Advance Information

MPC7400CE/D
Rev. 13, 6/2003

MPC7400 RISC Microprocessor Chip Errata

The MPC7400 is a PowerPC™ microprocessor. This document details all known silicon errata for the MPC7400. Table 1 provides a revision history for this chip errata document.

Table 1. Document Revision History

Rev. No.	Significant Changes
0–12	Earlier releases of document.
13	Updated template. Added Errors 24 and 25.

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

MPC7400 Revision	Part Marking	Processor Version Register
2.0	B	000C0200
2.1	C	000C0201
2.2	D	000C0202
2.6	H	000C0206
2.7	I	000C0207
2.8	J	000C0208
2.9	K	000C0209

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 ‘—’ entry means it does not apply.

Table 3. Summary of Silicon Errata and Applicable Revision

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
1	ICBI may cause icache corruption in 2:1 and 2.5:1 system bus modes	If an ICBI address broadcast collides with an instruction cache reload from the bus, the icache can be corrupted.	Operating system code and self modifying code.	Run the processor in 3:1 or higher mode.	Y	—	—	—	—	—	—
2	Incorrect value was written to the MSR after running POR ABIST	When running ABIST after POR, the renames remained valid causing MSR to be updated with the incorrect value.	Running ABIST after POR	Insert an ISYNC instruction at the interrupt vector 0xFFF0_0100	Y	Y	Y	Y	Y	Y	Y
3	Not all GPRs and FPRs are initialized after ABIST	Not all GPRs and FPRs are initialized after ABIST due to invalid instructions in the instruction buffers.	GPRs and FPRs may not be initialized during ABIST if the contents of the instruction buffers can be decoded to non-zero GPR or FPR destination addresses.	None	Y	Y	Y	Y	Y	Y	Y
4	L2 orphan may cause processor hang and/or data corruption	If a Data Reload Table entry is deallocated (due to speculative cancellation or store-miss-merging) the L2 may still return data to the result bus, causing a processor hang and/or data corruption.	Systems using L2 cache.	Set Data Reload Table size to 4 entries or disable the L2 cache.	Y	—	—	—	—	—	—
5	SPR register corruption after $\overline{\text{HRESET}}$	A race condition within the clock regenerator circuit may create SPR register corruption when switching from scan clock to gclk.	Systems which use SPR registers without initialization.	Initialize all SPR registers after reset and before use.	Y	—	—	—	—	—	—

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

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
6	Incorrect speculative and folding logic may cause processor hang or incorrect branch resolution	When folding a non-speculative branch and dispatching a speculative instruction in the sane cycle, possibility of forwarding incorrect non-speculative signal to future file, causing a processor hang or incorrect branch resolution.	Any code with this particular instruction sequence.	Disable branch folding by setting IABR with any address.	Y	—	—	—	—	—	—
7	Store data lost when store gathering is enabled during cache ops	Overlapping stores may be performed out of order when store gathering is enabled and a cache op is performed.	Stores may cause data corruption when store gathering is enabled.	Disable store gathering.	Y	—	—	—	—	—	—
8	Store data lost with mismatched LWARX/STWCX address pairs	If a STWCX that hits to a shared cache line is pending when a snoop transaction to a different address matches the reservation address, the STWCX may be canceled and expose stale data in the L2.	Any code which uses mismatched LWARX/STWCX address pairs.	1. Avoid mismatched LWARX/STWCX address pairs, or 2. Operate in MEI mode to prevent STWCX hit to shared, or 3. Turn off the L2.	Y	—	—	—	—	—	—
9	L2 global invalidate may not complete	If a mtspr that sets the L2I bit of the L2CR is followed closely by a mfspr from the L2CR, the mfspr may not return the correct value of the L2IP bit.	Systems which use the L2 cache.	1. Insert an isync instruction after any mtspr that sets the L2I bit of the L2CR, or 2. Use alternate code sequence to invalidate L2 (see full description), or 3. Tie CHK_ pin to $\overline{\text{HRESET}}$ for POR ABIST.	Y	—	—	—	—	—	—
10	Guarded speculative load may access memory	A guarded load on a mispredicted path that is supposed to be cancelled the cycle it is written into the reload table may be allowed to access memory.	May be present in any system.	Set the DL1 bypass disable bit in MSSCR1.	Y	—	—	—	—	—	—

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

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
11	Asserting $\overline{\text{TEA}}$ and $\overline{\text{ARTRY}}$ together may cause loss of data	Asserting $\overline{\text{TEA}}$ and $\overline{\text{ARTRY}}$ together in the first cycle of the snoop response window may cause loss of iside data.	Any system that permits the aggressive timing of $\overline{\text{TEA}}$ in the first cycle of the snoop response window.	Delay assertion of $\overline{\text{TEA}}$ until the second cycle of the snoop response window or later.	Y	Y	Y	Y	Y	Y	Y
12	Guarded speculative load with folding may access memory	A guarded load on a mispredicted path which is folded into by a subsequent load may be allowed to access memory.	May be present in any system.	Set the LFQ_SIZE bits in MSSCR1 to ‘10.’	Y	Y	—	—	—	—	—
13	Speculative instruction stream may cause duplicate data cache tags	A canceled speculative load on a mispredicted path causes a reload of the data cache in the same cycle that a store to the same cache line accesses the cache, creating duplicate data cache tags	May be present in any system.	1. Set the L1OPQ_SIZE bits in MSSCR1 to ‘10,’ or 2. Set the L1WDB_SIZE bits in MSSCR1 to ‘10,’ or 3. Set the DRLT_SIZE bits in MSSCR1 to ‘10.’	Y	Y	Y	Y	Y	—	—
14	Speculative or non-coherent transactions may cause loss of data	Speculative transactions or coherency transactions mapped to non-coherent cacheable space may be dropped in MAXBus mode, leading to an undesirable processor state	May be present in MAXBus systems.	In MAXBus mode: 1. Set the NGBE bit of MSSCR0 to ‘1’ and set the SPD bit of HID0 to ‘1,’ or 2. Set the L2MQ_SIZE bit in MSSCR1 to ‘1.’	Y	Y	Y	Y	Y	—	—
15	Data stream touch may cause duplicate data cache tags	A Data Stream Touch instruction that accesses the data cache in the same cycle that a canceled speculative load causes a reload of the same line to the data cache may create duplicate data cache tags.	May be present in any system that uses the DST(T)/DSTST(T) instructions.	1. Set the NOPDST bit in HID0, or 2. Set the DRLT_SIZE bits in MSSCR1 to ‘01,’ or 3. Flush the cache on identification of the problem and continue.	Y	Y	Y	Y	Y	—	—

