:1	15:2	8:1	17:2	9:1
MPC7447A	yes	yes[1]	no	no[1]	no	no[1]	no	no	no	no	no	no	no	no	no

1 This ratio is reachable only during Dynamic Frequency Switching (DFS).

Projected Solution:

Under review.

How to Reach Us:

Home Page:
www.freescale.com

email:
support@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
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

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
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: MPC7447ACE
Rev. 9
10/2007

freescale™
semiconductor