ARM1026EJ-S Errata List

CPU Cores Division

Document number: AS007-RLNC-000034 6.0
Date of Issue: 25 Sept 2003

Abstract

This document describes the known errata in the ARM1026EJ-S r0p0, r0p1 & r0p2

Keywords

Errata, bug, workaround, ARM1026EJ-S.

Contents

1 ABOUT THIS DOCUMENT

1.1 Change History

Issue	Date	Change
1.0	13 September 2002	Initial version.
2.0	21 November 2002	Added scaled register LSL #2 LDR/STR erratum. Added Instruction dropped near Predicted Branch erratum. Added incorrect Processor Mode erratum. Added incorrect Q Flag for DSP multiply-accumulate following Bounced Coprocessor Instruction erratum. Added DCache drop of buffered stores due to stalled WB erratum. Added TCM DMA erratum.
4.0	11 April 2003	Added unpredictable Debug behaviour during DBGTAPSM transitions erratum. Added X-propagation during Gate-level simulations erratum.
5.0	5 June 2003	Added BIU big endian erratum. Added IEXT false abort erratum.
6.0	25 Sept 2003	Added MBIST Datalog uninitialized erratum.

1.2 References

This document refers to the following documents.

Ref.	Document No	Author(s)	Title
1	ARM DDI 0244B	ARM	ARM1026EJ-S Technical Reference Manual, Revision: r0p2

1.3 Scope

This document describes the errata discovered in the implementation of the ARM1026EJ-S r0p0, r0p1 and r0p2 categorised by level of severity. Each description includes:

- where the implementation deviates from the specification
- the conditions under which erroneous behaviour occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a ‘work-around’ where possible.

1.4 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
WB	(cache) Write Buffer

2 CATEGORIZATION OF ERRATA

Errata recorded in this document are split into three groups:

Category 1 Features which are impossible to work around and severely restrict the use of the device in all or the majority of applications rendering the device unusable.

Category 2 Features which contravene the specified behavior and may limit or severely impair the intended use of specified features but does not render the device unusable in all or the majority of applications.

Category 3 Features that were not the originally intended behaviour but should not cause any problems in applications.

2.1 Errata Summary

The errata associated with this product are categorised in the following way.

Errata Description (No.)	Rev r0p0	Rev r0p1	Rev r0p2
Category 1			
3.1) Incorrect scaled register offset/pre-indexed LSL#2 LDR/STR address generation	**Yes**	**No**	**No**
3.2) Instruction dropped near Predicted Branch from Non-Cacheable Area to Cacheable Area	**Yes**	**No**	**No**
3.3) Incorrect Processor Mode saved in SPSR after Data Abort followed by BLX	**Yes**	**No**	**No**
3.4) Buffered write gets dropped by DCache when DCache stalled by WB	**Yes**	**No**	**No**
3.5) BIU incorrectly handles 8- and 16-bit transfers in big endian	**Yes**	**Yes**	**No**
Category 2			
4.1) Erroneous Q Flag Value for Enhanced DSP Multiply-Accumulate following Bounced Coprocessor Instructions/Hardware Watchpoints	**Yes**	**No**	**No**
4.2) Erroneous ETM Branch Phantom Valid with Bounced Coprocessor Instructions	**Yes**	**No**	**No**
4.3) DMA interaction with TCM controller	**Yes**	**No**	**No**
4.4) Unpredictable behaviour when in Debug state during DBGTAPSM transitions	**Yes**	**Yes**	**No**
4.5) False entry into the Instruction Prefetch Abort handler	**Yes**	**Yes**	**No**
Category 3			
5.1) Unpredictable Behaviour of CP15 r15 MMU main TLB write test operations	**Yes**	**No**	**No**
5.2) X-propagation during Gate-level simulations	**Yes**	**Yes**	**No**
5.3) MBIST Datalog Uninitialized	**Yes**	**Yes**	**Yes**

3 CATEGORY 1 ERRATA

3.1 Incorrect scaled register offset/pre-indexed LSL#2 LDR/STR address generation (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #257.

