PPC440G5V4 Errata

Technology: IBM Cu-11

ASIC Library Release Version: V10

Core Version: V4

This is a comprehensive document describing all known errata applicable to the PPC440G5V4 CPU core. Each errata is classified according to its category, which is a number 1 to 5 that represents a combination of the errata's impact on system performance and the ability of any proposed work around to minimize that impact. The following table explains the meaning of each category.

Categories

Errata Category	Category Definition
1	Major impact, no work around 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 is 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.

Errata Summary

The following table summarizes specific errata that are described in this document.

Errata	Fixed In	Category	Abstract	Page
37		3	Instruction stream may be corrupted if icbt executed in guaranteed mode with the target address matching the first line of an in-progress speculative pre-fetch	4
38		3	Instruction stream may be corrupted if the speculative pre-fetch mechanism is enabled to pre-fetch two or three lines	6
39		5	Exception Syndrome Register (ESR) Machine Check field not cleared by processor reset if Debug_Halt signal held active during reset	7
40		3	Instruction stream may be incorrect if an instruction fetch miss which is a Branch Target Address Cache (BTAC) hit causes a speculative pre-fetch request that matches the address of an existing line fill request	9

Design Notes

This section contains design-related recommendations that are based on the errata described in this document. The following table summarizes design notes contained in this section.

Item	Description	Date First Documented	Date Last Updated
1	Because use of the burst prefetch capability can cause multiple errata, it is recommended that this mode not be used. Use of this mode is no longer supported.	January 19, 2004	
2	An unsynchronized *dcread* instruction in the pipeline can cause erroneous data in a cache line flush from the data cache that is waiting to be written to memory via the PLB interface.	October 17, 2005	

1. Multiple errata items have been discovered and documented that are associated with the use of the I-cache PLB interface burst prefetching mode, an optional capability of the various implementations of the PPC440 CPU core. This capability, which is disabled at reset, is enabled by setting the CCR0[ICSLC] =/ 0b00. When CCR0[ICSLC] == 0b00, only PLB “line” transactions are performed. This is the “recommended” setting. Because use of this burst prefetch capability can cause multiple errata, it is recommended that this mode not be used. Use of this mode is no longer supported.
2. A *dcread* instruction in the pipeline, whether executed or not, may incorrectly modify the data of a cache line flush from the data cache (either by a cache line replacement or by a *dcbf* instruction) that is waiting to be written to memory via the Processor Local Bus (PLB) interface. When this condition occurs, the flush data may be overwritten and stored into the memory. This condition can occur even when the *dcread* is not actually executed because the *dcread* can be issued into the load / store pipe (L-pipe) even though it is not committed, and in fact, the execution stream branches away from the *dcread* because the conditional branch has not yet resolved at the time of the instruction issue. The presence of the *dcread* in the first stage of the L-pipe is sufficient to cause improper modification of the flush data. This *dcread* will never be executed or completed architecturally because of the incorrect branch prediction, but it will impact the data in the flush buffer. Note that this condition only occurs in “privileged” (supervisor) mode as *dcread* is a privileged mode instruction and its occurrence in “user” mode results in the *dcread* being processed as an illegal instruction (it will not be issued).

   Though this condition is not deterministically detectable, it is normally avoided by following the required programming practice of properly synchronizing the pipeline around the *dcread* instruction, specifically by preceding *dcread* with *msync* and following it with *isync* (*msync* → *dcread* → *isync*). There are, however,

two additional scenarios that must be considered to insure that an “unsynchronized” *dcread* is not allowed in the pipeline:

A. During debug activity, *trap* instructions can be placed in the instruction stream and replace arbitrary instructions in that stream. If the *msync* instruction is replaced by a *trap*, then the *dcread* will be issued even though the *trap* will cause the code stream to branch away from the *dcread* instruction. It is the responsibility of the programmer to not use *trap* instructions to cause a breakpoint on the *msync* instruction. Breakpoints can be implemented using the Data Address Compare debug facility.

