This document contains errata and design notes that affect designs using the PPC405D5X1, PPC405D5X2 and PPC405F6X1. Each erratum includes an overview, a description of the system impact and a description of possible work-around(s). Design notes cover items that are not considered errata but need a description beyond what is provided in published documentation. Table 1 identifies errata affecting cores by version. Tables 2 and 3 contain a summary of all errata and design notes.

Items are listed in numeric order. Gaps in the numeric sequence are intentional; only applicable items are included in this document.

Category definitions:

Errata are classified according to system impact and availability of a work-around.

1. Major impact, no workaround is available. A problem is said to have a major impact if it results in a system crash, a hard failure, an unrecoverable soft failure, significant performance degradation, or the storage of incorrect data.
2. Major impact, workaround is impractical to implement, or a substantial risk of encountering the same or additional problems, including performance issues, exist after the workaround is implemented.
3. Major impact, workaround available. Application of the workaround either eliminates the problem, or reduces it to a minor impact issue.
4. Minor impact, no workaround is available. Minor impact problems result in slight to moderate performance degradation, or are a functional variance from specification.
5. Minor impact, workaround is available. Minor impact problems result in slight to moderate performance degradation, or are a functional variance from specification.
6. Design enhancement.

Errata Summary

Table 1. List of all errata and core(s) affected

Issues	PPC405D5X1	PPC405D5X2	PPC405F6X1
CPU_121	X	X	X
CPU_147	X	X	X
CPU_153	X	X	X
CPU_162	X	X	X
CPU_163	X	X	X
CPU_187	X	X	X
CPU_197	X	X	X
CPU_208	X	X	X
CPU_210	X		
CPU_211	X	X	
CPU_212	X	X	X
CPU_213	X	X	

Table 2. Errata Summary

Item	Category	Description	Date First Docu-mented	Date Last Updated
CPU_121	3	The iccci instruction may errantly cause a Data TLB Exception.	4/19/99	4/19/99
CPU_147	3	Virtual memory marked as non-executable storage (storage attribute EX=0) can be loaded into the instruction cache using the icbt instruction.	8/12/99	4/27/00
CPU_153	5	Floating point enabled exceptions do not update CR1 field of the CR.	9/13/99	4/26/00
CPU_162	3	When in real mode, the 405 core may errantly make speculative instruction fetches from guarded storage.	1/7/00	2/24/00
CPU_163	3	Floating point enabled exception handlers cannot reliable determine if SRR0 points to the excepting instruction.	1/17/00	4/26/00
CPU_187	3	Access to guard storage via APU load/store double-word or APU load/store quad-word instructions is not architecturally compliant.	4/12/00	5/10/00
CPU_197	3	Incorrect real mode attributes may be used when accessing the last instruction in a 128 MB region.	11/17/00	11/17/00
CPU_208	3	icbt instructions executed with data relocation enabled may cause incorrect instruction execution if the icbt misses in the UTLB or does not have permission to access the page.	8/21/01	8/21/01
CPU_210	2	Interrupted stwcx. instructions may errantly write data to memory under certain DCU conditions.	8/30/01	8/30/01

Table 2. Errata Summary

Item	Category	Description	Date First Docu-mented	Date Last Updated
CPU_211	3	When deterministic multiplication is enabled (Core Input TIEC405DETERMINISTICMULT = 1), the mulhw[.],mullw[o][.] and mulli instructions may generate wrong result.	6/26/02	6/26/02
CPU_212	3	When attached to a multi-cycle OCM controller, the PPC405 core may assert the C405_dsocmWait signal during the assertion of DSOCM_c405Hold or DSOCM_c405Complete for an OCM store operation. The PPC405 will ignore a valid DSOCM_c405Complete signal for an OCM store while asserting C405_dsocmWait.	9/27/02	10/18/02
CPU_213	3	Incorrect data may be flushed from the data cache.	2/21/03	5/7/03

Design Note Summary

Table 3. Design Summary

Item	Description	Date First Docu-mented	Date Last Updated
	There are no Design Notes for the PPC405D5X1, PPC405D5X2 or PPC405F6X1.		

CPU_121 The iccci instruction may errantly cause a Data TLB Exception.

Category: 3

Overview:

When data-side relocation (data address translation) is enabled (MSR[DR] = 1), an iccci instruction errantly attempts an access check for the associated page. Since iccci invalidates the entire instruction cache, the effective address it generates is unnecessary.

Impact:

When data-side relocation (data address translation) is enabled, the execution of an iccci may cause a Data TLB miss exception.

