AMCC 405GP
PowerPC
Errata

Document Issue 1.00

September 2004

PPC405GP Embedded Processor Errata

APPLIED MICRO CIRCUITS CORPORATION

Preliminary

PPC405GP Rev E (IBM25PPC405GP-3xExxxCx)

This document contains errata and design notes that affect designs using the PPC405GP Rev E (PVR 0x40110145). 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. Refer to Tables 1 and 2 for a list of errata and design notes.

Errata are organized by item designator in alphabetical order. The item designator consists of an acronym for the affected functional unit and a numeric value. The numeric values assigned to errata and design notes are not necessarily consecutive.

List of Functional Unit Acronyms:

• CHIP	Errata particular to the chip implementation
• CPU	PPC405B3V5 processor core
• DMA	DMA controller
• DCP	Decompression Controller (CodePack™)
• EBC	External Peripheral Bus Controller
• EMAC	EMAC controller
• GPIO	General Purpose I/O controller
• IIC	IIC controller
• MAL	MAL controller
• OCM	On Chip Memory controller
• OPBA	On Chip Peripheral Bus Arbiter
• PCI	PCI controller
• PLB	Processor Local Bus Arbiter
• POB	On Chip Peripheral Bus Bridge
• SDRAM	SDRAM controller
• UART	UART controller
• UIC	Universal Interrupt Controller
• ZMII	EMAC to PHY interface bridge

Category definitions:

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

1. Major impact, no work-around 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, work-around is impractical to implement, or a substantial risk of encountering the same or additional problems, including performance issues, exist after the work-around is implemented.
3. Major impact, work-around available. Application of the work-around either eliminates the problem, or reduces it to a minor impact issue.
4. Minor impact, no work-around is available. Minor impact problems result in slight to moderate performance degradation, or are a functional variance from specification.
5. Minor impact, work-around is available. Minor impact problems result in slight to moderate performance degradation, or are a functional variance from specification.
6. Design enhancement.

PPC405GP Rev E (IBM25PPC405GP-3xExxxCx)

Preliminary

Errata Summary

Table 1. Errata Summary

Item	Category	Description	Date First Docu-mented	Date Last Updated
CHIP_11	3	When the IIC controller is operating as a slave with 10-bit addressing, the IIC controller may incorrectly detect Slave Write Needs Service.	5/22/00	5/22/00
CPU_121	5	The iccci instruction may errantly cause a Data TLB Exception.	4/19/99	4/19/99
CPU_147	5	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_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_190	3	An incorrect exception vector offset is generated when an alignment exception and Data TLB Miss exception occur at the same time.	5/11/00	5/11/00
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_200	3	Multiply accumulate (MAC) instructions having the same target register (RT) as an immediately preceding multiply accumulate instruction or multiply instruction may yield incorrect results.	2/9/01	4/24/02
CPU_201	3	Switching between virtual and real address modes may result in the execution of instructions fetched from the wrong real address.	2/16/01	7/26/01
CPU_202	3	A load string or load multiple instruction that accesses the Data Side OCM may provide incorrect data.	2/24/01	2/24/01
CPU_206	2	Unaligned store and store string operations to On-Chip Memory (OCM) may write incorrect data to the OCM.	4/27/01	4/27/01
CPU_208	1	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_209	2	A lwarx, stwcx. instruction sequence may cause a branch to link register (bclr) or branch to count register (bcctr) instruction to execute incorrectly.	8/28/01	8/28/01
CPU_210	1	Interrupted stwcx. instructions may errantly write data to memory under certain DCU conditions.	8/30/01	8/30/01
CPU_213	3	Incorrect data may be flushed from the data cache.	2/21/03	5/15/03
DCP_6	3	Executing instructions from a memory region configured as uncompressed after executing instructions from a memory region configured as compressed can cause the Decompression Controller (DCP) to generate a machine check exception or hang the PLB.	7/13/01	7/13/01
DCP_9	3	Compressed instruction blocks that cross a 1KB boundary in external memory can cause the Decompression Controller (DCP) to return incorrect instructions to the 405 processor core	7/13/01	7/24/01
EBC_5	4	Peripheral bank 4 of the external bus controller cannot be programmed as a "Write Only" bank.	11/3/99	11/3/99
EBC_20	3	No parity generated during a DMA memory to peripheral write.	4/3/03	5/15/03
EMAC_5	4	The EMAC constantly transmits a preamble pattern on the MII interface.	3/25/02	1/29/03
EMAC_7	5	Signal Quality Error (SQE) occasionally reported incorrectly during SQE test.	6/10/02	6/10/02

Table 1. Errata Summary

Item	Category	Description	Date First Docu-mented	Date Last Updated
EMAC_8	5	Soft Reset may not reset all logic in the EMAC	5/7/03	5/15/03
EMAC_9	4	Integrated Flow Control Mechanism may not function correctly	5/7/03	5/15/03
EMAC_10	4	Octet Counter Registers may not record an accurate count	5/7/03	5/15/03
IIC_6	5	Slave Transfer Count is not cleared correctly.	1/19/00	1/19/00
IIC_7	5	The IIC controller is temporarily placed in an invalid state.	1/19/00	1/19/00
IIC_9	3	Loss of Bus Arbitration while sending the address byte(s) causes an auto-retry of the transfer.	1/19/00	2/24/99
IIC_10	5	A loss of Arbitration while sending the 1st or 2nd address bit can cause the IIC Controller to malfunction.	1/19/00	1/19/00
IIC_11	5	The IIC Controller when operating as a slave may not correctly decode its 10-bit address.	1/19/00	3/13/00
IIC_12	5	The IIC Controller does not automatically awaken when a slave transfer is detected.	1/19/00	1/19/00
IIC_13	3	The IIC bus may hang in a multi-master environment during the 3rd address byte of a 10-bit read operation.	3/29/00	3/29/00
IIC_14	5	Incomplete transfer status incorrectly set after loss of bus arbitration.	3/29/00	3/29/00
IIC_16	3	Received data in slave mode may be lost.	5/10/02	5/10/02
PCI_18	3	Executing code from PCI address space may hang the CPU.	2/23/01	2/23/01
PCI_23	5	The PCI Bridge Controller fails to assert Sleep_Request to the Clock and Power Management (CPM) controller when in PCI synchronous mode.	9/19/00	9/19/00
PCI_24	1	The PCI Bridge Controller does not detect a parity error ($\overline{\text{PCIPErr}}$) asserted by a target during a write cycle.	2/27/01	1/10/03
PCI_25	1	Incorrect address or write data is driven on the PCI bus during a DAC transfer.	11/15/01	1/10/03
PCI_26	5	A parity error generated by outbound PCI reads cannot be masked by the PCIC0_CMD[PER] bit.	3/21/02	8/8/02
PCI_27	3	The internal PCI arbiter unfairly arbitrates when a master removes its request before the transfer.	9/24/02	10/23/02
PLB_4	3	Incorrect data may be stored in the PLB0_ACR during a DCR read after DCR write sequence.	8/21/01	8/21/01
SDRAM_10	4	Command leadoff settings (SDRAM0_TR[LDF]) of 3 or more MemClkOut cycles do not always generate the expected minimum delay.	9/5/01	9/5/01
SDRAM_12	3	SDRAM controller may issue unexpected mode register set command.	11/1/02	5/7/03
UART_1	3	Reading the UARTx_IIR may cause an interrupt to be lost or incorrect status to be read.	5/16/00	5/16/00

Design Note Summary

Table 2. Design Note Summary

Item	Description	Date First Docu-mented	Date Last Updated
1	When the processor is not powered (i.e., Vdd and OVdd = 0), additional precau-tions are needed if the IIC SDA and SCL are powered.	12/10/99	12/10/99
2	The SDRAM controller specification does not support mixing ECC and non-ECC memory banks.	12/10/99	12/10/99
4	Additional steps must be taken to awaken the UIC.	1/12/00	1/12/00
5	PLL Tune bits should be modified for Spread Spectrum clocking.	1/18/00	1/18/00

CHIP_11 When the IIC controller is operating as a slave with 10-bit addressing, the IIC controller may incorrectly detect Slave Write Needs Service.

Category: 3

Overview:

When the Slave Data Buffer (IIC0_SDBUFF) is full, i.e. contains 4 bytes of data and the Master does a 10-bit write operation, the slave (IIC controller) after receiving the 1st address byte incorrectly sets the Slave Write Needs Service bit (IIC0_XTCNTLSS[SWS]). The IIC controller should wait until after it receives the second address byte before setting the IIC0_XTCNTLSS[SWS]. Both address bytes are needed for 10-bit addressing.