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

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
16	Incorrect condition code on mismatched LWARX/STWCX pair	A STWCX instruction may be performed without setting the condition code if the store hits in the L2 and the LWARX instruction that set the reservation is to another coherency granule.	Any code which uses mismatched LWARX/STWCX address pairs	1. Avoid mismatched LWARX/STWCX address pairs, or 2. Turn off the L2.	Y	Y	Y	Y	Y	Y	Y
17	TLBSYNC may hang in the presence of a DST	The MPC7400 may not make forward progress if a DST has caused an MMU tablewalk, that MMU tablewalk was marked by a TLBIE instruction, and a TLBSYNC instruction is pipelined the cycle after the MMU tablewalk accesses the dL1 cache.	Any system which has an active DST engine while executing a TLBSYNC instruction in a privileged context.	Insert a DSSALL instruction before a TLBSYNC instruction.	Y	Y	Y	Y	Y	Y	Y
18	Queueing six transactions to secondary bus may hang the system	Queueing six transactions from a single MAX processor could use all Data Transaction Queue resources and hang the system if forward progress cannot be made by allowing MAX to complete at least one outstanding transaction.	Any system which allows six outstanding transactions from a single processor and which has a secondary bus with characteristics as detailed in full description.	1. Limit the number of outstanding transactions from a secondary bus to five in system logic, or 2. Mark the memory space on the secondary bus as guarded and avoid DST(ST)(T) and LMW instructions.	Y	Y	Y	Y	Y	Y	Y
19	Consecutive interrupts may be nonrecoverable	Consecutive performance monitor, decrementer, or thermal management interrupts may become non-recoverable.	Any system which enables a combination of performance monitor, decrementer, or thermal management interrupts at the same time.	Avoid enabling any combination of performance monitor, decrementer, or thermal management interrupts at the same time.	Y	Y	Y	Y	Y	Y	Y

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

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
20	Disabling L2 cache may hang processor	The MPC7400 may hang if the L2 cache is disabled during an outstanding instruction fetch.	Any system which executes code to disable the L2 cache.	1. Mark the code that disables the L2 cache as Cache Inhibited, or 2. Preload the code that disables the L2 cache into the instruction cache before execution.	Y	Y	Y	Y	Y	Y	Y
21	Timebase or decrementer may lose accuracy in nap mode	The timebase counter or decrementer may lose accuracy when transitioning from the nap power saving mode.	Any system which uses the nap power saving mode and requires absolute accuracy from the timebase counter or decrementer.	1. Avoid the use of nap mode, or 2. Avoid using the timebase counter or decrementer for highly accurate measurements.	Y	Y	Y	Y	Y	Y	Y
22	Read and write memory problem using COP EXMEM	Several clock regenerators that need to be involved in Exmem operation receive the wrong c1_test signal. This means that information in these latches is lost during an Exmem shift.	Any system which uses 'cop exmem' to get access to memory.	LSRL must be used for external memory access.	Y	Y	Y	Y	Y	Y	Y
23	IFTT mode does not identify **dcbt**/**dst** instructions as data fetches	The Instruction Fetch Transaction Type (IFTT) encoding differentiation mode does not correctly identify **dcbt**/**dst** instructions as data fetches.	Any system which depends on IFTT to differentiate instruction from data fetches.	1. Replace **dcbt**/**dst** with **dcbtst**/**dstst**. 2. Set HID0[NOPTI] and HID0[NOPDST] to no-op all touch instructions. 3. Remove **dcbt**/**dst** from the code.	Y	Y	Y	Y	Y	Y	Y

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

No.	Problem	Description	Impact	Work Arounds	Present in Version:						
					2.0	2.1	2.2	2.6	2.7	2.8	2.9
24	L2 global invalidate may not clear way 0 after L2 usage	The L2 global invalidate function may not clear way 0 of the L2 cache after L2 cache usage. The L2 global invalidate function operates correctly if it is performed before the L2 cache has been used for the first time.	Any system which uses the L2 global invalidate function after the L2 cache has been used.	1) Limit use of the L2 global invalidate function to before the L2 cache has been used for the first time. 2) Use L2 cache hardware flush.	Y	Y	Y	Y	Y	Y	Y
25	Live-lock when In PLL bypass mode	In PLL bypass mode (i.e., bus mode of 1:1 ratio), there is a possibility for a self-snoopable transaction to be self-$\overline{\text{ARTRY}}$ed forever.	Any system in PLL bypass mode that uses self-snoopable transactions that may be self-$\overline{\text{ARTRY}}$ed.	Avoid self-snoopable transactions in PLL bypass mode, or implement synchronization such that the transactions will not be retried.	Y	Y	Y	Y	Y	Y	Y

Error No. 1: ICBI may cause icache corruption in 2:1 and 2.5:1 bus modes

Overview:

An ICBI address broadcast which collides with an instruction cache reload from the system bus may cause icache corruption if the processor is running in core-to-bus clock ratios of 2:1 or 2.5:1.

Detailed Description:

If a self generated or externally generated ICBI request is snooped from the address bus in the same processor cycle in which the critical word of data for an instruction fetch is returned from the system bus, the data from the system bus will be written into the iL1 cache but the tag will not be updated with the new tag value. This results in icache corruption in which the instructions stored in the icache with the old tag value do not match the instructions stored in main memory for that corresponding address.

Note that for this scenario to happen, bits 20–26 of the ICBI address must not match bits 20–26 of the instruction fetch and the instruction address which is corrupted must be marked least recently used. This corruption will only cause a detectable software failure if the corrupted instruction address, already marked least recently used, is referenced by the instruction stream before it is deallocated from the iL1 cache by another miss to that same cache set.

Projected Impact:

Operating system code and self modifying code.

Work Arounds:

Run the processor at 3:1 or higher clock ratio to the bus.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 2: Incorrect value was written to the MSR after running POR ABIST

Overview:

When running Array Built In Self Test (ABIST) after Power On Reset (POR), the renames remain valid causing MSR to be updated with the incorrect value.

Detailed Descriptions:

After POR, MAX generates a global cancel and immediately branches to the system interrupt vector 0xFFF0_0100. Because ABIST will be running after POR, the bus holds off fetching vector 0xFFF0_0100 until ABIST is completed. During ABIST, all rename valid bits are set to initiate initialization of the register files. Once ABIST is completed, the fetching of 0xFFF0_0100 is started, but the renames remain valid. Because the global cancel was executed prior to ABIST, there is no mechanism to clear the rename valid bits. Any instructions executed in the interrupt routine that need a source operand will use the valid renames instead of the correct operand. If this source operand is an MTMSR, then the incorrect value is written to the MSR.

Projected Impact:

Systems that run ABIST after POR.

Work Arounds:

This problem can be fixed in software by inserting an ISYNC instruction at the interrupt vector, 0xFFF0_0100.

Projected Solution:

This may be fixed in a future MAX revision.

Error No. 3: Not all GPRs and FPRs are initialized after ABIST

Overview:

Not all GPRs and FPRs are initialized after ABIST due to invalid instructions in the instruction buffers.