Work-around:

There are two possible work-arounds. Work-around 1 avoids this erratum by temporarily disabling data address translation. Work-around 2 describes how to handle this erratum without disabling data address translation.

1. Before executing an iccci instruction, make sure the MSR[DR] is disabled. This can be done using the following pseudo code:

```
mfmsr Rx                    ! Rx is a scratch reg
andi  Ry,Rx,DR_MASK         ! clear MSR[DR] in scratch reg Rx
mtmsr Ry
isync
iccci 0,Rx                  ! The address does not matter.
mtmsr Rx
isync
```

2. When data-side relocation is enabled, ensure that the virtual address generated by the iccci (virtual address = {PID, effective address (RA | 0 + RB)}) has a corresponding page in the TLB.

CPU_147 Virtual memory marked as non-executable storage (storage attribute EX=0) can be loaded into the instruction cache using the icbt instruction.

Category: 3

Overview:

The icbt instruction should execute as a nop (no operation) when the effective address corresponds to a memory page marked as non-executable. Instead, an instruction cache line fill occurs when the effective address of the icbt instruction maps to memory region having the following three characteristics:

1. Marked as non-executable storage (storage attribute EX = 0).
2. Marked as cacheable (storage attribute I = 0).
3. Access is not prohibited by a zone fault (The access control field ZSEL references a ZPR field that does not prohibit access).

Impact:

Touching data belonging to a page marked as non-executable storage into the instruction cache with the icbt instruction can result in unnecessary memory accesses. Note that:

1. Memory marked as non-executable (EX = 0) cannot be executed even if loaded into the instruction cache.
2. icbt instructions are not compiler generated; they are isolated to assembly routines.

Work-around:

1. No work-around is necessary if either:
   a. The translation from virtual to real does not change.
   b. There are no occurrences where an icbt instruction causes a cache line fill of data from a page marked as non-executable storage.
2. Invalidate cache blocks (lines) loaded with data belonging to a page marked as non-executable storage. Use either an icbi or an iccci instruction.

CPU_153 Floating point enabled exceptions do not update CR1 field of the CR.

Category: 5

Overview:

Floating Point Enabled Exceptions (excluding Enabled Invalid Operation and Enabled Zero Divide Exceptions), that occur on floating point instructions with the RC bit set, architecturally should update the CR1 field of the CR, but do not.

Impact:

CR1 of the CR cannot be relied upon by the floating point enabled exception handler to describe the excepting floating point operation.

Workaround:

1. None is required if a or b is true:
   a. The floating point enabled exception handler is not used
   b. The FPSCR, not the CR1, is used by the exception handler.
2. If the floating point enabled exception handler is used, the interrupt handler must examine the instruction pointed to by SRR0 to determine if a CR update is needed and if so which CR field.

CPU_162 When in real mode, the 405 core may errantly make speculative instruction fetches from guarded storage.

Category: 3

Overview:

In real mode, if instructions (guarded storage or not) and memory mapped I/O (guarded storage) are within 1KB of each other, it is possible for the I/O to be speculatively accessed when the instructions are executed.

Impact:

Memory mapped I/O (MMIO) may be speculatively accessed. An unintentional access to MMIO could result in a loss of data.

Workaround:

Maintain at least 1KB of separation between instructions and memory mapped I/O.

CPU_163 Floating point enabled exception handlers cannot reliable determine if SRR0 points to the excepting instruction.

Category: 3

Overview:

When a floating point enabled exception exists while the MSR[FE0] and MSR[FE1] bits are off and either or both of the MSR[FE] bits are re-enabled via a mtmsr, rfi, or rfci instruction, the floating point enabled exception handler has no way of determining whether the exception was caused by the instruction pointed to by SRR0 or some earlier instruction.

Impact:

Floating point enabled exception handlers cannot reliable determine if SRR0 points to the excepting instruction.

Workaround:

1. None required if floating point enabled exceptions are not enabled via the FPSCR.
2. Perform a and b:
   a. Ensure no floating point instructions immediately follow a mtmsr that turns on either of the MSR[FE] bits. Architecturally an isync instruction should always follow a mtmsr.
   b. Before executing rfi or rfci instructions that re-enable the MSR[FE] bits ensure that there are no outstanding floating point enabled exceptions. Interrupt handlers that have no floating point instructions should not need to check the FPSCR.

CPU_187 Access to guard storage via APU load/store double-word or APU load/store quad-word instructions is not architecturally compliant.