Two types of failures are possible if the slave sets the IIC0_XTCNTLSS[SWS] in response to a 10-bit address when the IIC0_SDBUFF is full:

1. If the slave (IIC controller) is the device being addressed, it prematurely sets IIC0_XTCNTLSS[SWS] while it is still decoding the address.
2. If the slave (IIC controller) is the device not being addressed, it incorrectly sets IIC0_XTCNTLSS[SWS]. Note this case is possible when the IIC0_SDBUFF is filled with 4 bytes of data in anticipation of a slave read operation.

Note: Setting the Slave Write Needs Service bit (IIC0_XTCNTLSS[SWS]) will generate an IIC interrupt if the Enable IRQ on Slave Write Needs Service bit (IIC0_INTRMSK[EIWS]=1), Enable Interrupt bit (IIC0_MDCNTL[EINT]=1) and IIC Interrupt Enable bit (UIC0_ER[ITCIE]=1) are enabled.

Impact:

The IIC controller can hang the IIC bus when 10-bit addressing is used.

Work-around:

Use one of the following two work-arounds:

1. Prevent the Slave Data Buffer (IIC0_SDBUFF) from being full by ensuring it contains less than 4 bytes of data.
2. Do not use 10-bit addressing on the IIC bus if the PPC405xx/NPe405x IIC controller is used as a slave.

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

Category: 5

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: 5

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_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.

Work-around:

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

CPU_190 An incorrect exception vector offset is generated when an alignment exception and Data TLB Miss exception occur at the same time.

Category: 3

Overview:

If an alignment exception (vector offset of 0x0600) and a Data TLB Miss exception (vector offset of 0x1100) occur at the same time, the interrupt vector offsets of the two exceptions are logically OR'd together to produce a vector offset of 0x1700. An unaligned access by a lwarx, stwcx. or dcread instruction will cause an alignment exception and can also generate a Data TLB Miss exception. For 405 core+ASICs that include an APU attached device, there are four additional events that apply to this erratum: an unaligned APU load, an unaligned APU store, a trap on an APU Big Endian load/store or a trap on an APU Little Endian load/store.

Note:

1. The dcbz instruction does not apply to this erratum. This instruction can cause an alignment exception but it can not generate both an alignment exception and a Data TLB Miss exception.
2. An unaligned access using lwarx, stwcx. or dcread instruction is considered a programming error.

Impact:

An incorrect exception vector address is generated.

Work-around:

1. None needed if data relocation is disabled (MSR[DR] = 0).
2. At the exception vector offset 0x1700 place an unconditional branch to vector offset 0x1100 (the Data TLB Miss vector).

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:

The table below 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 3. 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_200 Multiply accumulate (MAC) instructions having the same target register (RT) as an immediately preceding multiply accumulate instruction or multiply instruction may yield incorrect results.

Category: 3

Overview:

This erratum affects multiply accumulate (MAC) instructions having the same target register (RT) as an immediately preceding multiply accumulate instruction or multiply instruction. The result stored in RT may not be correct.

In the examples below, target register (r9) of the MAC instruction at address A+4 may receive an incorrect result.

MAC instruction preceded by a MAC:

```
address A      maclhw  r9,r11,r12
address A+4    machhwu r9,r8,r13
address A+8    add     r3,r6,r4
address A+12   machhwu r11,r1,r2
```

MAC instruction preceded by a multiply:

```
address A      mulhw   r9,r11,r12
address A+4    machhwu r9,r8,r13
address A+8    add     r3,r6,r4
address A+12   machhwu r11,r1,r2
```

Note: This erratum affects all multiply accumulate instructions and multiply instructions.

Impact:

MAC instructions having the same target register (RT) as an immediately preceding MAC instruction or multiply instruction cannot be reliably executed.

Work-around:

Insert an instruction between MAC instruction pairs and multiply/MAC pairs if both instructions use the same target register. By rearranging the example code, the two instructions targeting r9 can be separated in one of two ways.

Method 1 separates the instruction pair with the add instruction.

MAC instruction preceded by a MAC:

```
address A      maclhw  r9,r11,r12
address A+4    add     r3,r6,r4
address A+8    machhwu r9,r8,r13
address A+12   machhwu r11,r1,r2
```

MAC instruction preceded by a multiply:

```
address A      mulhw   r9,r11,r12
address A+4    add     r3,r6,r4
address A+8    machhwu r9,r8,r13
address A+12   machhwu r11,r1,r2
```

Method 2 separates the instructions with the MAC instruction that targets r11.

MAC instruction preceded by a MAC:

```
address A      maclhw  r9,r11,r12
address A+4    machhwu r11,r1,r2
address A+8    machhwu r9,r8,r13
address A+12   add     r3,r6,r4
```

MAC instruction preceded by a multiply:

```
address A      mulhw   r9,r11,r12
address A+4    machhwu r11,r1,r2
address A+8    machhwu r9,r8,r13
address A+12   add     r3,r6,r4
```

CPU_201 Switching between virtual and real address modes may result in the execution of instructions fetched from the wrong real address.

Category: 3

Overview:

The PPC405 processor pre-fetches instructions ahead of the one which is currently being executed. It is the address of these pre-fetched instructions which may cause the problem, rather than the address of the instruction being executed. Commonly, the addresses being fetched are those directly following the instruction being executed, although if there is a branch instruction the processor may fetch instructions from the target of the branch. Most of the time, the distinction between the instructions being pre-fetched and the instruction being executed does not matter for this erratum description or workarounds, although it is mentioned where relevant.

This problem can be triggered in scenarios A-D, detailed below. They each share the common property that there is a change from virtual to real instruction address mode, or vice versa, typically by means of an interrupt or by an rfi or rfci instruction (MSR[IR] is changed). In each case, the virtual address of the virtual-mode instruction is similar to the real address of the real mode instruction. Similar to means that bits 0:26 of each instruction's effective address are identical, that is, they are within the range of a cache line, although the target code does not have to be cacheable.

A. Taking an Interrupt: When switching from a virtual address space to a real address space by means of an interrupt, and the virtual address of the interrupted instruction is similar to the real address of the first instruction of the interrupt handler, instructions from the virtual address space may be executed instead of the interrupt handler instructions in real address space.
B. As an example, consider an application executing code from virtual addresses 0x500-0x51C, which map to real addresses 0x42500-0x4251C, when an external interrupt occurs. If the EVPR is set to 0x0000, the processor should switch to real mode (MSR[IR]=0) and start executing interrupt handler instructions from real address 0x500. When this erratum occurs, the processor does switch to real mode, but erroneously executes instructions from virtual address 0x500, real address 0x42500.
C. Returning from an Interrupt: When switching from a real address space to a virtual address space by means of an rfi or rfci instruction, and the real address of the rfi/rfci instruction is similar to the virtual address of the instruction that is the target of the rfi/rfci, instructions from the real address space may be executed instead of instructions from the virtual address space.
D. Using rfi/rfci to Switch from Virtual to Real: When switching from a virtual address space to a real address space by means of an rfi or rfci instruction, and the virtual address of the rfi/rfci instruction is similar to the real address of the instruction that is the target of the rfi/rfci, instructions from the virtual address space may be executed instead of instructions from the real address space. This is a non-standard use of rfi/rfci, and will only occur in special circumstances (not normal interrupt handlers).
E. Using mtmsr to Transition from Virtual to Real or vice versa: This is usually only done in V=R (virtual address equals real address) space, which does not trigger this erratum. This case is mentioned for completeness and is not discussed further here.

Impact:

When the problem occurs, instead of executing instructions out of the new address space, the processor executes instructions from the old address space.

CPU_201 Continued

Work-around:

The problem does not occur and no work-around is required for each of the following cases:
- Instruction relocation is not used: MSR[IR]=0.
- Instruction relocation is used (MSR[IR]=1) and all instruction translations are V=R.
- Instruction relocation is used and there are non-V=R instruction translations, but there are no virtual addresses which match the addresses of any instructions executed in real mode.