Detailed Descriptions:

During ABIST, the GPRs and FPRs are supposed to be initialized sequentially to a known value. However, after HRESET, the instruction buffer contains invalid instructions with random data. These random bits could be decoded to actual GPR or FPR destination addresses, resulting in corrupted GPR or FPR registers.

Projected Impact:

GPRs and FPRs may not be initialized during ABIST if the contents of the instruction buffers can be decoded to non-zero GPR or FPR destination addresses.

Work Arounds:

No work around identified.

Projected Solution:

This may be fixed in a future MAX revision.

Error No. 4: L2 Orphan can cause processor hang and/or data corruption

Overview:

If a Data Reload Table entry clears its ‘needs_l2_data’ bit while that entry is accessing the L2 SRAMs (due to speculative load cancellation or store-miss-merging), the L2 may still signal the result bus that data for that entry is being returned. This may cause the processor to hang and/or create data corruption.

Detailed Description:

When the ‘needs_l2_data’ bit of a Data Reload Table entry goes away while the L2 is fetching data from the L2 SRAMs for this entry, the L2 can erroneously signal the result bus complex that the critical double/quad word is being returned, even though the transaction no longer needs the data. This situation can happen when speculative loads are cancelled or when store-miss-merging has written an entire cache line.

The erroneous signal to the result bus will not cause any harm unless the entry whose ‘needs_l2_data’ bit was cleared is reallocated with a new entry that represents a load operation. If this is the case, the result bus complex will incorrectly finish the load and write the wrong data to the rename registers. If the newly allocated load is part of a misaligned load (e.g., a load float double that crosses a double word boundary), the state of the rename register may be corrupted in a way such that the processor hangs.

Projected Impact:

This affects any system using the L2 cache.

Work Arounds:

Set bits 3:4 of MSSCR1 to b’11’ to force the Data Reload Table size to 4 entries or disable the L2 cache.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 5: SPR register corruption after $\overline{\text{HRESET}}$

Overview:

A race condition within the clock regenerator circuit may create SPR register corruption when switching from scan clock to gclk.

Detailed Description:

The basic problem is a race between jt2_scan_c2_ and gclk in the clock regenerator circuit. The scan clock jt2_scan_c2_ does not receive the same routing priority as gclk. The delay in propagating jt2_scan_c2_ across the chip causes the jt2_scan_c2_ signal to hold beyond the next gclk pulse. This creates a spurious c2 clock pulse from the clock regenerator. The jt2_scan_c2_ signal is the hold enable of the clock regenerator which causes the master latch to flush data from its input to the input of the slave latch. The spurious c2 clock pulse is fed to the slave latch which allows the data from the input of the master latch to propagate to the output of the slave latch.

The rename bus is precharged to 1s and the rename valid bit is set valid during reset. An SPR register that is written from the rename bus can become corrupted when the version of jt2_scan_c2_ which is seen at its clock regenerator exhibits the behavior described above.

Projected Impact:

This affects any system which uses SPR registers without initialization.

Work Arounds:

Initialize all SPR registers after reset and before use.

Table 4. Suggested Initial Values for SPR Registers

Register	Initial Value	Notes
DEC	0xFFFFFFFF	
TBU TBL	0x00000000 0x00000000	
MSR	0x00000040	MSR[IP] set
UPMC0 UPMC1 UPMC2 UPMC3	0x00000000 0x00000000 0x00000000 0x00000000	
HID0	0x00000000	
IABR	0x00000000	
DABR	0x00000000	
MSSCR0 MSSCR1	0x00000000 0x00000000	
L2CR	0x00000000	
ICTC	0x00000000	

Table 4. Suggested Initial Values for SPR Registers (continued)

Register	Initial Value	Notes
THRM1 THRM2 THRM3	0x00000000 0x00000000 0x00000000	
PIR	0x00000000	
VRSAVE	0x00000000	
MMCR0 MMCR1 MMCR2	0x00000000 0x00000000 0x00000000	
UMMCR0 UMMCR1 UMMCR2	0x00000000 0x00000000 0x00000000	
SIA	0x00000000	
USIA	0x00000000	
SPRG0 SPRG1 SPRG2 SPRG3	0x00000000 0x00000000 0x00000000 0x00000000	
DAR	0x00000000	
DSISR	0x00000000	
EAR	0x00000000	
SDR1	0x00000000	
SRR0 SRR1	0x00000000 0x00000040	SRR1[MSR_IP] from reset interrupt state
LR	0x00000000	
XER	0x00000000	
BAMR	0x00000000	
FPSCR	0x0000000000000000	

Projected Solution:

Fixed in Rev. 2.1.

Error No. 6: Incorrect speculative and folding logic may cause a processor hang

Overview:

Folding a branch and dispatching a speculative instruction in the same cycle may cause incorrect information to be sent to the future file, which may cause branches to hang or resolve incorrectly.

Detailed Description:

The problem occurs with a minimum sequence of four instructions:

1. An instruction that may set a CR bit
   (... a few instructions may be inserted here ...)
2. A branch which does not update count or link
3. A conditional branch (one that doesn't update count or link) that is conditional upon the same CR nibble that instruction 1 might set
4. Another instruction that may set the same CR nibble (a **stwcx.** in this instruction will cause a hang)

This will possibly cause the part to hang, or to take the wrong leg of a branch.

Instruction 1 gets dispatched.

Instruction 2 can resolve, and is marked to be folded in instruction-buffer.

Instruction 3 is a branch, which cannot resolve until instruction 1 is finished. So it goes into an unresolved branch slot and is folded out of the instruction buffer.

At this point, instruction 1 is in the completion buffer, waiting to finish.

Instruction 2 is in I-buff 0, marked non-speculative and to be folded.

Instruction 3 is in ubr0 slot waiting for instruction 1, not in the I-buffer.

Instruction 4 is in I-buff 1, marked as speculative.

Instructions 2 and 4 get dispatched in the same cycle. Normally, there is some logic around each I-buffer signal relating to dispatch, so that when an instruction is folded out of I-buff 0, I-buff 1 becomes the first dispatched instruction seen by completion.

The error was that this special fold case was not done correctly for the speculative signal going into the future file. Thus, instruction 4 was seen as a non-speculative instruction that updated the CR. Thus, the future file marked that CR nibble busy until that instruction completed. But it won't complete until the branch (instruction 3) resolves. But the branch won't resolve because the future file is incorrectly signaling that the CR nibble is busy. Thus, the processor will hang.

NOTE: If instruction 4 is not a STWCX., it will be able to finish speculatively. Its CR bits will be forwarded to the future file and on to the branch unit. Thus, if this instruction is not a STWCX., the failure might be seen as an incorrect branch resolution.

Projected Impact:

This has the potential to impact any code where branches are being folded, with back to back conditional branches, and with high latency instructions that update the CR register.

Work Arounds:

Disable branch folding by putting part in single step mode, branch trace mode, or by turning on IABR.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 7: Store data lost when store gathering is enabled during cache ops

Overview:

Overlapping stores may be performed out of order when store gathering is enabled and a cache op (**dcba**, **dcbi**, **dcbz**, **dcbf**, **dcbst**) is performed.

