MIPS R10000 Errata, Processor Revision 2.3, 2.4 & 2.5

July 2, 1996

MIPS Technologies Inc.
2011 N Shoreline Blvd
PO Box 7311
Mountain View, CA 94039-7311

MIPS Confidential

Errata #1: When an interrupt occurs while an uncached load, which results in a bus error, is executing, the interrupt must have lowest priority for an uncached operation; thus, R10000 should take the bus error exception. However, due to this bug the processor takes the *interrupt exception* before the *bus error exception*.

This bug only affects the order in which the two exceptions are taken. One affect could be, for example, if the external agent sends an interrupt whenever it returns a bus error response and the EPC is not changed in the handler there would be an infinite loop of the interrupt exception being taken, the uncached load being retried, and the interrupt exception being taken again. If there is no relationship between these 2 types of events, each happen only spuriously, AND it is OK to occasionally retry the uncached load which gets a bus error, the user wouldn't be affected. The bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

Workaround: There is no workaround, except disabling interrupts during uncached loads, if the order of the exception matters.

Errata #2:There are certain problems in the "*Performance Counters*". These problems were fixed and certain enhancements were made in revision 3.0.

a) When the *Performance Counter* is configured to count integer operations, the instructions which are issued but immediately aborted, because an operand is not available, are also counted. In revision 3.0 this was changed to not count aborted instructions.

b) When the *Performance Counter* is configured to count load issued, Prefetch instructions are not counted. This bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

c) When the *Performance Counter* is configured to count instructions issued, MADD instructions will be counted twice. Also, conditionally[1] or tentatively[2] issued floating point instructions that are killed increment the count. This bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

d) When the *Performance Counter* is configured to count total instructions issued, load/store instructions are counted each time they are retried. (A load/store can be re-issued if it does not complete its tag check cycle - either because of the DCache Tag being busy or there are already four ld/st had a miss in the cache and the newer one keeps re-trying every four cycles). Correctly, when the Performance Counter is configured to count loads issued or stores issued, the instructions should counted only once. This bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

e) In revision 2.6 and earlier revisions, when the *Performance Counter* was configured to count "decoded branches", the counter is incremented when a branch (conditional or unconditional) instruction is decoded (include those aborted & resolved) and inserted into the active list; even though, it may still be killed due to an exception or a prior mispredicted branch. In the revision 3.0, the counter is only incremented after the branch is resolved (determined if the prediction is correct or not).

f) Note that in rev 2.6 and earlier revisions, if a store graduates on a given cycle, all loads which graduate on that cycle do not increment this counter. However, revision 3.0 was fixed to count all loads.

g) In rev 2.6 and ealier revisions, this counter is incremented on the cycle after a virtual

1. Conditionally issued: If an instruction depends on the result of a previous instruction of the same class, depending on the availability of the execution unit, the instruction is issued to the execution unit assuming that the result will be available in the register in the next cycle. However, if the result is not ready then the issue is aborted and re-tried again.

2. Tentatively issued: If an instruction depends on a previous load instruction, depending on the availability of the execution unit, the instruction is issued to the execution unit assuming that the operand will be available in the register in the next cycle. However, if the result is not ready then the issue is aborted and re-tried again.

address coherence condition is detected, provided that the access was not flagged as a miss. This condition can only be raised for virtual page sizes of 4KB. In revision 3.0, this counter's meaning is changed. It is incremented on the cycle after either ALU1, ALU2, FPU1, or FPU2 marks an instruction as "done."

Workaround: No workaround.

Errata #3: Consider the following sequence of events which target a coherent cache line which is initially shared between processors 'A' and 'B' in a multiprocessor system with duplicate tags and CohPrcReqTar=0. The external agent(EA) then receives an Upgrade request from processor 'A' and then another Upgrade request from processor 'B' before it has a chance to send an invalidate intervention to 'B'. Generally, when an external agent(EA) receives an Upgrade request from it's processor 'A' and an Invalidate from another processor 'B', the EA will send a NACK for the Upgrade to proc.'A' and send an Invalidate to proc. 'A' for the same line. This will lead the processor 'A' to redo the Store which will cause cache miss and, in turn, a Coherent Read Excl. from proc. 'A'. However, there is another way to handle the conflict between the Upgrade and the Invalidate. The EA can send an ACK for the 'B's Upgrade followed by an External Intervention with exclusive to 'B' and provide Data Response for the Upgrade to 'A'. The data response for the Upgrade implies that the R10K was requested to invalidate the line after it presented it's upgrade request on the bus.