Category: 5

Overview:

Architecturally a data aligned load or store (not including multiples or strings) to storage marked as guarded should not be accessed speculatively. The 405 core breaks double and quad-word operations into 2 and 4 word transfers respectively. As an example, consider aligned double-word transfers to guarded storage. Once the first word of a double-word transfer has reached load write-back (LWB) it is uninterruptible. If the second piece is still in execute (EXE) or in write-back (WB) pipeline stage, it may be interrupted due to an asynchronous interrupt. When the interrupt handler returns to the interrupted load double-word, the instruction is restarted from the beginning causing the guarded storage location accessed by the first piece of the load double word to be accessed a second time. Similarly, for double-word store instructions that are interrupted immediately after the first piece of data has left the write-back stage, returning from the interrupt handler will restart the instruction causing the first piece to be accessed from the guarded location twice.

If the APU load/store double-word or APU load/store quad-word involves 32-bit or smaller RT or RS registers such that the double or quad-word transfer resembles a load/store multiple, then the guarded requirement does not apply. If the APU load/store double-word or APU load/store quad-word involves 64 or 128-bit registers respectively, then the guarded requirement does apply and accessing the guarded location twice is not architecturally compliant. Currently this is a problem for floating point load/store double-word operations to guarded storage.

Impact:

APU load/store double-word or APU load/store quad-word operations to 64-bit or 128-bit registers respectively that have a data-side effective address to guarded storage may access the guarded storage twice in presence of asynchronous interrupts.

Workaround:

Do not issue APU/FPU double or quad word storage instructions to guarded storage.

CPU_197 Incorrect real mode attributes may be used when accessing the last instruction in a 128 MB region.

Category: 3

Overview:

When executing instructions in real mode (MSR[IR] = 0), an access to the last instruction in a 128 MB region uses the real mode attributes of the next consecutive 128 MB region under any of the following conditions:

1. The last instruction in a 128 MB region contains a branch target that is non-cacheable or causes a cache miss.
2. The address restored by an rfi or rfci is the last instruction in a 128 MB region and this address is non-cacheable or causes a cache miss.
3. The next to the last instruction in a non-cacheable 128 MB region contains a branch which is predicted taken but is not taken.
4. The next to the last instruction in a non-cacheable 128 MB region contains an isync instruction.

Note:

1. Real mode attributes are specified by the ICCR, SU0R, and SLER registers.
2. All 128 MB regions have the same storage attributes after reset. Therefore, a branch instruction at the reset vector 0xFFFFFFFC is not affected by this erratum after a core, chip, or system reset.
3. In real mode, the storage regions wrap making the last storage region (0xF8000000 - 0xFFFFFFFF) and the first storage region (0x00000000 - 0x07FFFFFF) consecutive.

Impact:

Table 3 lists the impact of encountering one of the conditions listed above. The first column contains the real mode storage attribute of the 128 MB region being accessed and the second column contains the real mode storage attribute of the next consecutive 128 MB region.

Table 4. Description of Impact for Item CPU_197

Request from First Region	Request assumes the attribute of Second Region	Impact
Non-cacheable	Cacheable	An instruction cache line fill from the non-cacheable storage region may occur.
Cacheable	Non-cacheable	A desired instruction cache line fill from the cacheable storage region may not occur.
Big Endian	Little Endian	The request from the big endian storage region is treated as little endian storage. A program exception may be generated, or an incorrect instruction may be executed.
Little Endian	Big Endian	The request from the little endian storage region is treated as big endian storage. A program exception or machine check exception may be generated.
Non-compressed Storage Region	Compressed Storage Region	The request from the non-compressed storage region is treated as compressed storage. A program exception may be generated.
Compressed Storage Region	Non-compressed Storage Region	The request from the compressed storage region is treated as non-compressed storage. A program exception may be generated.

CPU_197 Continued

Work-around:

1. No work-around is required if any of the following apply:
   a. Code does not exist in the last two word locations of a 128 MB region.
   b. The real mode storage attributes of any 128 MB region containing executable code are identical to the attributes for the next contiguous 128 MB region.
   c. The last two words of a 128 MB region are only accessed while in virtual mode (MSR[IR] = 1).
2. Do not place code in the last word locations of a 128 MB region.
3. When performing a soft reset, branch directly to the entry point of the application code. Do not branch to the reset vector (0xFFFFFFFC). Unlike a core, chip or system reset, a soft reset does not guarantee that all 128 MB aligned regions have the same storage attributes. The branch at the reset vector in the last storage region (0xF8000000 - 0xFFFFFFFF) will obtain the storage attributes of the first storage region (0x00000000 - 0x07FFFFFF). If these storage attributes differ, unexpected results are possible.