B. In the case of normal program code (executing in supervisor mode), the occurrence of a conditional branch instruction with the branch target address not referencing legal PowerPC instruction code (e.g. pointing to a data area or random / uninitialized memory space) can result in a “random” *dcread* opcode being fetched into the pipeline and issued with the appropriate synchronization instructions. The specific conditions required to trigger this condition (in addition to those specified above) are:

   - The conditional branch is mispredicted
   - The mispredicted address location of the branch has a *dcread* opcode without the required synchronization instructions.

   Note that only conditional branch cases can trigger this condition.

   This scenario can be avoided by insuring that all branch targets contain legal PowerPC instructions. Investigation has shown that commonly available compilers generate valid instruction sequences both following and at the target of conditional branches.

Errata #37

Errata: Instruction stream may produce incorrect data if icbt is executed in guaranteed mode with the target address matching the first line of an in-progress speculative pre-fetch

Category: 3 (major impact, work around available. Application of work around reduces impact to minor)

Overview

Instruction cache block touch (**icbt**) instructions can be executed in one of two modes: guaranteed, or non-guaranteed. The mode is selected by the CCR0[GICBT], where a value of 1 specifies guaranteed mode. Guaranteed mode **icbt** instructions are guaranteed to perform the filling of the targeted instruction cache line into the instruction cache, assuming all architectural conditions are met (such as the memory page containing the targeted line being marked as cacheable via the associated TLB entry). Non-guaranteed mode **icbt** instructions, on the other hand, may or may not perform the filling of the targeted instruction cache line, depending on the particular condition of the instruction fetch mechanism at the time of the execution of the **icbt** instruction. Generally speaking, instruction cache line fills initiated by the execution of non-guaranteed mode **icbt** instructions may be cancelled if the instruction fetch mechanism determines that some other instruction cache line fill is required in order to keep the instruction execution stream moving.

If a guaranteed-mode **icbt** instruction for which the target address matches the address of an earlier cache line fill request which is the first line of a speculative pre-fetch burst request which has not yet been requested on the instruction read PLB interface, then a duplicate line fill request for the same line may be presented to the interface, resulting in multiple copies of that line in the instruction cache. This in turn could lead to an incorrect instruction stream.

Additional circumstances are also necessary, involving the precise timing of the execution of the **icbt** instruction and the state of the speculative pre-fetch request within the PPC440 core. The specific nature of these details are beyond the scope of this document.

Impact

This errata can result in the improper operation of the executing program, with undefined results.

Work Arounds

There are three possible work arounds for this errata.

1. Disable speculative pre-fetching altogether, by setting CCR0[ICSLC] to 2b00; or
2. Set the mode for **icbt** execution to non-guaranteed, by setting CCR0[GICBT] to 1b0; or
3. Carefully manage the code sequence and environment when executing guaranteed-mode **icbt** instructions in order to guarantee that the target of such an **icbt** instruction does not match the starting line of a speculative pre-fetch request.

In order to accomplish the third option shown above, software could do one of several things. First, system software could temporarily disable speculative pre-fetching prior to beginning the execution of a series of guaranteed-mode **icbt** instructions, and then re-enable speculative pre-fetching when finished. This could be accomplished with the following code sequence:

mtccr0	enable guaranteed-mode and disable speculative pre-fetch
isync	ensure completion of mode change
icbi	any address, ensures previous speculative pre-fetches have moved to a stage where they cannot conflict with subsequent **icbt** instructions
icbt	first (of perhaps several) guaranteed-mode **icbt** instructions
msync	ensure completion of all preceding **icbt** instructions
mtccr0	disable guaranteed-mode and re-enable speculative pre-fetch

Alternatively, software could ensure that a series of guaranteed-mode **icbt** instructions are always preceded by an **icbi** instruction (to any target address, as shown in the code sequence above), and then followed by no less than 14 no-op instructions. Furthermore, the guaranteed-mode **icbt** instructions must be known to not be targeting any of the instruction cache lines containing the **icbt** instructions themselves, or the 14 no-op instructions, or the instruction cache line immediately following the last one containing the 14 no-op instructions. By following the **icbt** instructions with this number of no-ops, software can guarantee that the instruction pre-fetch mechanism has not predicted any branch instruction targets such that it might be pre-fetching from an indeterminate address. If this technique is used, software must also ensure that interrupts do not occur during the guaranteed-mode **icbt** sequence; otherwise the first **icbt** executed upon returning from the interrupt may encounter a conflict with speculative pre-fetching that may have been initiated while executing the interrupt handler.