3.1.1 Summary

If an LDR/STR with scaled register offset or pre-indexed LSL #2 addressing mode follows a conditionally NOT executed or failing LDM/STM or LDC/STC instruction, the address generated by the scaled register LDR/STR will be erroneous.

3.1.2 Description

The erratum is observed in the following sequence of 2 instructions:

```
LDM/STM or LDC/STC                             ; Fails condition codes.
LDR/STR scaled reg offset/pre-indexed LSL #2 ; Transfer occurs to wrong address!
```

3.1.3 Conditions

This bug occurs only when the conditional LDM/STM/LDC/STC fails AND the next instruction is a scaled register offset or pre-indexed LSL #2 LDR/STR. If the LDM/STM /LDC/STC conditionally executes OR there are additional instructions between them, the erratum will NOT occur.

3.1.4 Implications

Incorrect address generated by the scaled register LDR/STR instructions may cause the application to not work as intended. The erroneous LDR/STR transfers may appear to occur spuriously since the bug is triggered by the potentially data dependent state of the conditional codes when the LDM/STM/LDC/STC occurs.

It is advised to search through any application code source, listings or binaries for this scenario.

3.1.5 Workaround

The workaround is to place at least one instruction, like a NOP, between conditional LDM/STM/LDC/STC and any scaled register offset or pre-indexed LSL #2 LDR/STR instructions, with the obvious caveat that the added instructions aren't conditionally failing LDM/STM/LDC/STCs or scaled register offset or pre-indexed LSL #2 LDR/STRs.

Before:

```
      LDMGTIA r1,{r2-r8}       ; Conditional LDM
      STR r12,[r1,r7, LSL #2]  ; Scaled register STR
```

After:

```
      LDMGTIA r1,{r2-r8}       ; Conditional LDM
      AND r0,r0,r0             ; Insert NOP
      STR r12,[r1,r7, LSL #2]  ; Scaled register STR
```

3.2 Instruction dropped near Predicted Branch from Non-Cacheable Area to Cacheable Area (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #262.

3.2.1 Summary

In combination with certain code sequences, an instruction in the vicinity of predicted branch instruction that is resident in a non-cacheable area with a target in a cacheable area will be skipped.

3.2.2 Description

An instruction near a conditional predicted taken branch (a backwards conditional branch) or a return stack prediction (BX R14/ MOV pc, r14) can potentially be skipped. The branch instruction must be resident in a non-cacheable area, and the target of the branch must be in a cacheable area.

3.2.3 Conditions

The problem only occurs when branch prediction and the MMU/MPU are enabled and a predicted taken conditional branch or return stack instruction is used.

Predicted unconditional branches are not affected by this problem.

3.2.4 Implications

A skipped instruction will cause potentially fatal program flow disruption.

3.2.5 Workaround

Avoid predicted taken conditional branches and predicted return stack instructions between differing cache area types.

3.3 Incorrect Processor Mode saved in SPSR after Data Abort followed by BLX (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #264.

3.3.1 Summary

An instruction that data aborts and is immediately followed by a BLX instruction may have the processor state bit (ARM/Thumb) set to the state of the target of the BLX.

3.3.2 Description

Upon entry to the data abort service routine, the T-bit will be set incorrectly in the SPSR as well as the return address in `R14_abt`. The address in the `R14_abt` register will be incorrect, as the offset applied to the data abort instruction address will be based on the wrong processor mode.

3.3.3 Conditions

The problem only occurs when branch prediction is enabled and a predicted `BLX Imm` instruction immediately follows the instruction that data aborts.

Additionally, this problem will only be exhibited if another predictable branch is encountered within the first two instructions at the BLX target.

3.3.4 Implications

The data abort service routine will return to the incorrect processor mode, causing ARM instructions to be interpreted as Thumb instructions or vice versa.

3.3.5 Workaround

A single no-op (NOP) can be inserted between any instruction that can data abort (load/store) and a BLX Imm.

3.4 Buffered write gets dropped by DCache when DCache stalled by WB

(Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #266.

3.4.1 Summary

The DCache will drop a store when an STM is crossing a cache line boundary and the WB becomes full and stalls the DCache on the first write to the next cache line.