CPU_208 icbt instructions executed with data relocation enabled may cause incorrect instruction execution if the icbt misses in the UTLB or does not have permission to access the page.

Category: 3

Overview:

icbt instructions executed with data relocation enabled (MSR[DR] = 1) may cause incorrect instruction execution if the icbt misses in the UTLB or does not have permission to access the page.

For the icbt to cause the instruction cache to deliver incorrect instructions to the fetcher a number of events must line up:
Conditions:

1. Data relocation is enabled (MSR[DR] = 1).
2. Either condition 2a or 2b is true.
   a. The TLB page referenced by the icbt instruction is not found in the UTLB.
   b. The TLB page referenced by the icbt instruction is marked protected by its ZPR settings.
3. A cache line fill completes while the execute logic is requesting the instruction cache unit to perform an icbt.
4. The fetcher has to be requesting an address for a new cache line followed by a request for the previous cache line. This condition occurs when the fetcher re-requests data thrown away because there was no room in the fetch queue.
5. In the same cycle as conditions 3 and 4, the icbt must be presented to the instruction cache. The instruction cache must not accept the fetch request this cycle, but does accept the icbt.

Impact:

An incorrect instruction may be executed after execution of an icbt instruction.

Note, this erratum does not affect compiler generated code. The icbt instruction is not a compiler generated instruction.

Work-around:

Perform one of the following:

1. Ensure data relocation is disabled (MSR[DR] = 0) when executing an icbt instruction.

or

2. When data relocation is enabled, ensure the TLB page referenced by an icbt instruction is in the UTLB and not protected by access control settings.

or

3. When data relocation is enabled, execute an isync instruction after every icbt instruction.

CPU_210 Interrupted stwcx. instructions may errantly write data to memory under certain DCU conditions.

Category: 2

Overview:

There are three resources that an aligned stwcx. instruction with write permission can alter:

1. The storage location at the data-side effective address if the reservation bit is set.
2. The CR field 0 (CR0) to indicate the success or failure of the store to the data-side effective address.
3. The reservation bit is cleared.

The PPC405 core breaks the stwcx. into two pieces in the EXE stage. The first piece moves to WB stage and performs the access check and the store operation if the reservation bit is set. If the first piece does not cause a storage exception the second piece in EXE updates the CR0 as it moves to the WB stage and clears the reservation bit when it is in the WB stage. Under certain data cache unit (DCU) conditions the first piece of the stwcx. can become immune to interrupts by leaving the WB stage while the second piece of the stwcx. remains susceptible to interrupts because it is stalled in the EXE stage. If the second piece is then interrupted by an asynchronous interrupt, the reservation bit and CR field 0 are not updated and the data is errantly written to memory if the reservation bit were set.

Impact:

Cannot reliably use stwcx. instructions in the presence of asynchronous interrupts under certain DCU conditions. Note that compilers do not generate stwcx. instructions.

Work-around:

Perform one of the following:

1. No work-around is required if stwcx. instruction is not used.
2. No work-around is required if asynchronous interrupt handlers are guaranteed to return to the interrupted stwcx. without having executed any other lwarx or stwcx. instructions. However, operating systems which use asynchronous interrupts to perform task switching usually cannot be relied upon to exhibit such behavior.
3. For stwcx. instructions that are executed in supervisor mode (MSR[PR]=0), mask asynchronous interrupts (MSR[CE,EE,ME,DE]) before executing the stwcx. and unmask asynchronous interrupts immediately after executing the stwcx..

   For stwcx. instructions that are executed in user mode (MSR[PR]=1), perform the following sequence to mask asynchronous interrupts before executing the stwcx. and to unmask asynchronous interrupts after

   executing the stwcx. (MSR[CE,EE,ME,DE]):

   ```
   sc (with parameter to mask asynchronous interrupts)
   stwcx.
   sc (with parameter to unmask asynchronous interrupts)
   ```

   Note that the system call handler may have to be updated to support parameters to mask and unmask asynchronous interrupts.