Detailed Description:

The following example code sequence will be subject to this erratum:

```
lwzx    EA:     00403080 (miss in data cache)
stwx    EA:     00403080
dcbz    EA:     00403080
stwx    EA:     00403084
```

When **lwzx** is executed, it should miss in the L1 cache and create a reload request. The following instruction **stwx** EA: 00403080 will hit to the reload request created by the previous **lwzx** and will remain in the committed store queue (CSQ) until the reload is complete. While the reload is in progress, the **dcbz** instruction will also hit on the reload request created by the **lwzx** and remain in CSQ. It should be noted here that **dcbz** instruction will modify 8 words starting at address location 00403080.

Due to this erratum, the next instruction, **stwx** EA: 00403084, erroneously gathers with **stwx** EA: 00403080. When the reload for the **lwzx** is complete, the gathered **stwx** will update address locations 00403084 and 00403088. Then **dcbz** will then update address locations 00403080—0040309A. It can be seen here that the address locations 00403084 and 00403088 are modified out-of-order resulting in data loss.

Projected Impact:

Possible data corruption in the system where store gathering is enabled and a cache op (**dcba**, **dcbi**, **dcbz**, **dcbf**, **dcbst**) is used.

Work Arounds:

Disable store gathering by clearing the store gathering enable (SGE) bit in the HID0 register. SGE is bit 24 in the HID0 register. Note that this has the following negative performance impact: two adjacent **stw** instructions mapped to write-through or cache-inhibited space will no longer gather into a single 8-byte bus operation. Instead, two separate 4-byte bus operations will be performed.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 8: Store data lost with mismatched LWARX/STWCX address pairs

Overview:

If a STWCX that hits to a shared cache line is pending when a snoop transaction to a different address matches the reservation address, the STWCX may be canceled and expose stale data in the L2.

Detailed Description:

The following sequence of events is needed to create this erratum:

1. A load instruction establishes a cache line in the exclusive state in the L1.
2. Because of repeated accesses to the same cache set as the cache line in step 1, the cache line is castout to the L2, still in the exclusive state.
3. Another load instruction forces the reload of the cache line from the L2 to the L1.
4. A store instruction stores to the cache line and changes the state in the L1 to modified, while the old data in the exclusive state remains in the L2.
5. A snoop READ transaction to the cache line changes the state in both caches to shared and pushes the correct data from the L1 to memory.
6. A STWCX instruction that hits to the shared cache line in the L1 but is different from the current reservation address invalidates the cache line in the L1, creates an entry in the reload table for the cache line, and ultimately causes a KILL transaction to the cache line to be queued in the L2MQ.
7. A snoop RESERVATION KILL (rwitm, rwitm atomic, rclaim, wr w/kill, wr w/flush, wr w/flush atomic, kill) transaction that matches the reservation address kills the reservation, causing the reload table entry for the pending STWCX hit to shared to be invalidated, causing the cache line to be lost and the L2MQ entry for the KILL transaction to be dequeued.
8. Any subsequent access to the cache line in this processor will incorrectly see stale data in the L2.

Projected Impact:

Any code which uses mismatched LWARX/STWCX address pairs.

Work Arounds:

Any of the following will serve as a work around to this erratum:

1. Avoid STWCX instructions that do not match the current reservation address or force a DCBF to the STWCX address before a STWCX that does not match the current reservation address.
2. Operate in MEI mode to avoid STWCX hit to shared.
3. Turn off the L2.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 9: L2 global invalidate may not complete

Overview:

If an **mtspr** that sets the L2I bit of the L2CR register is followed closely by an **mfspr** from the L2CR register, the mfspr may not return the correct value of the L2IP bit.

Detailed Description:

After an **mtspr** that sets the L2I bit of the L2CR register is executed, it takes two cycles before the L2IP bit of the L2CR register is updated. If during these two cycles an **mfspr** reads the L2CR, it will not return the correct value of the L2IP bit.

Consider the following code:

```
        addis   r1,r0,0x0020        # set L2I bit of L2CR
        mtspr   L2CR, r1

loop:   mfspr   r2,L2CR             # read L2CR
        andi.   r2,r2,0x0001        # mask off L2IP bit
        bne     loop                # repeat if L2IP == 1
```

This loop may exit after the first iteration. Because the loop is done, the system will think that the L2 has been fully invalidated, when in fact the invalidation is actually still in progress. If the L2 is then enabled (by setting the L2E bit in the L2CR), the global invalidation state machine will terminate without running to completion, thus leaving uninitialized data in the L2 tag.

Projected Impact:

Systems which use the L2 cache.

Work Arounds:

Any of the following will serve as a work around to this erratum:

1. Insert an **isync** instruction after any **mtspr** that sets the L2I bit of the L2CR.
2. Use the following code to invalidate the L2 tag:

```
addis   r1, r0, 0x0020
mtspr   L2CR, r1            # set L2I bit of L2CR
sync                        # wait for all memory transaction to
                              finish
addis   r1, r0, 0x0000
mtspr   L2CR, r1            # clear L2I bit of L2CR
```

   Note: This code is very similar to that recommended for flushing the L1 data cache and L2 cache using their respective hardware flush mechanisms.

3. Tie the $\overline{\text{CHK}}$ pin to $\overline{\text{HRESET}}$ to force ABIST to run on power-up. This will clear all arrays including the L2 tag, thus eliminating the need for L2 global invalidation before enabling the L2 cache.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 10: Guarded speculative load may access memory

Overview:

A guarded load on a mispredicted path that is supposed to be canceled the cycle it is written into the reload table may be allowed to access memory.

Detailed Description:

A conditional branch is mispredicted and folded out of the instruction stream. Because of dependencies, the conditional branch is not resolved until the completion buffer is completely filled with the speculative path. The key is that since the completion buffer is full with the speculative path, the first IDN, or completion buffer entry number, issued to the correct path will be the same as the first IDN of the speculative path. The first instruction on the speculative path is a guarded load which is allowed to access the DL1. The conditional branch is resolved not taken and the load is marked cancelled the same cycle it is written into the DL1 reload table.

In the next cycle, the cancel request for the load in the DL1 reload table is processed. This sets up several status bits, including ‘needs_reload’ and ‘needs_forward,’ to change in the next cycle. Also, the DL1 reload table makes a request to the L2 for the load. Finally, a new load instruction is dispatched on the correct path which is given the same IDN as the original guarded speculative load.

In the next cycle, the ‘needs_reload’ and ‘needs_forward’ status bits are processed. This sets up the invalidation of the DL1 reload table entry for the load in the next cycle. Also, the L2 makes a request to the L2MQ for the load. Also the IDN of the new load on the correct path is marked as the oldest in the machine. Because this new load has the same IDN number as the original guarded speculative load, the speculative bit in the DL1 reload table entry for the original guarded speculative load will be reset in the next cycle.