Otherwise, perform all of the following:

1. To avoid condition A) above, ensure that, except for V=R entries, there is no TLB entry which has a cacheable virtual instruction address space where the virtual addresses are the same as the real addresses used by the exception vectors ("the EVPR page"). In other words, no TLB entry exists where TLBHI[EPN[0:15]] = EVPR[0:15] and TLBLO[EX]=1 and TLBLO[I]=0. In practice, this may be a subset of work-around 2).
2. To avoid condition B) ensure that no rfi/rfci is executed in cached real mode where the real address of the rfi/rfci is similar to the virtual address of the rfi/rfci target (in SRR0/2), and instruction relocation will be turned on (SRR1/3[IR]=1), except for V=R. This can be accomplished by implementing one of the following:
   a. Do not create TLB entries (except V=R) where the virtual address is the same as the real address of any code which will execute in real mode and TLBLO[EX]=1.
      This restriction may not be hard to accomplish in many systems, where code resides in real memory which is typically located at a low address (starting at 0x0), and virtual addresses typically have some of their high bits set, so no virtual addresses will be equal to any real addresses. In other cases, however, the operating system must ensure that there are no instruction virtual addresses which are the same as any real address space containing code which may execute an rfi or rfci instruction.
   b. Ensure that all code that executes in real mode is non-cacheable by setting ICCR=0x00000000. This may have a significant performance impact on real-mode code, and thus on overall system performance.
   c. Ensure that all rfi or rfci instructions which execute in real mode are non-cacheable. This may be achieved by saving the value of the ICCR in a known area of memory, and setting ICCR=0x00000000 immediately before performing an rfi or rfci. At the start of all interrupt handlers the saved ICCR value can be loaded from memory and restored to the ICCR before continuing. Only a small portion of real-mode code executes with instruction caching off which should have little performance impact. However, this requires modification to the source code of all interrupt handlers.

CPU_201 Continued

d. Only execute rfi or rfci instructions from the 64K byte area defined by the EVPR ("the EVPR page"). Work-around 1) above ensures that there is no overlap between virtual addresses and the real address of the EVPR page. To implement this work-around, it is necessary to place both an rfi and an rfci instruction at free memory locations in the EVPR page. The rfi and rfci instructions in the EVPR page should be aligned on a cache-line boundary and followed by seven nop instructions, to ensure that when the processor pre-fetches instructions it does not find a branch instruction, which may cause it to pre-fetch from a page outside the EVPR. Then, locate all other rfi/rfci instructions and replace them with a branch to the rfi or rfci instruction in the EVPR page. This is made easier if the value of the EVPR is known before run-time. This work-around requires that either:
   - i. All rfi/rfci instructions which will be replaced are within the range of a branch instruction (32MB) from the target in the EVPR page, and a branch relative instruction is used to replace the rfi/rfci; or
   - ii. The EVPR page is placed in a location where it can be reached by a 'branch absolute' instruction, that is EVPR[0:6]=0b0000000 or 0b1111111, and a branch absolute instruction is used to replace the rfi/rfci.
   - iii. The contents of SRR0/1 are saved, and SRR0 is set to the address of the target in the EVPR page, SRR1 is set to the current contents of the MSR, and an rfi is executed to perform a 32-bit "branch" to the EVPR page. This avoids the address restrictions in i) and ii), but means that additional code must be inserted both before the original rfi (to save SRR0/1) and at the target in the EVPR page, to restore the saved SRR0/1 values before executing the rfi.

3. To avoid condition C) ensure that no rfi/rfci is executed in virtual mode, except for V=R, where the virtual address of the rfi/rfci is similar to the real address of the rfi/rfci target (in SRR0/2), and instruction relocation will be turned off (SRR1/3[IR]=0). In practice this may be a subset of work-around 2).

CPU_202 A load string or load multiple instruction that accesses the Data Side OCM may provide incorrect data.

Category: 5

Overview:

When all of the following five conditions are true, a load string or load multiple instruction (lswi, lswx, lmw) may write incorrect data to the general purpose register (GPR) file and corrupt data accessed by instructions that follow:

1. Data translation is enabled (MSR[DR]=1).
2. A load string or load multiple instruction reads data from the data side OCM.
3. The effective address accessed by the load string or load multiple instruction is not word aligned.
4. The load string or load multiple instruction causes a page boundary crossing.
5. The page crossing results in a data-side TLB miss and a UTLB hit.

Impact:

Reading the data side OCM using load string or load multiple instructions (lswi, lswx, lmw) may corrupt the target GPR register(s) and/or provide incorrect data to subsequent instructions that reference the targeted register(s).

Work-around:

No work-around is required if data translation is disabled (MSR[DR]=0). When data translation is active (MSR[DR]=1) do one of the following:

1. Place the entire Data-Side OCM address space in a single page to prevent page boundary crossings.
2. Ensure that the effective address of all load string and load multiple instructions is word aligned.

CPU_206 Unaligned store and store string operations to On-Chip Memory (OCM) may write incorrect data to the OCM.

Category: 5

Overview:

Unaligned store and store string operations to the OCM address space may write incorrect data to the OCM when the data cache is busy. As an example, data corruption may occur if the data cache is handling two outstanding cache line fills while a store or store string operation to the OCM address space is executed.

An unaligned store to OCM is split into two separate store operations. The second part of the unaligned store may write incorrect data to the OCM.

A store string is split into several separate store operations, each writing data to individual words in memory. Store string operations to OCM address space having a starting destination address that is not word aligned may write incorrect data to the OCM.

Impact:

Unaligned store and store string operations to OCM may write incorrect data to the OCM.

Work-around:

Perform one of the following:

1. Ensure a and b never occur:
   a. Unaligned store operations to OCM address space
   b. Store string operations having a starting destination address that is not word aligned in OCM address space
2. Perform a sync instruction before all unaligned store and store sting operations that write OCM.

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: 1

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_209 A lwarx, stwcx. instruction sequence may cause a branch to link register (bclr) or branch to count register (bcctr) instruction to execute incorrectly.

Category: 2

Overview:

The branch conditions for a branch to link register (bclr) or branch to count register (bcctr) instruction may be incorrectly evaluated if executed following one of the lwarx, stwcx. instruction sequences given in Code Sequence 1, 2, 3 and 4.

Code Sequence 1:

```
  address A       lwarx
  address A+4     mtlr
  address A+8     stwcx.
  address A+12    bclr
```

Code Sequence 2:

```
  address A       lwarx
  address A+4     mtctr
  address A+8     stwcx.
  address A+12    bcctr
```

Code Sequence 3:

```
  address A       lwarx
  address A+4     Any form of a branch
                  and link instruction
                  that is taken and has
                  target T
  address A+8     Any Instruction
  .
  address T       stwcx.
  address T+4     bclr
```

Code Sequence 4:

```
  address A       lwarx
  address A+4     Any form of a branch
                  and decrement instruc-
                  tion that is taken and
                  has target T
  address A+8     Any Instruction
  .
  address T       stwcx.
  address T+4     bcctr
```

Impact:

Branch to link register (bclr) and branch to count register (bcctr) instructions may not branch in the correct direction if following a lwarx, stwcx. instruction pair in Code Sequence 1, 2, 3 or 4

Note: lwarx and stwcx instructions are not compiler generated.

Work-around:

Avoid executing Code Sequence 1, 2, 3 or 4 by performing all of the following work-arounds:

1. Perform a or b.
   a. Place two or more instructions between lwarx and stwcx. instructions.
   b. Place no instructions between lwarx and stwcx. instructions.
2. If lwarx and stwcx. are separated by one instruction, make sure this instruction is not a mtctr or mtlr instruction.
3. Avoid placing a branch and link or a branch and decrement instruction immediately after a lwarx instruction.

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

Category: 1

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 execute (EXE) stage. The first piece moves to writeback (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 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 the 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_213 Incorrect data may be flushed from the data cache.

Category: 3

Overview:

The CoreConnect™ architecture based PPC405GP consists of masters and slaves interconnected via the Processor Local Bus (PLB). The PPC405 CPU features separate PLB master interfaces for the instruction cache unit (ICU) and the data cache unit (DCU). Any CPU load, store, or data cache control instruction may result in the CPU issuing one or more transactions through its DCU PLB interface. Whether or not a given data operation results in a PLB transaction depends on factors including the cacheability of the referenced address and, in the case of a cacheable access, the state of the data cache. Once the DCU initiates a PLB read or write request the transaction proceeds with timing that is dependent on both the PLB slave responsible for servicing the given address and PLB activity that may be occurring between other masters and slaves. If the PPC405 DCU interface performs the following specific sequence of PLB operations incorrect data may be flushed from the data cache:

1. Data cache flush of line1 (line write): This flush is a pre-conditioning operation and is not tightly coupled with the sequence of operations defined by steps 2-5. It is only required that this flush be the last data side write to the PLB prior to steps 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 line2 data cache fill. It may or may not cause data to be replaced or flushed.
4. Non-cacheable or write-through write of word4 (word, halfword or byte write): The timing of the request on the PLB for this write must occur within a narrow window immediately after the CPU receives the last data item from the 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 written to memory is incorrect.