3.4.2 Description

The processor executes an STM, which crosses a cache line boundary. The last store to the first cache line is accepted by the WB, which then becomes full and stalls the data side memory system including the DCache. However the first store to the next cache line is pending. This store although sequential with respect to the STM instruction, must be treated as a non-sequential store by the DCache i.e. a tag lookup must be performed. This lookup does not happen and the DCache classifies the store as having missed in the DCache. The store is thus never written into the DCache data arrays, although the cache is present and valid and hence the store should have updated the data arrays.

3.4.3 Conditions

To encounter this faulty behaviour the following conditions must be present: The STM to the first line must either be to a WT region of memory or miss in the DCache, while the second line must be present and valid in the DCache. The stores to the first cache line must fill up the WB and cause a memory stall. The memory stall must occur coincident with the first store to the next cache line or the store will not be dropped.

3.4.4 Implications

When the above conditions are met the DCache will drop the store data and the data arrays will not be updated resulting in a corrupted DCache content. If the data later is read, the data returned will be wrong.

3.4.5 Workaround

There are a couple of options. The first one is to not use STM instructions! The second one would be to ensure, that no STM is going to cross a cache line boundary.

3.5 BIU incorrectly handles 8- and 16-bit transfers in big endian (Revision r0p0, r0p1)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #273.

3.5.1 Summary

During any big endian byte or halfword transfer from either the instruction or data BIU, the low order address bits will be inverted. In the case of a byte transfer bit [0] will be inverted, and in the case of a halfword transfer bits [1:0] will be inverted. No issues exist with little endian transfers.

3.5.2 Description

The erratum is observed for any byte or halfword transfers in big endian only for both the instruction and data BIU.

3.5.3 Conditions

Any system which configures the BIGENDINIT bit high during reset of the processor or any system which changes the state of System Control Processor (CP15) register 1 bit 7 (B bit) to be set will encounter the problem under the following conditions:

- any data fetch instruction which performs a load or store or swap byte operation
    - load instructions: LDRB, LDRBT, LDRSB
    - store instructions: STRB, STRBT
    - swap instructions: SWPB
- any data fetch instruction which performs a load or store halfword operation
    - load instructions: LDRH, LDRSH
    - store instructions: STRH
- any instruction fetch which performs a 16-bit request to the AMBA instruction AHB interface. A 16-bit instruction fetch can only occur in a NC area of code with the non-cacheable prefetching engine disabled (CP15 register 15 – Debug Override Register bit 16, DNCP).

3.5.4 Implications

Any slave which is configured or constructed to understand big endian transfers will either load or store the incorrect byte or halfword due to the address bits being incorrectly inverted.

3.5.5 Workaround

A possible software work around is to configure any slave to map itself little endian for both little and big endian transfers, assuming the slave is configurable in software.

A possible hardware work around is to inverted the inverted address outputs, bits [1:0], of the processor on both the instruction and data AHB interface when the processor is configured to be in big endian state. This state can be observed on the CFGBIGEND output pin of the processor. Follow any recommended synchronization which switching the endianess of the processor.

4 CATEGORY 2 ERRATA

4.1 Erroneous Q Flag Value for Enhanced DSP Multiply-Accumulate following Bounced Coprocessor Instructions/Hardware Watchpoints

(Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #263.

4.1.1 Summary

Certain events that occur on an instruction that is followed by a DSP multiply-accumulate instruction that sets the Q flag (`SMLAWx` or `SMLAxy`) will show the Q flag set in the CPSR and SPSR_und upon entering the event handler. These events include bounced (thrown to software via the undefined instruction handler) coprocessor instructions, and watchpoints in hardware debug mode.

4.1.2 Description

If a coprocessor instruction (MCR/MRC/STC/LDC/CDP) is executed with the coprocessor number being 0 through 13, this coprocessor instruction is sent to external coprocessors. If the given coprocessor does not exist or rejects the instruction, the instruction is said to be BOUNCEed. Likewise, a CP14/15 coprocessor instruction can be internally rejected due to permission violations like attempting to run CP15 instructions in user mode or write certain CP14 registers in hardware debug mode.