CPU_210 Continued

4. Perform all of the following:

   a. Insert a sync or dcbt instruction before a stwcx. instruction. If a dcbt is used, use the same RA and RB as the stwcx. instruction. The use of the dcbt instruction may result in better performance than the sync instruction.

```
dcbt RA,RB (The RA and RB are the same as used by the stwcx. instruction.)
stwcx. RS,RA,RB
```

   b.At the end of all interrupt handlers execute a sync or dcbt instruction just before executing the rfi or rfci. If the dcbt is used, ensure that the effective address will not result in a machine check. Note that a dcbt to a non-cacheable address or a dcbt to a page that is not in the UTLB or a page without read permission still provides the intended effect. (dcbt instructions do not generate DTLB Miss interrupts nor do they generate Data Storage Interrupts.)

CPU_211 When deterministic multiplication is enabled (Core Input TIEC405DETERMINISTICMULT = 1), the mulhw[.], mullw[o][.] and mulli instructions may generate wrong result.

Category: 3

Overview:

When deterministic multiplication is enabled (Core Input TIEC405DETERMINISTICMULT = 1), the mulhw[.], mullw[o][.] and mulli instructions may generate wrong result.

Impact:

Cannot reliably use mulhw[.], mullw[o][.] and mulli instructions when deterministic multiplication is enabled.

Workaround:

1. No workaround is required if deterministic multiplication is disabled (TIEC405DETERMINISTICMULT=0).
2. Do not use the mulhw[.] or mullw[o][.] or mulli instruction.

CPU_212 When attached to a multi-cycle OCM controller, the PPC405 core may assert the C405_dsocmWait signal during the assertion of DSOCM_c405Hold or DSOCM_c405Complete for an OCM store operation. The PPC405 will ignore a valid DSOCM_c405Complete signal for an OCM store while asserting C405_dsocmWait.

Category: 3

Overview:

The PPC405 core uses the C405_dsocmWait signal to ensure that load data is returned in order. When there is a load miss or load non-cacheable that is accessing the PLB, the PPC405 asserts C405_dsocmWait for any subsequent load while waiting for data to be returned for a prior load. The C405_dsocmWait signal indicates to the OCM controller that the PPC405 is waiting for a prior load to return data and that the OCM controller must continue to provide data and control for its current load until the PPC405 can accept this data. It is a violation of the OCM interface specification for the PPC405 to assert C405_dsocmWait during any hold or complete indication for an OCM store operation.

The following code sequence may cause the PPC405 to assert C405_dsocmWait during an OCM store operation. The OCM controller must be a multi-cycle OCM controller.

```
load    (miss in the d-cache or non-cacheable. Load data must not be returned before the second
         load is executed.)
.       (0 or more other instructions while waiting for the load data to return.
.
.
store   (OCM) (More than 1 OCM stores may be required depending on the OCM controller used.
.       (0 or more other instructions depending on the OCM controller used.)
.
.
load    (either OCM or d-cache)
```

Specific implementations of an OCM controller will have a more precise set of instruction sequences that can cause a failure. For example, if using a typical 2 cycle OCM controller, the code sequences can be refined as follows:

```
load    (miss in the d-cache or non-cacheable. Load data must not be returned before the second
         load is executed.)
.       (1 or more other instructions while waiting for the load data to return.
.

store   (OCM)
store   (OCM)
load    (either OCM or d-cache)
```