For this erratum to occur the PLB slave servicing the data cache fill of line3 must return the cache line data in four consecutive PLB cycles. Since the PLB data buses are 64-bits wide and all of the slaves in the PPC405GP feature 32-bit external buses, line fill data is in most cases returned every other PLB cycle. For data to be returned in consecutive PLB cycles the slave must have accepted the line fill request from the CPU as a pipelined transfer, and read and buffered the line contents from external memory while at the same time another PLB transaction between a different master and slave maintained ownership of the on-chip PLB read data bus.

An instruction sequence that can cause the above series of data side PLB operations is as follows:

1. A load or store instruction that misses in the data cache and causes a fill to a line with the Least Recently Used (LRU) tag pointing to a valid dirty line, or a dcbf to a line with valid dirty data: This instruction causes the flush of line1.
2. A load or store instruction that misses in the data cache and causes a line fill to a line with its LRU pointing to a valid dirty line: This instruction causes both the fill of line2 and a subsequent flush of the dirty contents of line2 that were replaced by the fill.
3. A store instruction that misses in the data cache and causes a line fill: This instruction causes the fill of line3 and may or may not result in a line flush.
4. A store instruction to a memory address designated as either non-cacheable or write-through: This instruction causes the non-cacheable write of word4.
5. A load instruction requiring data from either or both of the last two 64-bit doublewords returned for the cache line fill of line3. This load does not cause a PLB operation, but is required to create the conditions necessary to cause this erratum.

CPU_213 Continued

These instructions need not execute with a particular timing relative to each other. Rather, they may be separated by other unrelated instructions and operations including branches, interrupts, instruction cache misses, and additional loads or stores that hit in the data cache.

Impact:

Incorrect data may be flushed (written) to memory from the data cache.

Work-around:

Perform one of the following work-arounds:

- Set CCR0 reserved bits 1 and 3. When these bits are set and there is a line fill pending or in progress for a data cache miss the data cache is prevented from servicing other CPU load or store requests. Because this work-around blocks data cache accesses during line fills it may affect performance.
- Mark all data memory as write-through using the DCWR register or W storage attribute of each TLB entry.

DCP_6 Executing instructions from a memory region configured as uncompressed after executing instructions from a memory region configured as compressed can cause the Decompression Controller (DCP) to generate a machine check exception or hang the PLB.

Category: 3

Overview:

The decompression controller (DCP) can hang the PLB or generate an incorrect error condition if the following sequence of events occurs:

1. The CPU fetches an instruction from target instruction address 1 (TIA1). TIA1 is located in a memory region that is configured as compressed and the fetch does not hit in the instruction cache.
2. The DCP reads and begins decompressing the block of instructions from external memory that contains the instruction at TIA1.
3. The CPU fetches an instruction from target instruction address 2 (TIA2). TIA2 is located in a memory region that is configured as uncompressed. This fetch misses in the I-cache and step 2 is still in progress.
4. The instruction execution flow is such that the CPU aborts the request made in step 3, but step 2 is still in progress.
5. The CPU core fetches an instruction from target instruction address 3 (TIA3). TIA3 is located in a memory region that is configured as compressed. This fetch misses in the I-cache and step 2 is still in progress.
6. The DCP acknowledges the request made in step 5 at the same time the read from step 2 completes. One of two events occur:
   a. If no other transfer is present on the PLB in the next 16 PLB clock cycles, the DCP causes an instruction machine check. This results in the DCP0_ESR register indicating that an error or time-out occurred during the index table entry (ITE) or compressed block fetch.
   b. If another transfer is present on the PLB in the next 16 PLB clock cycle, the PLB hangs.

Note, a memory region is configured as compressed if either:
- MSR[IR] = 0 and the corresponding bit in the SU0R register is set to 1.
- MSR[IR] = 1 and the U0 bit in the corresponding TLB entry is set to 1.

A memory region is configured as uncompressed if either:
- MSR[IR] = 0 and the corresponding bit in the SU0R register is set to 0.
- MSR[IR] = 1 and the U0 bit in the corresponding TLB entry is set to 0.

Impact:

Once the decompression controller is enabled and the CPU begins to execute instructions from a compressed memory region, instructions from an uncompressed region cannot be reliably executed. If only instructions from compressed memory regions are executed once the decompression controller is enabled, there is no impact.

Work-around:

Do not execute instructions from an uncompressed memory region once instructions have been executed from a compressed memory region.

DCP_9 Compressed instruction blocks that cross a 1KB boundary in external memory can cause the Decompression Controller (DCP) to return incorrect instructions to the 405 processor core.

Category: 3

Overview:

The decompression controller uses burst read transfers to read compressed instruction blocks from external memory into its internal 64 byte buffer. If a compressed instruction block crosses a 1KB boundary, the SDRAM controller or External Bus Controller (EBC) terminates the burst at the 1KB boundary. Under certain conditions, when the DCP restarts the burst, it will return invalid instructions to the 405 processor core. Incorrect program results, or a program exception (ESR[PIL]=1) can occur.

Impact:

Compressed code blocks cannot cross a 1KB boundary in external memory.

Work-around:

Obtain CodePack Code Compression Utility version 2.54 (or later). The utility has an optional parameter that will take into account the specified boundary when placing compressed instruction blocks into the target address space. Slightly lower compression results are experienced when using this feature because small unused holes can be left prior to the specified boundary.

In the compression utility template file, specifying the following parameter causes the utility to avoid 1KB boundaries when placing compressed instruction blocks into the target address space:

```
G 0x00000400
```

EBC_5 Peripheral bank 4 of the external bus controller cannot be programmed as a "Write Only" bank.

Category: 4

Overview:

Setting the Bank Usage field in the Peripheral Bank 4 Configuration Register (EBC0_PB4CR[BU]) to b'10 does not configure bank 4 as a "Write Only" bank.

Impact:

Read access to peripheral bank 4 cannot be removed without disabling both read and write access.

Work-around:

Use peripheral bank 0, 1, 2, 3, 5, 6 or 7 for write-only peripheral devices.

EBC_20 No parity generated during a DMA memory-to-peripheral write.

Category: 3

Overview:

Setting the parity enable bit in the DMA controller register does not produce the expected results during DMA memory-to-peripheral writes. The EBC drives all 0's on PerPar0:3 instead of odd parity.

Impact:

DMA memory-to-peripheral mode cannot be used if parity is required. Work-around requires the use of an EBC bank.

Work-around:

The recommended work-around is to use hardware device paced memory-to-memory DMA transfers. This type of transfer is still initiated using DMAReq, but as stated in the DMA section of the user's manual, a chip select is used in place of the DMAAck. An unused EBC bank is required since a chip select is needed. The EBC bank must be properly configured for accessing memory to perform memory-to-memory DMA transfers and the bank's parity enable bit (EBC0_BnAP[PEN]) for must be set. The EBC address pins can be ignored. The only requirement when the DMA is programmed is the source/destination address registers match the address range configured in the EBC bank.

EMAC_5 The EMAC constantly transmits a preamble pattern on the MII interface.

Category: 4

Overview:

The minimum value specified for the TLR (EMAC0_TMR1[TLR]) is wrong. The correct equation is:

min(TLR) = (MAL Burst Limit / 2) + 1

Since the MAL Burst Limit is 16, the correct minimum value for TLR is 9.

Impact:

An incorrect TLR setting (below the minimum) can cause the EMAC to constantly transmit a preamble pattern on the MII interface which blocks anything else from being transmit. This error does not generate an error indication such as an interrupt or a status bit.

Work-around:

Set the TLR value to 9 (EMAC0_TMR1[TLR]=9).

EMAC_7 Signal Quality Error (SQE) occasionally reported incorrectly during SQE test.

Category: 5

Overview:

Occasionally while running the Signal Quality Error (SQE) test, errors may be reported even though none have occurred. The SQE test is only used when operating an MII interface at 10Mbs in half-duplex mode, and is used to verify the Collision Detect logic in the PHY is operating properly.

Impact:

This is a minor problem and has no affect on data transfers with the EMAC.

Work-around:

Disable the SQE test in the EMAC.