In the next cycle, both the DL1 reload table entry valid bit and speculative bit are reset at the same time, causing the L2MQ to miss the fact that the instruction was canceled, providing the means for the guarded speculative load to access memory.

Projected Impact:

May be present in any system.

Work Arounds:

Set the DL1 bypass disable bit (bit 29) in MSSCR1.

Projected Solution:

Fixed in Rev. 2.1.

Error No. 11: Asserting $\overline{\text{TEA}}$ and $\overline{\text{ARTRY}}$ together may cause loss of data

Overview:

Asserting $\overline{\text{TEA}}$ and $\overline{\text{ARTRY}}$ together in the first cycle of the snoop response window may cause loss of I-side data.

Detailed Description:

The following bus activity will result in the errata:

$\overline{\text{TS}}$	$\overline{\text{AACK}}$ $\overline{\text{DBG}}$	$\overline{\text{ARTRY}}$ $\overline{\text{TEA}}$

The Bus Interface Unit will cancel the data tenure and mark the address tenure for resubmission to the bus. This is the normal course of action if the $\overline{\text{ARTRY}}$ has precedence over the $\overline{\text{TEA}}$. The Instruction Reload Table will cancel the address tenure and start another address tenure for the same address. This is the normal course of action if the $\overline{\text{TEA}}$ has precedence over the $\overline{\text{ARTRY}}$. The Bus Interface Unit will continue to resubmit the original address tenure until it completes without $\overline{\text{ARTRY}}$. At this point it will signal completion to the Instruction Reload Table which is no longer expecting that transaction to be active.

Projected Impact:

Any system that permits the aggressive timing of $\overline{\text{TEA}}$ in the first cycle of the snoop response window.

Work Arounds:

Delay assertion of $\overline{\text{TEA}}$ until the second cycle of the snoop response window or later.

Projected Solution:

This may be fixed in a future revision.

Error No. 12: Guarded speculative load with folding may access memory

Overview:

A guarded load on a mispredicted path which is folded into by a subsequent load may be allowed to access memory.

Detailed Description:

A conditional branch is mispredicted and folded out of the instruction stream. Because of dependencies, the conditional branch is not resolved until the completion buffer is completely filled with the speculative path. The key is that since the completion buffer is full with the speculative path, the first IDN, or completion buffer entry number, issued to the correct path will be the same as the first IDN of the speculative path. The first instruction on the speculative path is a guarded load which is allowed to access the DL1 and is written into the DL1 reload table.

In the next cycle, the conditional branch is resolved not taken and a cancel request for the speculative path is sent. The DL1 reload table makes a request to the L2 for the load despite the cancel. In addition, a subsequent load on the speculative path to the same cache line as the first speculative load accesses the DL1 and is sent to the load fold queue despite the cancel.

In the next cycle, the cancel request is processed by both the reload table and the load fold queue. In the reload table, several status bits, including ‘needs_reload’ and ‘needs_forward,’ will be reset in the next cycle. In the load fold queue, the entry will be invalidated in the next cycle. However, the cancel does not prevent the subsequent speculative load in the load fold queue from setting an active signal that indicates it requires the reload table entry of the first speculative load. In addition, the L2 makes a request to the bus miss queue for the first speculative load. Finally, a new load instruction is dispatched on the correct path which is given the same IDN as the original guarded speculative load.

In the next cycle, the original guarded speculative load is valid in the bus miss queue, despite having been cancelled. The active from the load fold queue prevents the reload table valid bit from being reset for one cycle, even though the ‘needs_forward’ and ‘needs_reload’ bits are already reset. Finally, the IDN of the new load on the correct path is marked as the oldest in the machine. Because this new load has the same IDN number as the original guarded speculative load, the speculative bit in the DL1 reload table entry for the original guarded speculative load will be reset in the next cycle.

In the next cycle, both the DL1 reload table entry valid bit and speculative bit are reset at the same time, causing the L2MQ to miss the fact that the instruction was canceled, providing the means for the guarded speculative load to access memory.

It is a violation of the architecture specification for a guarded load to access memory speculatively. This becomes a problem in applications using memory mapped I/O space where reading an address may set or reset a control flag.

Projected Impact:

May be present in any system.

Work Arounds:

Set the LFQ_SIZE bits (bits 5–6) in MSSCR1 to ‘10.’ Note that this will slow down load accesses to the DL1 to a maximum of one every other cycle.

Projected Solution:

Fixed in Rev. 2.2.

Error No. 13: Speculative instruction stream may cause duplicate data cache tags

Overview:

A canceled speculative load on a mispredicted path causes a reload of the data cache in the same cycle that a store to the same cache line accesses the cache, creating duplicate data cache tags.

Detailed Description:

A store to cache line A is dispatched, becomes the oldest instruction in the machine, reads data, and ends up in the completed store queue. This store is followed by a conditional branch that is mispredicted taken. The instruction at the target of the mispredicted branch is a load to non-overlapping bytes of cache line A. The load correctly bypasses the store, misses in the cache, and allocates a data reload table entry. The reload request from the load hits in the L2.

In cycle 0, the L2 completely returns the last beat of data for the speculative load request and this data is written into the data reload data buffer. Also in this cycle, the branch is resolved.

In cycle 1, the speculative load is marked as ready to reload the dL1, a speculative cancel is issued due to the mispredicted branch, and the store starts an access at the dL1.

In cycle 2, the reload for the speculative load starts an access at the dL1. The speculative cancel in the previous cycle causes the needs_reload bit for the speculative load data reload table entry to be reset. The store has progressed to the second stage of the dL1 pipeline and checks all reload table entries for an address collision. Because the needs_reload bit for the speculative reload has been reset, the address collision test fails and the store allocates a new data reload table entry.

In cycle 3, the second half of the speculative reload finishes. Cache line A is now valid in the cache and in the data reload table.

A subsequent access to the same cache line will identify the data reload table and/or data cache duplicate tag.

Projected Impact:

May be present in any system.

Work Arounds:

Any of the following will serve as a work around to this erratum:

1. Set the L1OPQ_SIZE bits (bits 7–8) in MSSCR1 to ‘10.’ Note that this will slow down store accesses to the DL1 to a maximum of one every other cycle.
2. Set the L1WDB_SIZE bits (bits 9–10) in MSSCR1 to ‘10.’ Note that this will slow down store accesses to the DL1 to a maximum of one every other cycle. Because L1 Write Data Buffer entries remain active for a longer period of time than L1 OP Queue entries, this work around is expected to result in lower performance.
3. Set the DRLT_SIZE bits (bits 3–4) in MSSCR1 to ‘10.’ Note that this will slow down load and store accesses to the DL1 to a maximum of one every other cycle. This work around is expected to result in the lowest performance of the three options.

Projected Solution:

Fixed in Rev. 2.8.