OR

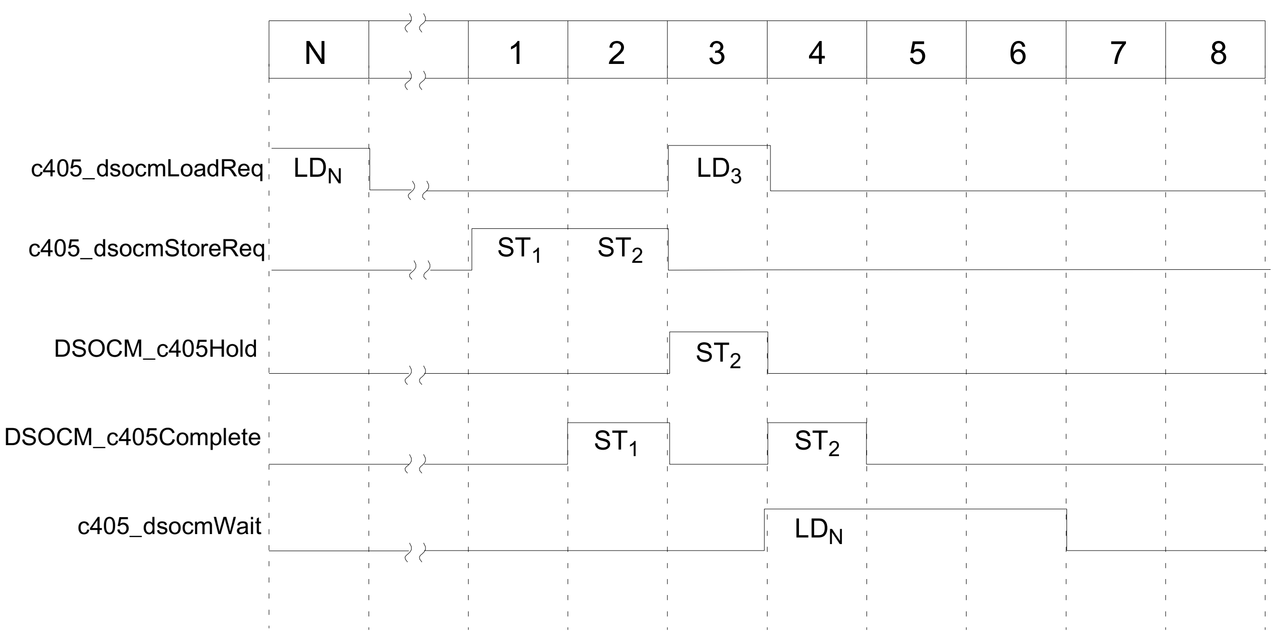
CPU_212 Continued

load (miss in the d-cache or non-cacheable. Load data must not be returned before the second load is executed.)
. (0 or more other instructions while waiting for the load data to
. return.)
.
store (OCM)
store (OCM)
store (OCM) (Need 3 or more consecutive OCM stores.)
. (0 or 1 other instructions between last OCM store and the load instruction)
load (either OCM or dcache)

Figure 1 and 2 are two examples showing C405_dsocmWait asserted for a load (LD_N) in the same cycle as DSOCM_c405Complete for a store (ST_2). In example 1, the PPC405 does not detect the DSOCM_c405Complete for ST_2 and continues to wait for a complete. In example 2, the PPC405 interprets the complete for LD_3 as the complete for ST_2. LD_3 is then treated as non-OCM access.

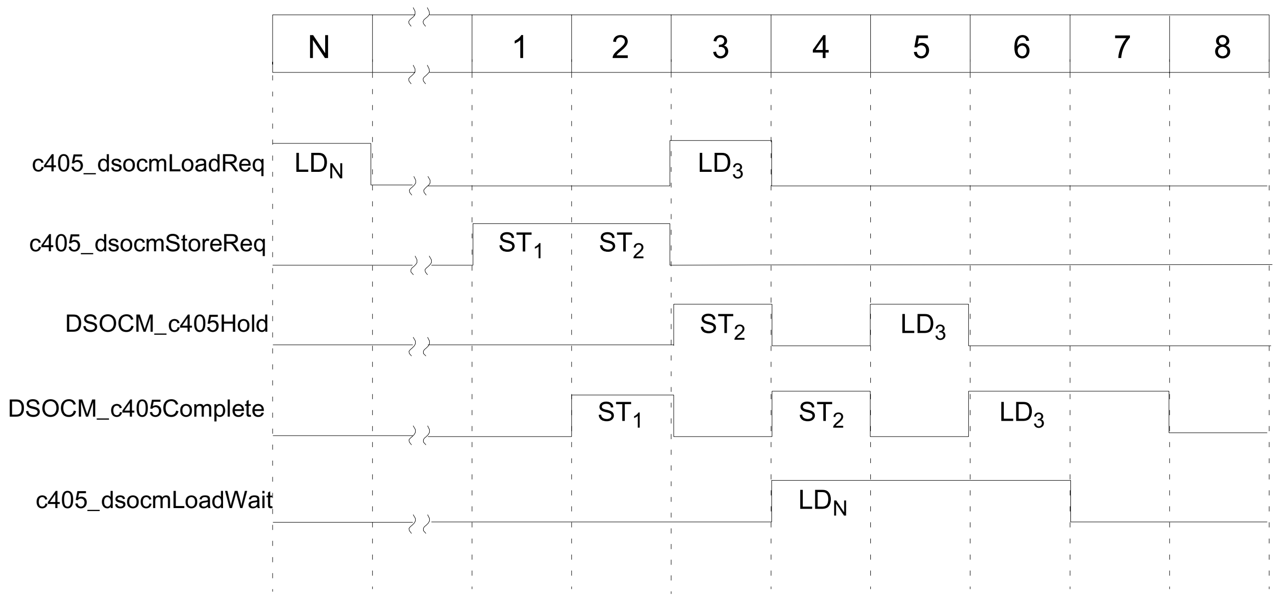
LD_N is a load from non-OCM memory.
ST_1 and ST_2 are stores to OCM memory.
LD_3 is a load from non-OCM memory in example 1 and a load from OCM memory in example 2.