In these cases, any instructions that are in the pipeline will be flushed and the undefined instruction handler will be entered. However, the flush of the DSP multiply-accumulate instruction is not completed correctly, and the resultant Q-flag value generated by this instruction is updated.

Additionally, if a load or store instruction attempts to read or write from an address that has a watchpoint set on it, and the debug is in hardware (Halt) mode, and a DSP multiply- accumulate instruction follows the watchpointed instruction, the Q-flag will also be set before the watchpoint halts the processor.

The DSP multiply- accumulate instructions affected by this problem are: `SMLABB, SMLABT, SMLATB, SMLATT, SMLAWB`, and `SMLAWT`. The DSP add and subtract instructions are NOT affected.

4.1.3 Conditions

This problem can be encountered whenever there is a back-to-back sequence of these two classes of instructions. This problem may not be exhibited for all rejected CP14 or CP15 instructions.

Additionally, if branch prediction is enabled, the two instructions can be separated by a predicted unconditional or conditional branch and still exhibit the problem if the branch is predicted properly.

4.1.4 Implications

Under most program conditions, a BOUNCEd coprocessor instruction will be either emulated or cause an error condition. In the case of emulation, the coprocessor emulation code must appear transparent to the main program and therefore will save and restore the state of the processor before modifying any registers. Coprocessors cannot directly read the Q flag, so emulation code will not rely on the Q flag state of the initial program. The emulation code will return to the instruction following the coprocessor instruction, in this case the DSP multiply- accumulate instruction. This instruction will be repeated with identical operands, so the resultant Q flag state will be correct after the DSP multiply- accumulate instruction is executed. Under these conditions, the bug is benign, since to the main program code stream will maintain the proper Q flag and the emulation code does not rely on a specific value of the Q flag.

A single exception to this rule is the following code sequence, which contains a move from coprocessor instruction followed by a DSP multiply-accumulate instruction that has the source accumulate value Rn dependent on the coprocessor instruction:

```
MRC   p9, 0, r2, c0, c0, 0
SMLABB r7, r9, r0, r2
```

In this case, the SMLABB instruction will initially erroneously generate the Q flag based on the value of r2 prior to the MRC instruction. The emulation code will update r2 with the correct value, and return to the SMLABB instruction. However, the Q flag is sticky, and if the initial r2 prior to the MRC caused an overflow, and the value of r2 written by the MRC emulation code does not, then the Q flag will remain set, potentially causing a failure in the main program code.

Note This problem is not exhibited if the dependency is in either of the multiply operand source registers Rm or Rs. These are the second or third field of the SMLABB instruction shown above.

For the case of hardware debug watchpoints, the user should be aware that the state of the Q bit read out from the processor may be prematurely set if a watchpoint is hit on a load or store instruction that occurs directly before a DSP multiply-accumulate instruction. This can be determined by inspection of the code stream around the halted program counter address.

4.1.5 Workaround

The workaround is to place at least one instruction, like a NOP, between the coprocessor and the DSP multiply-accumulate instruction.

4.2 Erroneous ETM Branch Phantom Valid with Bounced Coprocessor Instructions (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #254.

4.2.1 Summary

The ETMCORECTL[3] signal may assert erroneously around branches to coprocessor instructions that take an Undefined Instruction trap.

4.2.2 Description

The bug occurs when branch prediction is enabled and a predicted branch next instruction is a coprocessor instruction that takes an Undefined Instruction trap.

When branch prediction is enabled, a predicted branch instruction may be removed from the instruction execution stream, and the condition codes of the predicted branch are folded onto the predicted branch next instruction, and only a single instruction is issued to the processor integer unit. The predicted next instruction is called the branch phantom.

ETMCORECTL[3] is the ARM1026EJ-S processor signal that indicates a branch phantom instruction is valid, and it should only assert once per branch phantom instruction.

When the bug occurs, ETMCORECTL[3] signal is asserted twice. The first assertion of ETMCORECTL[3] signal is correct and indicates a valid branch phantom instruction. The second assertion of the ETMCORECTL[3] signal occurs prior to entry into the Undefined Instruction handler, and is incorrect.