Finally, it may be the case that software is knowledgeable enough of the branching and other instruction discontinuity circumstances preceding and succeeding any guaranteed-mode **icbt** sequence (including the possible targets of the **icbt** sequence), that it is assured that the **icbt** instructions do not target any possible starting line of a speculative pre-fetch request that could be in progress at the time of the execution of the **icbt** instructions. The manner in which this is ensured may be system dependent, but if known, then the other software limitations shown above could be avoided.

Errata #38

Errata: Instruction stream can be incorrect if the speculative pre-fetch mechanism is enabled to pre-fetch two or three lines

Category: 3 (major impact, work around available. Application of work around reduces impact to minor)

Overview

The instruction stream can become incorrect if the speculative pre-fetch mechanism is enabled to pre-fetch either two or three additional instruction cache lines after an instruction cache miss. More specifically, the following conditions can cause this:

- If the speculative pre-fetch mechanism is enabled by setting CCR0[ICSLC] to either 0b10 or 0b11 (that is, two or three lines),

and

- If the PPC440 begins a burst request for these two or three lines on the instruction PLB interface, but the instruction stream is re-directed prior to the completion of the speculative pre-fetch burst operation (e.g., due to an interrupt or a branch misprediction),

but

- The new instruction stream comes back to an address within the second or third line of the speculative pre-fetch burst.

Additional circumstances are also necessary, involving the precise timing of the arrival of the speculative pre-fetch data and the re-direction of the instruction stream, but these details are beyond the scope of this document.

Impact

If this scenario occurs, the results of any subsequent instruction execution will be undefined.

Work Arounds

There are two possible work arounds for this errata:

1. The speculative pre-fetch mechanism must either be disabled (CCR0[ICSLC] set to 0b00) or set to only pre-fetch one cache line (CCR0[ICSLC] set to 0b01); or
2. The PLB subsystem attached to the instruction PLB interface of the PPC440 core must assert the “Burst Terminate” signal in response to any speculative pre-fetch burst request of two or three cache lines, and in such a fashion as to guarantee that only the first cache line of the speculative pre-fetch burst will be received by the PPC440. This is accomplished by asserting the “Burst Terminate” signal in the same cycle in which the next-to-last “Data Acknowledge” associated with the first cache line is asserted. This causes the PPC440 core to deassert the “Burst” output signal prior to, or coincident with, the last “Data Acknowledge” assertion associated with the first cache in the first cache line, thereby ending the burst transfer.

Errata #39

Errata: Exception Syndrome Register (ESR) Machine Check field not cleared by processor reset if Debug_Halt signal held active during reset

Category: 5 (minor impact -- work around available)

Overview

The ESR[MCI] field indicates that a Machine Check exception has been detected as associated with an instruction, and execution of that instruction has been attempted. Under normal operation, such an occurrence causes a Machine Check interrupt, and the interrupt handler software clears ESR[MCI]. If, however, such instruction Machine Check exceptions occur while Machine Check interrupts are disabled (due to MSR[ME] being 0), the processor continues its attempts to execute instructions (with or without machine checks associated with them), and ESR[MCI] remains set until it is either cleared by software or by reset.

The ESR[MCI] field is provided as a Machine_Check core output signal so the core can indicate to system logic that Machine Check exceptions associated with instruction execution have occurred. System logic can monitor this signal and provide some indication (e.g., via a light on a development board) to the user that such exceptions have occurred. Accordingly, ESR[MCI] is cleared by processor reset so the processor will give a proper indication of whether any new such exceptions occur during the execution of the initial instruction execution sequence after the reset operation.