Figure 1. Example 1, LD_3 is a non-OCM access.



CPU_212 Continued

Figure 2. Example 2, LD_3 is an OCM access.



The code sequences will vary depending on the specific design of the OCM controller. The more cycles of hold that the OCM controller uses the greater the number of cycles that can exist between the OCM store and a following load operation. Two stores are required for the 2 cycle OCM controller example given above because (with the OCM controller used) the first store request gives a Complete without any Hold cycles. The second store issues Hold cycles because the OCM controller is not finished with the first store.

Impact:

When attached to a multi-cycle OCM controller, the PPC405 may assert the C405_dsocmWait signal during the assertion of DSOCM_c405Hold or DSOCM_c405Complete for an OCM store operation. The PPC405 will ignore a valid DSOCM_c405Complete signal for an OCM store while asserting C405_dsocmWait. The PPC405 will be left in a state with an incomplete OCM store operation, possibly causing future OCM operations to be missed or possible PPC405 hang conditions. A specific OCM controller design may not tolerate the inadvertent assertion of C405_dsocmWait during an OCM store operation.

> **Note:** This erratum does not affect designs using the SHOCMIP_405. The SHOCMIP_405 is the OCM controller available in the Burlington IP Library and used by the PPC405GP and PPC405GPr.

CPU_212 Continued

Workaround:

Perform one of the following:

1. Set bit 1 of CCR0 during initialization or prior to any OCM store accesses. Setting this reserved bit causes the PPC405 data cache unit (DCU) to block requests when there is a d-cache miss.

   Both load and store d-cache misses block further requests and therefore, may affect performance. Since the work-around for this erratum does not require stores to block requests, the DCU can be configured as non-allocating for store operations by setting the Store WithOut Allocate bit (CCR0[SWOA]=1). In this way, the DCU blocks requests for load misses instead of both load and store misses. This configuration produces fewer situations that block requests and tends to improve performance for this workaround. However, disabling allocation may also increase the number of store misses. If store misses exceed the depth of the store queue, the pipeline stalls. Therefore, experimentation on the target application is required to determine the optimal configuration.

   Note: When modifying bit 1 of the CCR0, use code sequence 2 of the CCR0 programming guidelines in section 4.5.1 of the PowerPC 405xx Embedded Processor Core User's Manual.
2. Execute a sync instruction prior to executing any OCM stores to create a barrier between any PLB load accesses and the OCM store operations.
3. When precise code sequences are known for a specific OCM controller, split these code sequences to eliminate the possibility of the problem occurring.
4. Design the OCM controller to react to C405_dsocmWait signal for store operations in a manner similar to that required for load operations.

CPU_213 Incorrect data may be flushed from the data cache.

Category: 3

Overview:

Incorrect data may be flushed from the data cache as a result of a specific sequence of PLB operations between the PPC405 DCU and a PLB slave. For this erratum, the PLB slave must be capable of returning read data acknowledges in consecutive PLB cycles during a PLB line read.

The instruction sequence that can cause this erratum is defined by a series of operations on the PPC405 DCU PLB interface. Since these instructions cause bus operations which are much slower than the execution of the instructions themselves, the proximity in execution time of instructions and their corresponding bus operations are not fixed. The instruction sequence can be separated by other unrelated instructions and operations including branches, interrupts, and instruction cache misses; as long as, the required bus operations at the PPC405 DCU PLB interface occur in a tight back-to-back sequence.

In order to have timing between operations on the PPC405DCU PLB interface needed for this erratum, the PLB slave must be able to return read data acknowledges in back-to-back PLB cycles. Even with this required slave behavior, the erratum may not occur since it also depends on other less well defined PLB and slave timings.