The bug that we have found is triggered when there is a data response given for an upgrade request. It could cause the R10000, which receives the block for its upgrade request, to not write-back data due to the store instruction which caused the upgrade. The bug exists in revision 2.5 and all previous revs. This bug will be fixed in revision 2.6.

Workaround: There is no workaround for this bug except just NACK'ing the Upgrade Request instead of sending the Data Response.

Errata #4: There is an 8 entry ITLB which is maintained as the subset of the JTLB. For every TLB write, the index of the JTLB entry, referenced by the TLB write, is compared to that in the ITLB. If there is a match, the valid bit for the ITLB entry is cleared to maintain the subset property. In certain cases, due to a glitch in the compare logic the ITLB entry is invalidated when it is not required. This bug do not cause any functional problem. It only causes the execution to, intermittently, take a few extra cycles to refill the ITLB from the JTLB (transparent to the user), which might result in a minuscule performance degradation. Due to the indeterministic behavior, this bug could also result in deviation between two processors operating in "lock-step"; such as, in fault tolerant systems. This bug exists in revision 2.4 and all previous revs. This bug will be fixed in revision 2.5.

Workaround: There is no real workaround; however, lowering voltage and PClk frequency might help. A software work around, to remove indeterministic behavior, would be to flush the ITLB just prior to writing the JTLB.

Errata #5: The bug occurs only when You've shifted data into the boundry scan chain then leave the shift-DR state for a while, then return to shift-DR and want to continue shifting data into the BSChain. You'll see the bug only if you care about preserving the previously shifted data. For example:

1. shift-DR (example - shift in 16 bits)
2. exit1-DR
3. pause-DR (example - wait an hour)
4. exti2-DR (tdi is undefined by 1149 standard here so it can be 0,1 or Z)
5. shift-DR (example - shift in 16 bits)

6. repeat steps 1-5 until shifted in enough bits to test board

(The exact names of the above state names may be different from the JTAG standard)

The problem occurs when re-entering shift-DR (step 5 above). The data held by the first flip-flop in the chain is overwritten before it is shifted into the 2nd FF. The data is lost and the undefined value on tdi from the pause-DR state is stored by the 1st FF in the boundry scan chain. This bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

Workaround: The only workaround is not to go though the Puause-DR state. This may be OK for some test assuming that you can shift in all the Bits in the boundary scan register continuously.

Errata #6: Consider the execution of a *ll/sc* instructions sequence with the target address in catchable coherent space and the local cache state initially shared. When the sc is executed, R10000 needs to make an *upgrade request* on the *SysAD* bus. If the external completion response for this upgrade is an Error, R10000 is supposed to take a *DBE exception*. The bug causes R10000 to instead *invalidate* the primary cache line and complete the *sc* with a failure result. Some systems may persistently return an *Error Response* for this upgrade, i.e. if it means to allow the R10000 to read the address but prevent it from writing it, which would result in a live-lock situation with the R10000 stuck spinning in the *ll/sc* loop. This same bug applies to the execution of *lld/scd*, but does not cause any incorrect behavior for the execution of any other store instruction. This bug exists in revision 2.6 and all previous revs. This bug will be fixed in revision 3.0.

Workaround: Avoid any situation that persistently returning an Error Response as stated above.

Errata #7: Invalidate followed by *ACK* causes protocol error or hang:

A sequence of coherent system bus activity may cause the R10000 to either commit a protocol error which could cause data corruption, or result in the R10000 hanging. The hang condition is interruptible and is associated with accessing a specific address. There is no way in software to either avoid the bug or verify that it was responsible for a failure without the help of the external agent or a logic analyzer trace.

The following sequence of activity occuring on the system bus is necessary but not sufficient for the bug to cause a failure.