4.2.3 Conditions

This problem can only occur if branch prediction is enabled and a predicted branch next instruction is a coprocessor instruction that takes an Undefined instruction trap.

A coprocessor instruction can take an Undefined Instruction trap due to several reasons; the system control processor (CP15) instructions are executed in User mode, the coprocessor instruction is not implemented, the external coprocessor does not exist, the coprocessor instruction executes an illegal condition, etc.

4.2.4 Implications

There is no functional impact on systems that use the ARM1026EJ-S processor without the ETM10RV.

In embedded systems that use an external ETM10RV connected to the ARM1026EJ-S processor to provide real-time tracing of instructions and data, the most common result of this bug will be tracing of an invalid instruction between a predicted branch instruction and entry into the Undefined Instruction handler.

4.2.5 Workaround

In systems where the ETM10RV is not used, no workaround is required.

In systems where the ETM10RV is connected to the ARM1026EJ-S processor, the condition that caused the coprocessor instruction to take an Undefined Instruction trap must be removed.

4.3 DMA interaction with TCM controller (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #270.

4.3.1 Summary

Control state machine that grants ownership of TCM busses to external DMA controller contains an erroneous transition that can result in unintended grant to DMA. This unintended grant can prevent ARM1026EJ-S-initiated transfers from gaining immediate access to the TCM.

4.3.2 Description

The TCM DMA interface relies on a fixed protocol (state machine) that must be adhered to both within the TCM controller (internal to ARM1026EJ-S) and in the external DMA request engine. This state machine is documented in ARM1026EJ-S Technical Reference Manual [1], section 17.2.5 “DMA interaction with the TCM controller.” An error in the specification and implementation of this state machine in the TCM control logic resulted in an erroneous transition from idle (TCM1) state to DMA-ownership state (DMA) under conditions where the waitstate indicator (DRWAIT, IRWAIT) is inactive. Inclusion of the RWAIT term in the transition equation from TCM1 to DMA is the exact error, it should not factor into this state transition. The conditions that specify remaining in state TCM1 are similarly erroneous.

The consequences of this error depend on the activity of the DMA request engine and external waitstate logic that are part of the partner-built/partner-specific portion of the design (specifically the a10mTCMMem level of hierarchy where TCM RAMs get placed and partners implement their DMA and/or waitstate control logic). The TCM interface ownership machine (TCM vs. DMA) can enter the DMA state erroneously, preventing a core-based TCM transfer from occurring immediately. This is a performance issue. Under certain circumstances of a periodic DMAEN signal, where the inactive period of DMAEN is only a single cycle between subsequent DMA requests, the state machine never recovers from erroneous entry to the DMA state and a pending transfer on the TCM interface is stalled infinitely, deadlocking the processor. This periodic DMAEN generation is an unlikely real-world scenario, as a DMA requestor would back off eventually once the DMA transfer is complete. However, the fact that the TCM state machine can enter the DMA state erroneously under various real-world DMAEN/RWAIT scenarios does impact performance of the TCM interface, potentially delaying TCM transfers unnecessarily.

4.3.3 Conditions

The lockup scenario can only occur with a repeated, strictly periodic RDMAEN signal - particularly where RDMAEN goes low for only a single cycle between separate requests for DMA control of the TCM interface. Erroneous entry into DMA state (performance concern scenario) occurs when RDMAEN=1 & RWAIT=0, irrespective of RCS.

4.3.4 Implications

Potential deadlock scenario where a TCM access never completes, stalling the processor infinitely, under contrived scenario on the TCM DMA request interface. Performance degradation of TCM transfers when switching back and forth between TCM transfers and DMA transfers.

4.3.5 Workaround

In systems where TCMs are unused or where there is no DMA activity on the TCM interface, no workaround is required.

In systems where a DMA request for the TCM interface does occur, the RDMAEN indicator must remain low for at least 2 cycles before being reasserted.