EMAC_8 Soft Reset may not reset all logic in the EMAC

Category: 5

Overview:

Soft Reset (EMAC0_MR0[SRST]) may not reset all logic in the EMAC.

Impact:

Once Soft Reset has completed, and the receive channel is re-enabled, an extra data double word may be added at the beginning of the first received packet without any bad status indication on this packet when forwarded to the MAL.

Work-around:

Before Soft Resetting the EMAC, first disable the receive channel (EMAC0_MR0[RXE]=0), then wait for the RX MAC Idle bit to be set (EMAC0_MR0[RXI]=1). Once idle, enable the Soft Reset bit (EMAC0_MR0[SRST]=1).

EMAC_9 Integrated Flow Control Mechanism may not function correctly

Category: 4

Overview:

The integrated flow control mechanism is used only in Full-duplex mode in order to avoid receive FIFO overrun. Two watermarks are set on the receive FIFO. When the high watermark is crossed upwards, an urgent pause packet is internally assembled and transmitted causing the other end's transmitter to stop. When the low watermark is crossed downwards, a zero pause packet is internally assembled and transmitted, thus causing the other end to resume transmission.

There are three problems with the integrated flow control mechanism:

1. If a request is issued, and in the meanwhile the opposite watermark is crossed, the request initiator could take the request back. A request is taken back by de-asserting the request signal without waiting for acknowledge. This condition can lead to a deadlock in the handshake protocol, since the request initiator cannot know if the transmitter received the request (or if it is going to acknowledge the request).
2. The integrated flow control mechanism cannot issue two consecutive urgent pause packet requests.
3. When a request for a zero pause packet is issued, and until an acknowledge is accepted, the receive FIFO can theoretically reach the high watermark. The issue pause mechanism is not aware of this high watermark crossing, and thus does not send an urgent pause packet.

Impact:

The impact of the three problems list above is as follows:

1. Total failure of the issue pause mechanism preventing future pause control frames from being issued as a response to watermark crossings.
2. FIFO overflow - A user may want to set the Pause Timer Register to a small value that causes only a short stop, which would not necessarily cause the receive FIFO to fall below the low watermark.
3. FIFO overflow - The FIFO overflow occurs because the high watermark crossing does not cause an immediate request for non-zero pause packet transmission.

Work-around:

No work-around is available. Integrated Flow Control should be disabled (EMAC0_MR1[EIFC]=0).

EMAC_10 Octet Counter Registers may not record an accurate count

Category: 4

Overview:

The Transmitted Octet Count Register (EMAC0_OCTX) and Received Octet Count Register (EMAC0_OCRX) contain the total number of bytes transmitted and received respectively on the media (including bad packets). After a packet is transmitted or received, the transmitter or receiver sends the number of frame bytes to the count register logic, where it is added to the proper counter register. Since the transmit and receive logic are on a different clock domain from the register logic, synchronization across the clock boundary is necessary. In the process of crossing the clock boundary, the count from the transmitter or receiver may be incorrectly sampled, therefore, causing the counter registers to record inaccurate values.

Impact:

The EMAC0_OCTX and EMAC0_OCRX registers may not contain an accurate count.

Work-around:

No work-around is available.

IIC_6 Slave Transfer Count is not cleared correctly.

Category: 5

Overview:

The Slave Transfer Count (IIC0_XFRCNT) is not cleared when the Slave Write Complete (IIC0_XTCNTLSS[SWC]) or Slave Read Complete (IIC0_XTCNTLSS[SRC]) status bits are cleared. It is cleared when the Slave Write Needs Service (IIC0_XTCNTLSS[SWS]) or the Slave Read Needs Service (IIC0_XTCNTLSS[SRS]) bits in the Extended Control and Slave Status register (IIC0_XTCNTLSS) are cleared. In transfers of 3 bytes or fewer the "Needs Service" bits are not set, only the "Complete" bits are set. Thus, for smaller transfers, the Slave Transfer Count (IIC0_XFRCNT) will not be cleared to zero when the completed transfer of the slave has been handled (i.e. when the slave transfer complete status is cleared the transfer count is not cleared).

Impact:

For smaller transfers, the Slave Transfer Count (IIC0_XFRCNT) will not be cleared to zero when the completed transfer of the slave has been handled (i.e. when the slave transfer complete status is cleared the transfer count is not cleared).

Work-around:

Manually write 0x0 to bits 7:4 of the Transfer Count register (IIC0_XFRCNT) just prior to clearing the "Complete" status bit, (IIC0_XTCNTLSS[SWC] or IIC0_XTCNTLSS[SRC]).

IIC_7 The IIC controller is temporarily placed in an invalid state.

Category: 5

Overview:

Loss of arbitration in the 3rd address byte during a 10-bit read operation temporarily places the IIC Controller into an invalid state. In a 10-bit read, the transaction proceeds as follows:

Start, Address Byte 1 with bit 8 = write operation, Address Byte 2, Repeated Start, Address Byte 1 with bit 8 = read

Where Address Byte 1 = any value 0xF0 through 0xF7 and the 2nd transmission of Address Byte 1 uses the exact same value as the 1st.

Thus, the malfunction is during the 2nd transmission of address byte 1 (the 3rd address byte in the transaction described above). The IIC Controller resumes normal operation once the current transfer ends or once a new one starts.

Impact:

The IIC controller is temporarily placed in an invalid state.

Work-around:

None needed.

IIC_9 Loss of Bus Arbitration while sending the address byte(s) causes an auto-retry of the transfer.

Category: 3

Overview:

In the situation described below, the IIC Controller will automatically retry a transfer. Instead of performing an automatic retry, the IIC Controller should abort the transfer.

In a system having two IIC masters, IIC Controller (M1) and Master 2 (M2), and one IIC slave, S1. It is assumed in this example that M1 and M2 start at exactly the same time and write the same 3 address bytes to S1. (After sending byte 3 neither master issues a STOP because both masters want to own the IIC bus.) M1 and M2 then issue repeated STARTS at exactly the same time. The transfer being arbitrated by M1 is a read transfer from S1 and the transfer being arbitrated by M2 is a write transfer. If M1 loses arbitration and M2 wins, M2 owns the bus. M1 waits until M2 issues a STOP and then *AUTOMATICALLY* retries the failed read. While M2 owns the bus, it may send several bytes of data to S1. A problem occurs when M1 reads the 3 bytes of set-up information sent prior to losing arbitration. This data may have been overwritten. If overwritten, M1 will unknowingly read incorrect data.

Impact:

In a multiple master configuration, an IIC Controller may read corrupted data from a slave if there is a loss of arbitration as described in the overview.

Work-around:

The following 5 steps describe the work-around:

1. Find an illegal or unused address that is as close to 0x06 or 0x07. If possible use either 0x06 or 0x07 as these are reserved addresses.
2. Precede any set of writes and reads that must be done to a shared slave using the Hold Bus Ownership bit in the Control register (i.e. any set of operations that must be done to a slave with exclusive ownership of the bus) with a write of two bytes of zero data, 0x00, to the address chosen in step 1.
3. Wait for an interrupt from the IIC core. The error status should be set along with the incomplete and transfer aborted bits in the extended status register.
4. The IIC core will now own the bus as indicated by:
   a. Halt/Stop is not set in the status register
   b. The extended status register shows that the core is in the Master Transfer state (bits 1:3)
5. Perform the desired operations using the Hold Bus Ownership bit in the control register.

IIC_10 A loss of Arbitration while sending the 1st or 2nd address bit can cause the IIC Controller to malfunction.

Category: 5

Overview:

If the IIC Controller of the PPC405GP loses bus arbitration while sending the 1st or 2nd address bit, it does not properly transition from the master state to the slave state. Therefore, the IIC does not acknowledge a transfer if it is addressed as a slave by another master. This non-acknowledgement will cause the IIC master to either issue a STOP or re-do the failed transfer. In either case, the IIC Controller recovers and correctly responds to the master.

Note: If the IIC Controller is not addressed as a slave, this problem does not occur.

Impact:

There is a slight loss of bus performance.

Work-around:

The master that wins the IIC bus arbitration should re-do the transfer if the transfer fails.

IIC_11 The IIC Controller when operating as a slave may not correctly decode its 10-bit address.

Category: 5

Overview:

The IIC Controller when operating as a slave in 10-bit address mode may not correctly decode its 10-bit address. This problem occurs when an IIC master addresses the IIC Controller as a slave.

> **Note:** The IIC Controller when operating as a slave correctly decodes its address in 7-bit address mode.