(In the following description, two addresses A and B are used. The addresses are related in such a matter that when referenced by R10000, the cache lines retrieved would occupy the same index in the scache and have different tags. Also, all operations to the addresses are cache coherent.)

1) R10000 does an *upgrade request* for address A on the *SysAD* bus

2) Later, between 1 and 3 cycles before the external agent ACKs the request, it also sends an ex ernal invalidate request targetted at the R10000 for address B. The number of cycles depends on the value of *SysClkDiv*.

If this activity occurs, then either of the following types of failures are possible:

a) A subsequent attempt by the R10000 to execute a load, or store to address B would result in the instruction hanging.

b) Even though the R10000 will return a *Shared* state for address B (if approriate) using the SysState pins and should atomically invalidate the line, it may soon afterwards make an upgrade request for the cache line at address B. This is a coherency protocol error and could result in data corruption.

This bug exists in revision 2.5 and all previous revs. This bug will be fixed in revision 2.6.

Workaround: Avoid issuing an *external invalidate request* between 1 to 3 cycles before the *ACK response* is sent; or send the *external invalidate request* after the *ACK response*.

Errata #8: *Intervention* and outstanding request performed in wrong order.

Consider the following sequence of events:

R10000 does a *Processor Read Block Excl./Shared* with a *writeback* underneath it. The writeback is delayed from going on the bus (because the processor is busy doing something else or *SysWrRdyB* is de-asserted). *External Read Response* comes back followed by an *ACK*. An *Intervention Excl* for that same (read) address closely follows the *ACK*. In such a situation, the intervention should be held off till the line is refilled. However, due to this bug the intervention might (in some cases) be executed by the processor before the refill. This would cause the processor to, incorrectly, send the *state response* for the *intervention* as *invalid* while the state of the refilled line would not end up to be *invalid*. This bug exists in revision 2.5 and all previous revs. This bug will be fixed in revision 2.6.

Workaround: This bug rarely occurs if the separation between the *ACK*'ing of a *processor read request* and an *external intervention request* is 15 cycles or more (note that this is by default true if the *ACK*'ing occurs in conjunction with the first data cycle of the *read response*). The likelihood of hitting the bug is even less if the system doesn't have short memory latencies; and does not de-assert *SysWrRdyB* often. The probability of occurence depends, accumulatively, on the above three conditions.

Errata #9: Consider a *load*, *store*, or *pref* instruction which targets address "*A*", is executed due to incorrect speculation, and results in a R10000 request which receives an *Error completion response* from external agent. If a *pref* instruction which also targets address "*A*" is executed while the request is still outstanding, and the processor should soon afterwards take an exception due to an unrelated instruction or interrupt, then, there is a slight chance that the exception will be skipped or the wrong *EPC* will be recorded for the exception. This bug exists in revision 2.5 and all previous revs. This bug will be fixed in revision 2.6.

Workaround: The workaround is to not execute *pref* instructions for target addresses which could result in an Error external completion response.

Errata #10: Consider a *load*, *store*, or *pref* instruction which targets address "*A*", is executed due to incorrect speculation, and results in a R10000 request which receives an *Error completion response* from external agent. If another *load*, *store*, or *pref* instruction to the same address "*A*", is executed, soon after; the second instruction could associate the *Error response* as its own and the R10000 could take a *DBE exception* for that instruction. This could be an issue if the system is designed such that one reference *"type A"* to a specific address should cause a *DBE exception* but a second *"type B"* to that address shouldn't (the reference *"type"* could be either *load* or *store* or *prefetch*). The R10000 could take an exception for the reference "*type B"*; if it recently decoded the *"type A"* after speculating incorrectly, and the *"type A"* targeted the same address as the first *"type"*. Note that in the above description *"type A"* is not same as *"type B"*. This bug exists in revision 2.5 and all previous revs. This bug will be fixed in revision 3.0.

Workaround: The workaround is to have the *DBE exception* handling routine allow for the possible case that the *bus request* which received the *Error response* was not caused by the instruction pointed at by the *EPC*, but a different type of instruction which targeted the same address.