4.3.6 Corrective Action

The performance issues associated with this erratum have been corrected in ARM1026EJ-S r0p1. The ARM1026EJ-S Technical Reference Manual (section 17.2.5) has been amended (Revision r0p1) to clarify the requirements of the external, partner-built DMA engine. Partner-built DMA engines must adhere to the defined (state-machine) protocol to conclude when it is safe for the DMA engine to take ownership of the TCM RAM interface. A specification addition now mandates that in systems where a DMA request for the TCM interface does occur, the RDMAEN indicator must remain low for at least 2 cycles before being reasserted.

4.4 Unpredictable behaviour when in Debug state during DBGTAPSM transitions (Revision r0p0, r0p1)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #272.

4.4.1 Summary

In debug state, the processor may have unpredictable behaviour when the DBGTAP state machine transitions through certain states.

4.4.2 Description

When the processor has entered halt mode debug state and the external debugger has control over the processor, the JTAG DBGTAP state machine (DBGTAPSM) may progress through different states.

When the DBGTAP state machine transitions through different states, example from Update-IR to Select-DR-Scan, the reset control to the Debug Status and Control Register (DSCR) bits may glitch.

This may lead to unpredictable behaviour in the processor, such as unexpected clearing of the DSCR bits, incorrect Comms channel operation, and the processor deadlock in debug state.

This bug can only occur when the DBGTAP state machine undergoes one of the following two transitions:

Capture-IR (4’b1110) to Exit1-IR (4’b1001)

Update-IR (4’b1101) to Select-DR-Scan (4’b0111)

4.4.3 Conditions

The problem only occurs when the processor is halted in debug state, the DBGTAP state machine is transitioning through certain specific states, and the DBGTAPSM state values do not all get updated or switched at the same time.

The likelihood of the problem occurring is rare, as the glitch in the DSCR reset caused by the DBGTAPSM transitioning must meet the minimum pulse width (tminpwl or tminpwh) constraint, which is typically specified to be higher than the Clock-to-Q propagation delay.

It is also important to note that not all implementations of the ARM1026EJ-S processor will exhibit this behaviour since the exact times at which the DBGTAPSM state values transition relative to each other vary from one silicon implementation to another, and from one silicon process to another.

4.4.4 Implications

Nonreal-time debugging of the processor is affected. The external debugger may not be able to examine and alter the processor state correctly.

This bug does not affect normal functional use or operation of the processor, i.e. when the processor is in monitor mode.

4.4.5 Workaround

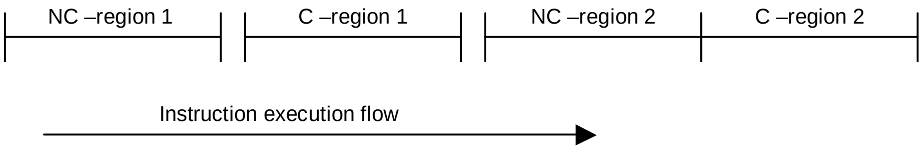
There is no practical workaround for this erratum.

4.5 False entry into the Instruction Prefetch Abort handler (Revision r0p0, r0p1)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #275.

4.5.1 Summary

During execution of code, an instruction can falsely trigger entry into the prefetch abort handler. The executed instructions have to reside in both cacheable and non-cacheable pages in memory as shown in the diagram below:



With the non-cacheable prefetching engine enabled (coprocessor 15 register “Debug Override” bit 16 = 0), an abort when filling from “NC-region1” is returned during a WRAP4 burst for the last non-critical word. This abort is held in a buffer and is only cleared on a sub-sequent fill to the buffer of the non-cacheable prefetching engine. This abort from “NC-region 1” is falsely returned when the very next fill to the buffer occurs to “NC-region 2”. Hence, the instruction prefetch abort handler is falsely entered.

4.5.2 Description

In order to describe the issue, the instruction execution flow must be understood for the four memory regions defined in the summary section.

A fill occurs in “NC-region 1”. This fill must contain a branch which is executed by the processor. The instructions that have been prefetched following the branch during the same fill that the branch was fetched must abort. None of the aborted instructions ever execute due to the branch flushing the instruction pipeline; however, the non-cacheable prefetching engine buffer will hold the abort information for the instructions in it's buffer. The executed branch instruction changes program flow to "C-region 1". "C-region 1" then branches to the last 32-bit word in "NC-region 2", so the very next access is to "C-region 2". The data of "NC-region 2" is returned to the processor; however, the next access to "C-region 2" falsely aborts which causes the processor to falsely enter the prefetch abort handler.