Impact:

Some IIC bus addresses cannot be used.

Work-around:

Avoid using the following values in the "Hi Slave Address" (IIC0HSADR) and "Lo Slave Address" (IIC0LSADR) registers:

Note that X="don't care" and V="a particular value"

1. Hi Address = "1111_0xxx" = Lo Address, the Lo Slave Address should not be set to a 10-bit address-like code.

   Example of good and bad settings:
   Bad setting: Hi Adr= 0xF0 and Lo Adr=0xF0
   Good setting: Hi Adr= 0xF0 and Lo Adr= 0x70

2. If any other Lo Slave Address is "VVVV_VVVV0" then this IIC Lo Slave Address cannot be "VVVV_VVV1", it must be different by more than just the least significant bit.

   For example: If another IIC bus slave has a Lo Address of 0x70, then the core cannot have a Lo Address of 0x71. A good setting for the core would be 0x73 or 0x72.

3. If any other Lo Slave Address is "VVVV_VVVV1" then this IIC Lo Slave address cannot be "VVVV_VVV0", it must be different by more than just the least significant bit.

   For example: If another IIC bus slave has a Lo Address of 0x71, then the core cannot have a Lo Address of 0x70. A good setting for the core would be 0x72 or 0x73.

IIC_12 The IIC Controller does not automatically awaken when a slave transfer is detected.

Category: 5

Overview:

IIC slave transfers to the PPC405GP will not be acknowledged if the IIC Controller is asleep.

Impact:

The IIC Controller does not automatically awaken when a slave transfer is detected.

Work-around:

Perform the following two steps:

1. Awaken the IIC Controller by clearing the IIC bit field of the CPM Enable and CPM Force (CPC0_ER[IIC]=0 and CPC0_FR[IIC]=0) registers.
2. Have the IIC master re-send the non-acknowledged transfer.

IIC_13 The IIC bus may hang in a multi-master environment during the 3rd address byte of a 10-bit read operation.

Category: 3

Overview:

The IIC controller may malfunction in a multi-master environment under the following conditions:

1. The IIC controller and another IIC master believe they own the bus.
2. The IIC controller and another IIC master initiate a 10-bit read operation at the same time.
3. The IIC controller and another IIC master access the same slave address.
4. The IIC controller operates at lower frequency than the other master.
5. The IIC master uses the minimum permissible SDA hold time in a repeated start operation.
6. The IIC controller wins arbitration in the address that follows the repeated start signal (or at least does not loose arbitration).

When these conditions occur the IIC controller does not detect the fall of SCL and is therefore out of sync with the address bits sent on the IIC bus.

Impact:

The IIC controller either hangs the IIC bus or does a not-Acknowledge to end the transfer.

Work-around:

Use the work-around described for IIC_9.

IIC_14 Incomplete transfer status incorrectly set after loss of bus arbitration.

Category: 5

Overview:

When the IIC controller does a chained write of one byte followed by another write (chained or non-chained), and the core loses IIC bus arbitration within the 1st byte of the second write, the Incomplete Transfer bit in the Extended Status register is incorrectly set to logic one. Since arbitration was lost during the 1st byte, the byte is still in the master data buffer. The status therefore is incorrect.

Impact:

A false error condition recorded in the Extended Status Register.

Work-around:

Perform one of the following:

1. Use the work-around described for IIC_9.
2. Ignore the Incomplete Transfer bit when the Lost Arbitration bit is set and the Master Data Transfer Count = 0x0.

IIC_16 Received data in slave mode may be lost.

Category: 3

Overview:

Any data received when the slave data buffer is full (i.e. contains four bytes of data) is lost.

Impact:

Data received in slave mode can be lost.

Work-around:

Do not wait until the slave data buffer is full before reading it.

PCI_18 Executing code from PCI address space may hang the CPU.

Category: 3

Overview:

As with all high-performance processors, the PPC405 CPU features an instruction pipeline designed to provide the execution unit with a steady stream of instructions. To optimize throughput, the instruction fetcher attempts to keep the pipeline full by requesting instructions along the most probable path of code execution. Whenever an instruction fetch cannot be fulfilled by the instruction cache, either a 4-word or 8-word line fill request is forwarded via the on-chip Processor Local Bus (PLB) to either the SDRAM, PCI, or peripheral bus controller. In the case of instruction fetches from PCI address space, the PCI controller obtains and internally buffers the requested instructions from the PCI target before returning them to the CPU. If during this period of time an instruction such as a conditional branch or an interrupt alters the code flow such that the pending instruction fetch from PCI address space is no longer necessary, the CPU may abort the transfer.

Impact:

If the CPU aborts an instruction fetch from PCI address space and then subsequently performs a data read from PCI address space, the read may hang the PLB bus. This hang condition will persist until the CPU fetches another instruction from PCI address space.

Work-around:

If the CPU is not fetching instructions from PCI address space no work-around is required. Otherwise, ensure that the CPU is the only master in the PPC405GP accessing PCI address space and do all of the following:

1. Completely disable instruction caching. If operating in real mode, MSR[IR] = 0, set ICCR = 0. If running in virtual mode, MSR[IR] = 1, ensure that all TLB entries that allow execution, TLBLO[EX] = 1, also inhibit instruction caching: TLBLO[I] = 0.
2. Program CCR0[24] = 1. Setting this undocumented bit ensures that the CPU cannot continuously satisfy instruction requests from the instruction cache fill buffer.
3. Program PCIL0_PMMnHA = 0x0 for all PLB to PCI mappings containing executable code. This prevents the PCI Bridge from generating dual address cycles on the PCI bus.

Items 1 & 2 in this work-around ensure that any CPU data request from PCI address space will be followed by an instruction fetch, thus preventing the PCI Bridge from becoming stuck in a hang condition. Following an aborted instruction fetch, the PCI Bridge may fail to complete a delayed read. Therefore, the PCI target providing instructions to the PPC405GP may be unresponsive until its delayed read discard timer expires after 215 PCI clocks. Also, the target may detect this situation as an error condition. The system must be tolerant of these events.

Note: If the PCI target is another PPC405GP, its delayed read discard timer will expire, allowing forward progress, and no error will be detected.

Since executing directly from PCI address space is relatively slow, and because data in PCI address space cannot be cached, (the PCI Bridge does not support cache line write operations) it is highly recommended that systems boot from PCI and load their application into SDRAM memory. Boot code executing from PCI space should set CCR0[24] = 1 immediately after the branch from address 0xFFFFFFFC. Once the load process completes and execution is from SDRAM address space, enable instruction caching and clear CCR0[24].

PCI_23 The PCI Bridge Controller fails to assert Sleep_Request to the Clock and Power Management (CPM) controller when in PCI synchronous mode.

Category: 5

Overview:

When in synchronous mode, the PCI Bridge Controller fails to assert Sleep_Request to the CPM controller in response to Sleep_Init. Sleep_Init is generated by setting the PCI bit in the CPM Enable Register, CPC0_ER[PCI].

Impact:

The PCI Bridge Controller core fails to enter sleep mode.

Work-around:

Treat the PCI Bridge Controller as a "class 1" sleep device, and force it to sleep, rather then requesting it to sleep. To force the PCI Bridge Controller to sleep, first place all devices on the PCI bus to sleep and then place the PCI Bridge Controller to sleep by setting the PCI bit in the CPM Force Register, CPC0_FR[PCI].

PCI_24 The PCI Bridge Controller does not detect a parity error ($\overline{\text{PCIPErr}}$) asserted by a target during a write cycle.

Category: 1

Overview:

The PCI Bridge Controller fails to detect a parity error ($\overline{\text{PCIPErr}}$) asserted by a target during a write cycle under the following conditions:

1. The PCI Bridge is operating in Asynchronous Mode.
2. The PCI Bridge is running an outbound (PLB-to-PCI) write.
3. The PCI target asserts $\overline{\text{PCIPErr}}$ in response to detecting a data parity error.
4. The parity error is for the last beat of a burst or for a single beat.
5. The PCIClk frequency is within the following range: 0.7*SyncPCIClock ≤ PCIClk ≤ 1.1*SyncPCIClock

   PCIClk ranges affected by this erratum for typical clock configurations:

   - PLB Clk = 100MHz, SyncPCIClock = 50MHz, 35MHz ≤ PCIClk ≤ 51MHz
   - PLB Clk = 133MHz, SyncPCIClock = 66.7MHz, 46.7MHz ≤ PCIClk ≤ 66.7MHz
   - PLB Clk = 133MHz, SyncPCIClock = 44.4MHz, 31.1MHz ≤ PCIClk ≤ 46MHz

   Note: The SyncPCIClock is an internal clock derived from the PLB Clk and PCIClk is supplied by the PCI bus.