Error No. 14: Speculative or non-coherent transactions may cause loss of data

Overview:

Speculative transactions or coherency transactions mapped to non-coherent cacheable space may be dropped in MAXBus mode, leading to an undesirable processor state. The problem does not occur in 60x mode.

Detailed Description:

Transactions that are canceled after being written into the L2 miss queue (L2MQ) of the bus interface unit (BIU) may leave the L2MQ in an undesirable state. Transaction cancellations can occur in one of two ways. In the speculative cancellation case, the transactions are loads from two separate mispredicted branches. In the coherency cancellation case, the transactions are consecutive stores that are converted to coherency-only KILL transactions due to store miss merging.

The sequence and timing of events needed to sensitize the errata are quite complicated and rely on two critical timing windows occurring.

A transaction is written into the first entry of the L2MQ. Two more transactions are written into the second and third L2MQ entries and are subsequently canceled, leaving two empty holes in the L2MQ. One more transaction is written into the fourth entry of the L2MQ. The first transaction then starts and completes an address tenure.

In the first of the two critical timing windows, the fourth transaction starts an address tenure, is cancelled, and is eliminated from the queue before completing the address tenure. $\overline{\text{ARTRY}}$ is asserted for this address tenure that was just eliminated from the L2MQ. The queue is now in an undesirable state.

In the second of the two critical timing windows, another transaction is dispatched to the L2MQ during the last core cycle of the $\overline{\text{ARTRY}}$ bus cycle indicated above. This is allocated the L2MQ entry previously allocated by the first transaction. Due to the state of the L2MQ at the time, this transaction will not immediately make a bus request. Because the L2MQ is in an undesirable state, any further transactions that are queued to the L2MQ may go to the system bus out of program order or they may hang the processor.

Projected Impact:

May be present in MPX bus systems.

Work Arounds:

Any of the following will serve as a work around to this erratum:

1. Set the speculative disable bit (bit 22) in HID0 to ‘1.’ This will disable data cache and instruction cache speculation. Also, set the non-global broadcast enable bit (bit 22) in MSSCR0 to ‘1.’ This will force all non-coherent transactions to the system data bus.
2. Set the L2MQ_SIZE bit (bit 14) in MSSCR1 to ‘1.’ This will limit the L2 miss queue to 1 entry and will avoid the scenario described above.
3. Run the processor in 60x mode. The problem is not present in 60x mode.

Projected Solution:

Fixed in Rev. 2.8.

Error No. 15: Data stream touch may cause duplicate data cache tags

Overview:

A Data Stream Touch instruction that accesses the data cache in the same cycle that a canceled speculative load causes a reload of the same line to the data cache may create duplicate data cache tags.

Detailed Description:

A data stream touch (DST, DSTT) or data stream touch for store (DSTST, DSTSTT) instruction to cache line A is dispatched and sets up a data stream engine. This data stream touch is followed by a conditional branch that is mispredicted taken. The instruction at the target of the mispredicted branch is a load to non-overlapping bytes of cache line A. The load misses in the cache and allocates a data reload table entry. The reload request from the load hits in the L2.

In cycle 0, the L2 completely returns the last beat of data for the speculative load request and this data is written into the data reload data buffer. Also in this cycle, the branch is resolved.

In cycle 1, the speculative load is marked as ready to reload the dL1, a speculative cancel is issued due to the mispredicted branch, and the data stream touch starts an access at the dL1.

In cycle 2, the reload for the speculative load starts an access at the dL1. The speculative cancel in the previous cycle causes the needs_reload bit for the speculative load data reload table entry to be reset. The data stream touch instruction has progressed to the second stage of the dL1 pipeline and checks all reload table entries for an address collision. Because the needs_reload bit for the speculative reload has been reset, the address collision test fails and the data stream touch allocates a new data reload table entry.

In cycle 3, the second half of the speculative reload finishes. Cache line A is now valid in the cache and in the data reload table.

A subsequent access to the same cache line will identify the data reload table and/or data cache duplicate tag.

Projected Impact:

May be present in any system that uses the DST(T)/DSTST(T) instructions.

Work Arounds:

Any of the following will serve as a work around to this erratum:

1. Set the NOPDST bit (bit 30) in HID0 to ‘1.’ This will effectively no-op all DST(T)/DSTST(T) instructions.
2. Set the DRLT_SIZE bits (bits 3–4) in MSSCR1 to ‘01.’ This will set the data reload table to two entries. The data stream engine will only access the cache if the data reload table has both entries free.
3. Identify the problem by setting the EIEC bit (bit 13) in HID0 to ‘1’ and enabling machine check interrupts. Schedule a hardware flush of the cache on identification of the problem and continue. This solution may only be appropriate in systems where there is no snooping traffic to the affected cache line.

Projected Solution:

Fixed in Rev. 2.8.

Error No. 16: Incorrect condition code on mismatched LWARX/STWCX pair

Overview:

A STWCX instruction may be performed without setting the condition code if the store hits in the L2 and the LWARX instruction that set the reservation is to another coherency granule.

Detailed Description:

The errata requires that cache line A be resident in the L2 but not the data L1. This can happen if the line was cast out of the data L1 into the L2 for pure data accesses or if the line was loaded into the L2 at the time of instruction fetch due to instruction/data cache line sharing.

First, a LWARX instruction sets a reservation to cache line B.

Next, a STWCX instruction to cache line A misses in the L1 and then makes an access to the L2.

Then, a snoop RESERVATION KILL (rwitm, rwitm atomic, rclaim, wr w/kill, wr w/flush, wr w/flush atomic, kill) transaction that matches the reservation address (cache line B) kills the reservation.

If the above conditions are met, there is a one cycle window in which the STWCX can successfully perform the store and yet report an unsuccessful condition code. Use of mismatched LWARX/STWCX pairs is highly discouraged.

Projected Impact:

May be present in any system which uses mismatched LWARX/STWCX address pairs.

Work Arounds:

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

1. Avoid STWCX instructions that do not match the current reservation address or force a DCBF to the STWCX address before a STWCX that does not match the current reservation address.
2. Turn off the L2. The problem will not occur if the L2 is disabled.

Projected Solution:

This may be fixed in a future revision.

Error No. 17: TLBSYNC may hang in the presence of a DST

Overview:

The MPC7400 may not make forward progress if a DST instruction has caused an MMU tablewalk, that MMU tablewalk was marked by a TLBIE instruction, and a TLBSYNC instruction is pipelined the cycle after the MMU tablewalk accesses the dL1 cache.

Detailed Description:

The errata requires that a DST engine be active during a TLBSYNC instruction. The DST engine must also have been activated in the same context as the TLBSYNC instruction, meaning that it was initiated while the processor was in the privileged state.

First, the DST engine makes a request for an address that initiates an MMU tablewalk.

Then, a TLBIE instruction marks the MMU tablewalk instruction because of a potential change in translation. A mark will identify all memory requests that were translated before the TLBIE instruction was executed. Because all TLBIE instructions are snooped from the processor bus regardless of the processor that initiated the transaction, the source of the TLBIE instruction is not relevant.