The following is the sequence of bus operations as seen on the PPC405 DCU PLB interface:

1. Data Cache Flush of line1 (line write): This flush can occur anytime prior to the failing sequence of bus operations. It is only required that this flush be the last data side write to the PLB prior to encountering the following failing sequence. This is a pre-conditioning operation and not tightly coupled with the failing sequence of operations defined by items 2-5.
2. Data Cache Fill of line2 (line read): This fill replaces a dirty line and causes the flush in step 5.
3. Data Cache Fill of line3 (line read): This fill is to a different congruence class than the Data Cache Fill line2. It may or may not cause data to be replaced or flushed. The timing of the request for the read of line3 must occur on the PPC405 PLB interace before the last read data acknowledge for the read of line2. The address acknowlege for the read of line3 must not occur before 3 plb cycles after the last read data acknowledge for the read of line2. The last two read data acknowleges for the read of line3 must occur in back to back PLB cycles.
4. Write of word4 (word, halfword or byte write) non-cachable or writethru: The timing of the request on the PPC405 PLB interface for this write must occur within the one cycle window for the cycle immediately following the last read data acknowledge of Data Cache Fill of line3.
5. Data Cache Flush of line2 (line write): This flush is due to the data cache fill of line2 in step 2. The data flushed to memory is incorrect.

CPU_213 Continued

The following code sequence can cause the required sequence on the PLB interface:

1. load or store instruction that misses in the data cache and causes a line fill to the line with the LRU pointing to a valid dirty line, or a dcbf to a line with valid dirty data: This instruction causes the Data Cache Flush of line1.
2. load or store instruction that misses in the data cache and causes a line fill to a line with the LRU pointing to a valid dirty line: This instruction causes the Data Cache Fill line2 and Data Cache Flush of line2.
3. store instruction that misses in the data cache and causes a line fill: This instruction causes Data Cache Fill of line3 and may or may not result in a line flush.
4. store instruction to memory marked non-cacheable or write thru: This instruction causes the write of word4.
5. load instruction that hits in the same line as the line currently being filled by the Data Cache Fill of line3: In addition to hitting in the same line, this load must hit in the last or second from last beat, or in the last or second from the last read data acknowledge of the Data Cache Fill of line3. For a 64-bit slave, the load must hit word 4, 5, 6, or 7 of the 8-word line. For a 32-bit slave, the load must hit word 6 or 7 of the 8 word line. This load does not cause any plb bus operation, but it is required to obtain the correct timing between the request for write of word4 and the last read data acknowledge or Data Cache Fill of line3.

Impact:

If the PPC405 is in a system where PLB slaves can return read data acknowledges at the PPC405 DCU PLB interface in consecutive PLB cycles, incorrect data may be flushed (written) to memory from the data cache.

Workarounds:

No workaround is needed if none of the PLB slaves in the system can return read data acknowledges in back-to-back PLB cycles.

Perform one of the following workarounds if the system contains a PLB slave which can return read data acknowledges in back-to-back PLB cycles.

1. Set CCR0 bits 1 and 3.

   The CCR0[1] and CCR0[3] bits cause the data cache unit (DCU) to block requests generated by the CPU when certain stages of the DCU control pipeline are full. These bits block further requests when a valid request occupies the corresponding stage in the DCU control flow. When these bits are set both load and store data cache misses and load non-cachables block further requests and therefore may affect performance.

   Note: When modifying bits 1and 3 of the CCR0, use code sequence 2 of the CCR0 programming guidelines in section 4.5.1 of the PowerPC 405xx Embedded Processor Core Users Manual.
2. Mark all data memory as write-through using the DCWR register or W storage attribute of each TLB entry.

Design Note(s)

(There are no Design Notes for the PPC405D5X1, PPC405D5X2 or PPC405F6X1.)

Revision Log

Rev	Contents of Modification
Oct 7, 2002	Errata document version 4.0.0 • Item added: CPU_212 • Introduction modified (page 1)
Oct 18, 2002	Errata document version 4.1.0 • Updated work-around 1 of CPU_212
March 13, 2003	Errata document version 5.0.0 • Item added: CPU_213
May 9, 2003	Errata document version 5.1.0 • Updated: CPU_213
April 8, 2004	• Errata document version 5.2.0 • Updated to include the PPC405F6X1 • CPU_211 and CPU_213 do not apply to the PPC405F6X1.

This document is intended for hardware system manufacturers and developers of applications, operating systems, tools, firmware, and other software. It is provided to you to describe conditions under which errata may occur in IBM hardware and to enable you to conduct your own investigation into how your system and software may be impacted and to integrate workarounds as needed.

This document lists errata that IBM has characterized as of April 8, 2004. IBM does not represent or warrant that this errata list is complete. IBM may add errata to this list in the future. Please check with your IBM sales representative regularly to verify that you have the most current version of this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at http://www.ibm.com/chips

PPC405_core_5_2_0.fm.5.2.0
April 8, 2004