Impact:

A parity error may not be detected during a write cycle.

Work-around:

Select a PCIClk outside the failing frequency range such that PCIClk < 0.7*SyncPCIClock or PCIClk > 1.1*SyncPCIClock.

PCI_25 Incorrect address or write data is driven on the PCI bus during a DAC transfer.

Category: 1

Overview:

When the PCI Bridge Controller is the initiator of a DAC (Dual Address Cycle) transfer in Asynchronous Mode, it may drive incorrect address and/or write data values on the PCI bus. This error can occur if the internal SyncPCIClock and the externally supplied PCIClk have clock periods within ±2ns of one another. The following are PCIClk frequency ranges for typical clock configurations that are affected by this erratum:

- PLB Clk = 100MHz, Sync PCI Clk = 33.3MHz, 31MHz ≤ PCIClk ≤ 36MHz
- PLB Clk = 100MHz, Sync PCI Clk = 50MHz, 45MHz ≤ PCIClk ≤ 55MHz
- PLB Clk = 133MHz, Sync PCI Clk = 66.7MHz, 59MHz ≤ PCIClk ≤ 66.7MHz

Note:

(1) DAC cycles are used to address PCI targets in the address space above 4 Gigabytes.
(2) The SyncPCIClock is an internal clock derived from the PLB Clk and the PCIClk is supplied by the PCI bus.

Impact:

Address and data values driven on the PCI bus are not reliable during a DAC transfer.

Work-around:

Use one of the following clock configurations: PLB Clk/SyncPCIClock/PCIClk = 100MHz/100MHz/66MHz, 100MHz/50MHz/33MHz, 133MHz/44MHz/33MHz.

PCI_26 A parity error generated by an outbound PCI read cannot be masked by the PCIC0_CMD[PER] bit.

Category: 4

Overview:

The Parity Error Response bit (PCIC0_CMD[PER]) determines whether a PCI error is recorded by the PLB Bus Error Assertion Event status bit (PCIC0_ERRSTS[MEAE]) and reported to the PLB master. However, the PCIC0_CMD[PER] does not reliably mask a parity error generated by an outbound PCI read. (Outbound traffic is generated when the PCI Bridge is a PLB slave and a PCI Master.) The table below list all scenarios where PCIC0_CMD[PER] fails to correctly mask outbound parity errors.

Note: The PCIC0_CMD[PER] correctly masks the PCIC0_STATUS[DPE] for outbound reads.

Table 4. List all affected PCIC0_CMD[PER] and PCIC0_ERREN[MEAE] bit combinations

PCI data bus Parity Error	PCIC0_CMD[PER]	PCIC0_STATUS[DPE]	PCIC0_ERREN[MEAE]	PCIC0_ERRSTS[MEAE]	Error Reported to PLB Master
Outbound Read	0	0	0	May report an error regardless of the PCIC0_CMD[PER]	No error reported
Outbound Read	0	0	1	May report an error regardless of the PCIC0_CMD[PER]	May report an error
Outbound Read	1	1	1	May not report an error	May not report an error

Impact:

Outbound read parity errors are unreliably masked by the PCIC0_CMD[PER] bit.

Work-around:

1. To prevent a PCI Parity error from being reported to a PLB master, mask PCI parity errors and disable error reporting by setting PCIC0_CMD[PER]=0 and PCIC0_ERREN[MEAE]=0. Note, clearing the PLB Error Assertion Enable (PCIC0_ERREN[MEAE]=0) prevents any bus error from being reported to a PLB master.
2. There is no work-around to guarantee a PCI parity error is reported to a PLB master when PCI parity errors are unmasked and error reporting is enabled, PCIC0_CMD[PER]=1 and PCIC0_ERRSTS[MEAE]=1.

PCI_27 The internal PCI arbiter unfairly arbitrates when a master removes its request before the transfer.

Category: 3

Overview:

The internal PCI arbiter does not track fairness in the situation given in *Table 5* and described below.

Cycle 1. Master A performs a read arbitration request (via one of PCIReq1:5) to the PPC405GP (inbound).
Cycle 2. Master A is granted the bus.
Cycle 4. Master B performs a read arbitration request to the PPC405GP (inbound).
Cycle 5. The PPC405GP retries the request from Master A (by asserting PCIStop and deasserting PCITRDY). The read is handled as a delayed transaction.
Cycle 6. When Master A is retried, Master B deasserts its read request for one clock cycle.
Cycle 7. Master B is granted the bus and begins a read cycle on the bus.
Cycle 9. Master B is retried by the PPC405GP.
Cycle 11. One PCI clock cycle after the retry in cycle 9, Master B detects the bus is idle with the grant going away and immediately starts a new read cycle.
Cycle 13. Master B is again retried because the PPC405GP has read data available for Master A.

Cycles 11 through 14 repeat. With the early removal of the Master B read request in cycle 10, the arbiter fairness algorithm does not block Master B and Master A does not receive grant long enough to obtain the bus.

Table 5. PCI Tenure Sequence demonstrating erratum PCI_27

Cycle	Master	REQ_A	GNT_A	REQ_B	GNT_B	FRAME	IRDY	DEVSEL	STOP
1		0	1	1	1	1	1	1	1
2		0	0	1	1	1	1	1	1
3	A	0	0	1	1	0	1	1	1
4		0	0	0	1	1	0	1	1
5		0	1	0	0	1	0	0	0
6		1	1	1	0	1	1	1	1
7	B	1	1	0	1	0	1	1	1
8		0	1	0	0	1	0	1	1
9		0	1	0	0	1	0	0	0
10		0	1	1	0	1	1	1	1
11	B	0	1	1	1	0	1	1	1
12		0	0	0	1	1	0	1	1
13		0	1	0	0	1	0	0	0
14		0	1	1	0	1	1	1	1
15	B	0	1	1	1	0	1	1	1
16		0	0	0	1	1	0	1	1
17		0	1	0	0	1	0	0	0
18		0	1	1	0	1	1	1	1

PCI_27 Continued

Master B removes its read request in cycle 10 before the arbiter records which master received the grant. Subsequent requests by Master B are treated as its first. The arbiter repeatedly grants the bus to Master B and does not give Master A an opportunity to complete the delayed read. This situation continues until either the delayed-read discard timer expires or the PPC405GP generates an outbound request. The delayed-read discard timer expires after 2^{15} PCI clocks.

Impact:

The PCI arbiter repeatedly grants the bus to only one master resulting in a live lock condition.

Work-around:

Perform one of the following if a PCI Master behaves as Master B described in the overview:

1. Delay the $\overline{\text{PCIReq}}$ from the PCI Master by one PCI clock by inserting a flip-flop between PCI Master $\overline{\text{PCIReq}}$ pin and the PCI arbiter (arb_req# = flip_flop_Q; flip_flop_D = master_req#, flip_flop_CLK = PCI_CLK)

   If timing can be met, the output of the flip-flop can additionally be AND'ed with the input for better performance (arb_req# = flip_flop_Q & flip_flop_D; flip_flop_D = master_req#, flip_flop_CLK =PCI_CLK)
2. Use an external PCI arbiter.

PLB_4 Incorrect data may be stored in the PLB0_ACR during a DCR read after DCR write sequence.

Category: 3

Overview:

If a DCR write operation (mtdcr) to the PLB Arbiter Control Register (PLB0_ACR) is immediately followed by a DCR read operation (mfdcr) of the same register, the PLB0_ACR will latch in 0s instead of the intended DCR write data.

Impact:

Incorrect data may be stored in the PLB0_ACR.

Work-around:

Do not read the PLB0_ACR register immediately after writing it. Separate a DCR read from a DCR write of the PLB0_ACR with a DCR read of any register other than PLB0_ACR.

SDRAM_10 Command leadoff settings (SDRAM0_TR[LDF]) of 3 more MemClkOut cycles do not always generate the expected minimum delay.

Category: 4

Overview:

The SDRAM0_TR[LDF] parameter controls the number of clock cycles from address/command assertion to bank select (/BankSeln) assertion. For settings of 3 or more MemClkOut cycles, the SDRAM Controller does not always generate the expected minimum delay. This error occurs when /CAS remains asserted between successive read or write commands. In these cases, the minimum delay is 2 MemClkOut cycles instead.