Finally, a TLBSYNC instruction must follow the MMU tablewalk back-to-back in the pipeline. Because the TLBSYNC is always high priority in the L1OPQ, this opens a one cycle window where the TLBSYNC may pass the marked MMU tablewalk at arbitration to the L2. Once the TLBSYNC has passed the marked tablewalk, it will progress to the system bus. This processor will then infinitely self-ARTRY the TLBSYNC instruction because of the marked MMU tablewalk instruction. The system will be in a livelock condition.

Projected Impact:

Any system which has an active DST engine while executing a TLBSYNC instruction in a privileged context.

Work Arounds:

Insert a DSSALL instruction before a TLBSYNC instruction.

Projected Solution:

This may be fixed in a future revision.

Error No. 18: Queueing six transactions to secondary bus may hang the system

Overview:

Queueing six transactions from a single MAX processor to a secondary bus could use all data transaction queue resources and hang the system if forward progress cannot be made by allowing MAX to complete at least one outstanding transaction.

Detailed Description:

This errata details a scenario where a system may not make forward progress. The errata requires a series of queued transactions to a secondary bus. For this situation to occur, the MAX processor must also be required to snoop a transaction from another device connected to a secondary bus. The errata can affect both uni-processor and multiprocessor implementations.

First, the MAX processor initiates, and the system chipset logic accepts, six transactions to a secondary bus. These transactions do not need to be consecutive, although all six transactions need to have completed their address tenure without completing a data tenure or a tenure on the secondary bus. The six transactions fill up the six entry data transaction queue (DTQ) in the MAX processor, making it impossible for the processor to issue or snoop any more transactions until at least one of its DTQ entries is freed by the completion of an outstanding (pipelined) data tenure.

Then, the system chipset requires that a memory transaction, which was already accepted or posted to the secondary bus, make an address transaction on the processor system bus. Because the DTQ of the processor is full, there is no room for a data tenure should the MAX processor need to push modified data in response to the snooped transaction. The MAX processor will assert $\overline{\text{ARTRY}}$ to the snooped transaction until one of the queued transactions finishes a data tenure, allowing the processor to complete the snoop copyback.

If the system logic is unable to allow a data tenure for any of the queued transactions from the processor until the snooped write has completed, neither the Max processor nor the secondary bus can proceed until the other is complete. This could result in a deadlock scenario.

Projected Impact:

Any system which allows 6 outstanding transactions from a single processor and which has a secondary bus with the characteristics detailed above. While few system logic designs would have such characteristics, those which cannot retry transactions from devices connected to the secondary bus might show susceptibility to this errata.

Another possible design which might be affected would have an arbitration scheme such that the device the MAX processor is snooping is always given higher priority than the target device(s) of the MAX processor’s queued transactions.

Finally, any system in which a snooped transaction will prevent the execution of any other transactions until it completes can also be impacted.

Work Arounds:

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

1. Limit the number of outstanding transactions to a secondary bus from a single processor to five in system logic.
2. Mark the memory space on the secondary bus as guarded and avoid DST(ST)(T) and LMW instructions.

Projected Solution:

This may be fixed in a future revision.

Error No. 19: Consecutive interrupts may be nonrecoverable

Overview:

Consecutive performance monitor, decrementer, or thermal management interrupts may become nonrecoverable.

Detailed Description:

The performance monitor, decrementer, and thermal management interrupts are all gated by the external interrupt enable bit of the MSR (bit 16). Once processing of one of these interrupts is started, the hardware will reset MSR(EE) = 0 to prevent taking a second interrupt during the processing of the first interrupt. The process of resetting the MSR(EE) bit takes two processor clocks.

The priorities for these interrupts are as follows:

1. PFM—Performance Monitor Interrupt
2. DEC—Decrementer Interrupt
3. TMI—Thermal Management Interrupt

If a higher priority interrupt request occurs the first cycle of lower priority interrupt processing, the machine may appear to start processing the lower priority interrupt and then switch to processing the higher priority interrupt. In addition, the SRR0 register will contain the address of the lower priority interrupt vector and the SRR1 register will contain the MSR values normally seen during interrupt processing, including MSR(EE) = 0 and MSR(RI) = 0. This results in the loss of the last instruction address, effectively making the interrupt non-recoverable.

Note that other interrupts maskable by the MSR(EE) bit including the System management interrupt and the external interrupt are not affected by this errata.

Projected Impact:

Any system which enables a combination of performance monitor, decrementer, or thermal management interrupts at the same time.

Work Arounds:

Avoid enabling any combination of performance monitor, decrementer, or thermal management interrupts at the same time.

Projected Solution:

This may be fixed in a future revision.

Error No. 20: Disabling L2 cache may hang processor

Overview:

The MPC7400 may hang if the L2 cache is disabled during an outstanding instruction fetch.

Detailed Description:

The problem centers around the interaction of the instruction cache and the L2 cache as the L2 cache is disabled. The scenario is as follows:

1. An ifetch misses in the icache and allocates a reload table entry.
2. When the instructions return from the BIU, they are forwarded around the icache and dispatched as well as written to the IRLDQ.
3. One of these instruction is a **mtspr** targeting the L2CR. This instruction disables the L2.
4. When all beats of data return to the IRLDQ, the IRLT arbitrates to reload the L2. Because the L2 is now disabled, it does not expect reload requests from the IRLT.
5. The unexpected reload request is mishandled by the L2 and passed to the BIU as an ifetch miss.

Projected Impact:

Any system which executes code to disable the L2 cache.

Work Arounds:

1. Mark the code that disables the L2 cache as cache inhibited.
2. Preload the code that disables the L2 cache into the instruction cache before execution. This requires the code be structured in such a way that the instruction fetch be completed before the mtspr is executed. For example:

```
offset   instruction

1C       branch to offset 3C      /* preload mtspr without executing */

20       mtspr 1017, r?           /* disable L2 */
24       sync
28       isync
2C       branch to offset 60
3C       branch to offset 40

40       sync
44       isync
48       branch to offset 20

60       next sequential fetch
```

Projected Solution:

This may be fixed in a future revision.

Error No. 21: Timebase or decrementer may lose accuracy in nap mode

Overview:

The timebase counter or decrementer may lose accuracy when transitioning from the nap power saving mode.

Detailed Description:

The nap power saving mode is entered when the processor asserts $\overline{\text{QREQ}}$ and the system logic responds with $\overline{\text{QACK}}$. The timebase counter and decrementer switch from being clocked by the normal gclk distribution tree to the pll gclk2 distribution tree. The normal gclk distribution tree is shut off to conserve power.

When the nap power saving mode is exited, the gclk distribution tree is re-started and the timebase counter and decrementer switch back to being clocked by the normal gclk distribution tree. The nap power saving mode can be exited in one of the following ways:

1. Deasserting $\overline{\text{QACK}}$ to allow the processor to snoop.
2. Asserting $\overline{\text{INT}}$, $\overline{\text{SMI}}$, or $\overline{\text{MCP}}$.
3. Taking a decrementer, performance monitor, or thermal management interrupt.
4. Asserting hard or soft reset.