4.5.3 Conditions

Any system which configures the state of System Control Processor (CP15) register 15 bit 16 (DNCP bit) of the "Debug Override" register to be clear will encounter the problem under the conditions described in section 4.5.1and 4.5.2.

4.5.4 Implications

False entry into the prefetch abort handler will occur.

4.5.5 Workaround

Set the System Control Processor (CP15) register 15 bit 16 (DNCP bit) of the "Debug Override" register to a logic 1 to disable the non-cacheable prefetching engine.

5 CATEGORY 3 ERRATA

5.1 Unpredictable Behaviour of CP15 r15 MMU main TLB write *test operations* (Revision r0p0)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #255.

5.1.1 Summary

CP15 r15 MMU test operations (used to directly write entries to the main TLB under software control) have unpredictable behaviour, specifically in terms of which way is actually written. This bug does *not* affect normal functional use/operation of the MMU. It involves only special CP15 *test operations* that a user may choose to execute under special circumstances to directly manipulate TLB entries. These CP15 r15 MMU main TLB write test operations are only allowed if ARM1026EJ-S is configured in MMU mode (MMUnMPU macrocell pin set to logic 1).

5.1.2 Description

CP15 r15 MMU test operations involve direct manipulation of main TLB entries under software control. For these operations, the CP15 r15 Debug & Test Address Register is used to specify which way and indexed entry is to be written. An error in the design makes the way specification unpredictable on test operation writes to the main TLB. This error does not affect test operation reads of the main TLB. Most importantly, this error does not affect normal functional use/operation of the MMU.

5.1.3 Conditions

Problem occurs only if CP15 r15 MMU test operations are used to manipulate (write) specific TLB entries. This is done as a 4-instruction sequence involving:

(1) MCR p15,0,Rd,c15,c1,0 (write Debug/Test Address reg)

(2a) MCR p15,4,Ra,c15,c3,0 (write TAG main (2-way) TLB storage reg)

(2b) MCR p15,4,Rb,c15,c5,0 (write PA/PROT main (2-way) TLB storage reg)

(3) MCR p15,4,Rc,c15,c7,0 (transfer 2-way storage into RAM)

The transfer of TAG and PA/PROT information from temporary storage registers into RAM does not properly use bit[31] of the Debug/Test Address register to specify which way the entry should be written to, so it is unpredictable as to whether the information will land in way 1 or way 0 in the TLB RAM.

5.1.4 Implications

The only conceivable use for the main TLB test operations involves a user who wants to completely bypass the hardware tablewalking scheme implemented in ARM1026EJ-S and instead implement a “soft” (software) TLB by performing the virtual to physical address translation via a specific (user-written) software routine. This routine would have to be invoked by setting the CP15 Debug Override register’s ADTM and/or AITM bits (Abort on D/I TLB miss, MRC/MCR p15, 0, Rd, c15, c0, 0), and interrogating the FSRs on aborts to assess if an abort was the result of a TLB miss. The Debug Override register ADTM and AITM features are completely functional, only the main TLB test operation writes are affected by this erratum.

5.1.5 Workaround

No workaround required for normal functional MMU behaviour. Using Debug Override register functionality in concert with CP15 r15 MMU test operations to build a “soft” (software) TLB to bypass normal ARM1026EJ-S hardware tablewalking functionality is not supported in this release due to this error.

5.2 X-propagation during Gate-level simulations (Revision r0p0, r0p1)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #271.

5.2.1 Summary

Running a functional test or CRF test vector on the gate-level netlist may fail or miscompare.

5.2.2 Description

In the ARM1026EJ-S some, but not all of the flops are resetable. This is fine from a functional point of view, and when running functional assembly tests on the Register Transfer Level (RTL) representation of the design.

However, it causes problems when running the functional tests or CRF test vectors on the gate-level netlist. A state point from a non-resetable flop has an "x' state during reset in simulation. During gate-level simulations, an "x" state for an initial condition can propagate causing the functional test to fail or the CRF test vector to mismatch.