Impact:

Most systems are not affected by this erratum since SDRAM0_TR[LDF] is typically set for 2 MemClkOut cycles. If SDRAM timing cannot be met with a delay of 2 MemClkOut cycles, the SDRAM controller must be run at a lower frequency.

Work-arounds:

For systems affected by this erratum, the SDRAM Controller must be run at a lower frequency. Refer to chapter 7 of the PPC405GP User's Manual to select a clock ratio for generating a lower MemClkOut frequency.

SDRAM_12 SDRAM controller may issue unexpected mode register set command.

Category: 3

Overview:

This erratum can only occur when the SDRAM controller Burst Read Prefetch Granularity is set to 32 bytes, SDRAM0_CFG[BRPF]=2'b10.

When the SDRAM controller in the PPC405GP is initialized it issues a Mode Register Write (MRW) command to the SDRAM memory. This MRW command is encoded on the SDRAM address bus and serves to configure SDRAM memory device interface parameters including the CAS latency, burst order, and burst length. Once initialized, the SDRAM controller never issues another MRW command to the SDRAM memory.

In normal operation, the controller auto-refreshes SDRAM memory at regular intervals as determined by the setting in the Refresh Timer Register, SDRAM0_RTR. When a refresh is scheduled to occur, the SDRAM controller should wait for any in-progress SDRAM data transfers to complete, precharge any open SDRAM memory pages, and then issue the auto-refresh command to the memory. If the SDRAM controller is programmed for a Burst Read Prefetch Granularity of 32 bytes, SDRAM0_CFG[BRPF]=2'b10, the SDRAM controller may not wait for the current SDRAM transaction to complete and incorrectly superimpose a precharge command onto the SDRAM address/control bus. If this occurs at a time when /CAS is low the SDRAM will be presented with an unexpected MRW command. The specific mode register value written to the memory is indeterminate as it corresponds to whatever address the SDRAM controller was driving at the time of the unexpected MRW command. After being reprogrammed with an incorrect mode register value correct data transfer between the SDRAM memory subsystem and PPC405GP is no longer possible.

Impact:

Incorrect data may be read and/or written to SDRAM memory.

Work-arounds:

Program the SDRAM Burst Read Prefetch Granularity to 16 bytes by setting SDRAM0_CFG[BRPF]=2'b01. This is the default, recommended value as documented in the PPC405GP User's Manual. Note that selecting a prefetch granularity of 32 bytes does not improve performance and in some instances may actually result in lower throughput as extra data is unnecessarily read from SDRAM.

UART_1 Reading the UARTx_IIR may cause an interrupt to be lost and incorrect status to be read.

Category: 3

Overview:

The interrupt pending status, UARTx_IIR[IP], is delayed by one on-chip processor bus (OPB) clock cycle relative to the interrupt priority, UARTx_IIR[IPL], when the UARTx_IIR is updated. The following are two situations where this error may be observed:

1. There is a one cycle window where the UARTx_IIR[IPL] bit field holds status for an interrupt condition while the interrupt pending bit, UARTx_IIR[IP], indicates no pending interrupt.
2. There is a one cycle window where the interrupt priority bit field has been cleared, UARTx_IIR[IPL]=0b000, because the interrupt condition is no longer true and the interrupt pending bit UARTx_IIR[IP] indicates an interrupt is pending.

A Transmit Hold Register Empty interrupt (THRE) can be lost if the UARTx_IIR is read during the clock cycle when the UARTx_IIR[IPL] bits indicate the THRE condition but the UARTx_IIR[IP] bit indicates no interrupt pending. In this case, reading the UARTx_IIR will clear the interrupt condition and the PPC405 CPU will never see the UARTx_IIR[IP] bit indicate that an interrupt occurred.

Impact:

The PPC405 CPU may read an incorrect status and not detect a pending interrupt by reading the UARTx_IIR.

Work-arounds:

Perform one of the following:

1. Poll the UART for transmitter and receiver status by reading the Line Status Register (UARTx_LSR). By the time the PPC405 CPU can read the UARTx_IIR register after the line status register indicates that the transmitter is empty, the interrupt pending bit is cleared.
2. Evaluate the UARTx_IIR register value using both the interrupt priority level bit field and the interrupt pending bit to determine if an interrupt is pending. If any bits in the UARTx_IIR[IPL] bit field are set or if the UARTx_IIR[IP] bit is cleared, an interrupt has occurred.

Design Notes:

1. When the processor is not powered (i.e., Vdd and OVdd = 0), additional precautions are needed if the IIC SDA and SCL are powered. Under this circumstance, the processor is in a high-impedance state and may be damaged and/or experience a degradation in reliability. (Note: The IIC SDA and SCL I/Os are 5V tolerant which implies that the voltage on these I/Os may be as high as 5.5V.) The following critical items must be considered:
   a. If the input voltage to the PPC405xx/NPe405x exceeds the ESD protection diode drop and is supplied with a current in excess of 147mA, the IIC SDA and SCL I/Os will be permanently damaged.
      i (Vsource – (4*0.8)/(Rsource)
      ii Vsource = source voltage = 5.5V maximum
      iii Rsource = source resistance = Pull-up Resistor for 0.147 = (5.5 – 3.2)/Rsource or Rsource = 2.3/0.147 = 15.65 ohms (minimum)

      Raising the value from this minimum of 15.65 ohms and/or reducing Vsource from 5.5 V will give added margins of safety against the I/O being permanently damaged.
   b. A sustained (i.e. several hours) maximum current greater than 8.1 mA will weaken or degrade the reliability of the IIC SDA and SCL I/Os. To improve reliability choose a pull-up resistor, Rsource, greater than 284 ohms, 0.0081 = (5.5 – 3.2)/Rsource = 284 ohms. (The recommended value for Rsource in the processor data sheet is 3K ohms. 284 ohms is the minimum for protection against damaging currents and 3K ohms is the maximum that guarantees a logic 1/high with a pull-up.)
   c. Use a series resistor to ensure that the voltages applied to SDA and SCL inputs do not exceed a 3.2 V drop across the ESD diodes when Vdd = OVdd = 0 V. The choice of series resistor depends on the pull-up resistor used and the required voltage levels at the inputs of the other devices on the IIC bus (Vih, min = 2.4 V).
2. The SDRAM controller does not support mixing ECC and non-ECC memory banks.

4. To avoid generating extraneous interrupts when waking the Universal Interrupt Controller (UIC), perform the following steps before enabling sleep mode, CPC0_ER[UIC]=1, and after disabling sleep mode, CPC0_ER[UIC]=0.

   Prior to enabling sleep mode:
   1. Save the contents of the UIC Masked Status Register (UIC0_MSR).
   2. Save the contents of the UIC Enable Register (UIC0_ER).
   3. Set UIC0_ER to zero. (Disable all interrupts.)

   After exiting sleep mode:
   1. Write the ones complement of the saved contents of the UIC0_MSR to the UIC0_SR.
   2. Restore the state of the UIC0_ER.

5. See application note, “Using a Spread Spectrum Clock Generator with the PowerPC 405GP”

Revision Log

Rev	Contents of Modification	
August 20, 2002	Errata document version 3.0.0 • Items added: EMAC_5, EMAC_7, IIC_13, IIC_14, IIC_16, UART_1 • Items updated: PCI_26 • Changed item numbers for the following items:	
	Old Item Number	New Item Number
	2	EBC_5
	9	CPU_147
	10	CPU_162
	15	CPU_121
	25	IIC_6
	26	IIC_7
	27	IIC_9
	28	IIC_10
	29	IIC_11
	30	IIC_12
	39	DCP_2
	51	CPU_190
	53	CHIP_11
	62	PCI_23
	65	CPU_197
	66	CPU_200
	67	CPU_201
	68	PCI_18
	69	CPU_202
	70	PCI_24
	72	CPU_206
	73	DCP_6
	74	DCP_9
	75	CPU_208
	76	CPU_209
	77	CPU_210
	78	SDRAM_10
	79	PLB_4
	81	PCI_26
	D3	1
	D5	2
	D7	4
	D9	5
November 8, 2002	Errata document version 4.0.0 • Item added: PCI_27 • Modified introduction (page 1)	
January 31, 2003	Errata document version 5.0.0 • Modified items: EMAC_5, PCI_24 • Item added: PCI_25	
May 15, 2003	Errata document version 6.0.0 • Items added: CPU_213, EBC_20, EMAC_8, EMAC_9, EMAC_10, SDRAM_12	