If any of the first three categories of events happen in a six processor-clock cycle window after having entered the nap state, the timebase counter and decrementer may miss clock ticks for the remainder of the six processor-clock cycle window. This will not cause harm to the operation of the processor, but may cause a loss of accuracy in the timebase counter or decrementer.

Projected Impact:

This can be an issue for any system which uses the nap power saving mode and requires absolute accuracy from the timebase counter or decrementer. For instance, if a timebase is maintained across two processors in a multiprocessing system, there can be scenarios where one processor may appear to drift with respect to the other processor.

Work Arounds:

1. Avoid the use of nap mode, or
2. Avoid using the timebase counter or decrementer for highly accurate measurements.

Projected Solution:

This may be fixed in a future revision.

Error No. 22: Memory access problem using COP EXMEM

Overview:

Accessing memory using the ‘cop exmem’ command may fail randomly.

Detailed Description:

Several clock regenerators that need to be involved in Exmem operation receive the wrong c1_test signal. This means that information in these latches is lost during an Exmem shift. Hence, accessing memory using the ‘cop exmem’ command may fail in a random fashion.

Projected Impact:

This can be an issue for any system which uses COP to access memory.

Work Arounds:

Use LSRL instead of EXMEM to access memory.

Projected Solution:

This may be fixed in a future revision.

Error No. 23: IFTT mode does not identify dcbt/dst instructions as data fetches

Overview:

The Instruction Fetch Transaction Type (IFTT) encoding differentiation mode does not correctly identify **dcbt** or **dst** instructions as data fetches.

Detailed Description:

When the HID0[IFTT] bit is set, the MPC7400 processor can differentiate instruction fetches from data fetches using their TT encoding on the system bus. Data fetches are identified as READ ATOMIC (TT = 11010), while instruction fetches are identified as READ (TT = 01010).

Touch-for-load instructions including **dcbt** and **dst** are identified as READ (TT = 01010) regardless of the setting of the HID0[IFTT] bit. These instructions will perform as expected from a processor perspective regardless of their TT encoding on the system bus.

Projected Impact:

This can be an issue for any system which depends on IFTT to differentiate instruction from data fetches.

Work Arounds:

1. Replace **dcbt** instructions with **dcbtst** and replace **dst** instructions with **dstst**. Note that this could have the detrimental effect of lowering performance because loads cannot fold to touch-for-store instructions. See the MPC7400 User's Manual, Section 3.5.3.2, for a caution on touch-for-store instructions.
2. Set HID0[NOPTI] and HID0[NOPDST] to no-op all touch instructions.
3. Remove **dcbt** and **dst** instructions from the code.

Projected Solution:

This may be fixed in a future revision.

Error No. 24: L2 global invalidate may not clear way 0 after L2 usage

Overview:

The L2 cache global invalidate function may not clear way 0 of the L2 cache after L2 cache usage. The L2 cache global invalidate function operates correctly if it is performed before the L2 cache has been enabled for the first time.

Detailed Description:

The MPC7400 supports global (not flash) invalidation of the L2 cache through the L2CR[L2I] parameter. Setting L2CR[L2I] causes a global invalidation of the L2 cache. A global invalidation is performed by automatically sequencing through the L2 cache tags and clearing all bits of the tag (tag data bits, tag status bits, and FIFO bit) for each of the two ways: way 0 and way 1.

When sequencing through the L2 cache tags for way 0 and way 1, the L2 global invalidate way select is masked by an internal latch. Depending on the state of the internal latch, the L2 global invalidate function may not clear the tags for way 0 of the L2 cache.The L2 global invalidate function will always correctly clear way 1 of the L2 Cache.

The state of this internal latch is always correct after the assertion of $\overline{\text{HRESET}}$. Setting L2CR[L2I] after an $\overline{\text{HRESET}}$ assertion, but prior to enabling the L2 cache, will correctly invalidate all of the L2 tags. Using the L2 global invalidate after enabling the L2 may not invalidate all of the L2 tags.

Projected Impact:

This can be an issue for any system which uses the L2 global invalidate function after the L2 cache has been enabled for the first time.

Work Arounds:

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

1. Limit use of the L2 global invalidate function to before the L2 cache has been enabled for the first time.
2. Use L2 cache hardware flush. Using L2CR[L2HWF] will correctly invalidate all of the L2 tags, but this feature will maintain coherency. The system should be able to handle possible castouts of modified data. Once the L2 cache has been enabled and used, there is no way to guarantee the invalidation of the L2 tags without maintaining coherency.

Projected Solution:

This may be fixed in a future revision.

Error No. 25: Live-lock when in PLL bypass mode

Overview:

When running in PLL bypass mode, when a self-snoopable transaction (TLBIE, ICBI, SYNC, TLBSYNC) is self-ARTRYed for any reason, it will be self-ARTRYed forever.

Detailed Description:

The MPC7400 was designed to perform all snooping operations in core to bus clock ratios of 2:1 or greater. These two cycles are needed to collect all the necessary snoop responses from around the chip and drive the ARTRY, SHD0, and SHD1 pins as necessary.

In PLL bypass mode, the core and system clocks are the matched, which results in a single internal snoop cycle per system clock. The problem centers around a hold latch in the ARTRY logic that is normally reset in the second internal snoop cycle. Once this latch is set for a self-snooped transaction, there is no way to clear it.

The value in this hold latch will then propagate to the pins, causing a false single-cycle assertion on the ARTRY pin. The part will then react as if it had asserted ARTRY again, causing an infinite cycle of ARTRY assertions for the self-snooped transaction. Note that this false single-cycle assertion of ARTRY could result in problems for other processors in an MP system.

This problem can also occur with the SHD0 and SHD1 pins, although it is not expected to result in system issues.

Projected Impact:

This can be an issue for any system that uses the nonstandard, PLL bypass mode of operation and self-snoopable transactions (TLBIE, ICBI, SYNC, TLBSYNC) that may be self-ARTRYed.

In PLL bypass mode, a self-snoopable transaction that is self-ARTRYed for any reason will be self-ARTRYed forever. In addition, the false single-cycle assertion of ARTRY caused by this issue could also result in problems for other processors in an MP system.

Work Arounds:

Avoid using TLBIE, ICBI, SYNC, TLBSYNC in PLL bypass mode, or implement a synchronization routine such that these instructions never have a reason to be retried.

Projected Solution:

This erratum will not be fixed in future revisions of the MPC7400. This is a processor limitation for this nonstandard mode of operation (PLL bypass mode).

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-480-768-2130
(800) 521-6274

JAPAN:

Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu Minato-ku
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre, 2 Dai King Street
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

(800) 521-6274

HOME PAGE:

www.motorola.com/semiconductors

MPC7400CE/D