In silicon, an "x" state does not exist and all state points resolve to a true or false, i.e. "1" or "0", value.

5.2.3 Conditions

The problem occurs only when running functional assembly tests or CRF test vectors on the ARM1026EJ-S gate-level netlist. It does not occur when running the tests on the RTL model.

5.2.4 Implications

This problem does not occur in silicon, and does not affect the normal functional use or operation of the processor.

5.2.5 Workaround

Constraining the simulator to use only 2 states, "1" and "0", instead of the usual 4 states, "z", "x", "0", and "1", will disallow all "x" states during gate-level simulation, and allow the functional tests to complete without fail. As an example, the Synopsys VCS simulator has a +2state option to enable 2-state simulation mode.

Mismatches that occur when running CRF test vectors on the gate-level netlist due to the x-propagation problem caused by non-resetable flops can be ignored. However, there is no easy and quick discernable method to distinguish between valid mismatches and mismatches due to x-propagation.

5.3 MBIST Datalog Uninitialized (Revision r0p0, r0p1, r0p2)

ARM Bug tracking database entry: ARM1026EJ-S bugscougar #276.

5.3.1 Summary

Reading of MBIST datalogs to determine failing array may include random data on "passing" arrays, making identification of defective array difficult. Primary Pass/Fail functions of MBIST tests are not impaired.

5.3.2 Description

In the ARM1026EJ-S core, DFT MBIST logic is included at the array interfaces for use with an optional ARM MBIST controller. Embedded within this logic is the ability to datalog and read the first failing location of arrays after test completion. This is used to identify defective arrays for possible MBIST bitmapping mode and for soft binning of failure arrays during production test.

A MBIST validation hole has been discovered, as datalog registers are not actually reset. This results in random data in datalog registers. Once a test has completed, reading the datalog registers may result in random data for passing arrays. For example, if one of ten arrays fail during a test, datalog reads of all ten arrays will contain failure data only one of which is valid and correct. Bitmap mode is not affected since datalog reads occur on a per array basis only after failure has been detected.

5.3.3 Conditions

The problem occurs only with DFT MBIST tests that perform datalog dumps after test completion when using the ARM MBIST controller and testbench.

5.3.4 Implications

This problem does not affect the normal functional use or operation of the processor and does not affect MBIST pass/fail validity. Failure analysis of MBIST arrays is impacted.

5.3.5 Workaround

A workaround exists by performing a datalog dump of all registers prior to test execution while ignoring datalog outputs. Execution of a datalog dump results in back filled zeroes into the datalog registers, effectively resetting the values. To workaround this datalog problem when generating test vectors, the following code can be added to the end of the “ResetMBIST” task found in “bist_tasks.v” of the “tbenchbist” directory.

```
reg [15:0] array_reg;
reg [5:0]  ctl_reg;

      // ctl_reg 6'b001111 selects data dispatch scanout
      force ctl_reg = 6'b001111;
      force MBISTRESULT[2:0] = 3'bxxx;  force Monitor=1'b0;// avoiding tester pattern boundaries
      force MBISTDATAIN=0;

      @(negedge CLK);

      force array_reg = 16'b0000000000000001; // IRam
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=86;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000000000010; // ITag
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=43;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000000000100; // IVal
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=38;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000000001000; // DRam
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=86;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000000010000; // DTag
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=43;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000000100000; // DVal
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=38;
      bmapDLOG(cnt,1'b0,LOG);

      force array_reg = 16'b0000000001000000; // DDty
      loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
      cnt=24;
      bmapDLOG(cnt,1'b0,LOG);
```

```
force array_reg = 16'b0000000010000000; // Mmu
loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
cnt=123;
bmapDLOG(cnt,1'b0,LOG);

force array_reg = 16'b0000000100000000; // ITcm
loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
cnt=88;
bmapDLOG(cnt,1'b0,LOG);

force array_reg = 16'b0000001000000000; // DTcm
loadInstr(array_reg,4'h0,4'h0,4'h0,ctl_reg,6'b000000);
cnt=88;
bmapDLOG(cnt,1'b0,LOG);
```