The Debug_Halt signal is used to hold the processor in a “stopped state”, in which instruction execution is suspended awaiting user-initiated debug operations via the JTAG interface. This signal can be asserted during the reset operation, causing the processor to stop at the completion of the reset operation (before executing the first instruction at the reset address).

If the Debug_Halt input signal to the PPC440 core is held active while the processor is reset, the ESR register will not be automatically cleared by hardware. Specifically, the ESR[MCI] field will not be reset to 0 as intended (its value will be undefined), and hence the Machine_Check output signal of the core may or may not be asserted after the reset operation.

If the processor is reset in this fashion, and the ESR[MCI] field happens to contain the value 1 after the reset operation completes, then when the Debug_Halt signal deasserts and the processor begins executing the initial instruction sequence, it does so with the Machine_Check output signal asserted until the ESR[MCI] field is cleared by software.

Note that the assertion of Debug_Halt during the reset operation does not, in and of itself, *introduce* the assertion of the ESR[MCI] field. That is, in order for ESR[MCI] to be asserted after a reset operation with Debug_Halt asserted, there must exist some legitimate cause for the assertion of ESR[MCI] prior to the reset operation, in which case the assertion of Debug_Halt simply prevents ESR[MCI] from then getting cleared by reset. Similarly with a power-on reset, the state of ESR[MCI] is undefined prior to the reset operation and hence it may continue to be asserted after such a reset with Debug_Halt asserted. However, with a warm (non-power-on) reset, if ESR[MCI] is 0 prior to the reset operation, it will continue to be 0 after reset, independent of the assertion or deassertion of Debug_Halt during the reset operation.

Impact

If the processor is reset in this fashion, the system may receive an incorrect indication that Machine Check exceptions have occurred subsequent to the reset operation.

Work Arounds

System software must clear ESR[MCI} as part of the initial instruction sequence after the reset operation, and system hardware must be tolerant of the potential for the assertion of the Machine_Check output signal after the reset operation and prior to software having cleared ESR[MCI].

Errata #40

Errata: Instruction stream may be incorrect if an instruction fetch miss which is a Branch Target Address Cache (BTAC) hit, causes a speculative pre-fetch request that matches the address of an existing line fill request.

Category: 3 (major impact, work around available. Application of work around reduces impact to minor)

Overview

The instruction stream may be incorrect if two line fill requests are made for the same instruction cache line. This scenario can occur if an instruction fetch miss which is a Branch Target Address Cache (BTAC) hit, causes a speculative pre-fetch request that matches the address of an existing line fill request.

A fetch miss of an instruction which gets a BTAC hit will cause the next fetch address to be redirected to the target of the branch. This speculative prefetch should not be allowed to execute. The prefetch continues through the processor pipeline, and eventually generates a line fill request. If another fetch miss of the same address is in the pipeline at the same time, two line fill requests will be made for the same line, resulting in duplicate tags in the Instruction Cache.

There is a mechanism that recognizes a speculative pre-fetch request matches a previous fetch miss, and thus prevents the speculative prefetch from making a line fill request. However, this mechanism does not recognize the duplicate prefetch in this case because the address that represents the speculative prefetch has been updated with the target of the BTAC hit.

Impact

If this scenario occurs, the results of any subsequent instruction execution will be undefined. Duplicate tags in the instruction cache data array can have two effects:

1. If parity error detection is enabled, a parity error exception will occur when a fetch with the same tag performs a search in the array.
2. If parity error detection is not enabled, a fetch will have undefined results.

Work Arounds

There are three possible work arounds for this errata:

1. Disable speculative prefetching by setting CCR0[ICSLC] to b00.
2. Disable the BTAC by setting CCR0[DBTAC] to b1.
3. Carefully manage the code sequence and environment so a branch that hits in the BTAC will not miss in the I-cache. This work around is not practical to implement in a normal software development and operating system environment.

Revision Log

Revision Date	Contents of Modification
October 17, 2005	Initial distribution for Version 4 of PPC440G5.

Printed in the United States of America

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 work arounds as needed.

This document lists errata that IBM has characterized as of . 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 12533-6531

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