ARM7TDMI-S Errata List

IP Products Division

Document number: FR002-PRDC-002719 7.0
Date of Issue: 20th March 2009

Abstract

This document describes the known errata of releases rev2, rev3, rev3a, rev4p0-p2 and rev4p3 of the ARM7TDMI-S design.

Keywords

ARM7, ARM7TDMI, ARM7TDMI-S, errata

This is a working document throughout the product lifecycle and, as such, the content may be modified as new information is uncovered.

Contents

1 ABOUT THIS DOCUMENT 5

1.1 References 5

1.2 Scope 5

1.3 Terms and Abbreviations 5

2 CATEGORISATION OF ERRATA 6

2.1 Errata Summary 6

3 CATEGORY 1 ERRATA 8

3.1 Undefined exceptions falsely taken 8
3.1.1 Summary 8
3.1.2 Conditions 8
3.1.3 Workaround 9
3.1.4 Validation Coverage 9
3.1.5 Corrective Action 9
3.1.6 Impacted revisions 9

4 CATEGORY 2 ERRATA 10

4.1 Debug Request coincident with Exceptions 10
4.1.1 Summary 10
4.1.2 Conditions 10
4.1.3 Implications 10
4.1.4 Workarounds 11
4.1.5 Impacted revisions 11

4.2 Breakpoints following Exceptions or Watchpointed Store 12
4.2.1 Summary 12
4.2.2 Conditions 12
4.2.3 Implications 12
4.2.4 Workarounds 13
4.2.5 Impacted revisions 13

4.3 Breakpoints following Multi-cycle Instructions 14
4.3.1 Summary 14
4.3.2 Conditions 14
4.3.3 Implications 14
4.3.4 Workarounds 14
4.3.5 Impacted revisions 14

4.4 Watchpoint followed by Exceptions 15
4.4.1 Summary 15
4.4.2 Conditions 15
4.4.3 Implications 15
4.4.4 Workarounds 15
4.4.5 Impacted revisions 15

4.5 Debug Entry coincident with Debug Request 16
4.5.1 Summary 16
4.5.2 Conditions 16
4.5.3 Implications 16
4.5.4 Workarounds 16
4.5.5 Impacted revisions 16

4.6 Abort link register not properly updated on data aborts in Thumb state 17
4.6.1 Summary 17
4.6.2 Conditions 17
4.6.3 Implications 17
4.6.4 Workaround 17
4.6.5 Impacted revisions 17

4.7 PC is load with an incorrect value when DBGRQ is asserted asynchronously during the execution of a modify PC instruction. 18
4.7.1 Summary 18
4.7.2 Conditions 18
4.7.3 Implication 18
4.7.4 Workaround 18
4.7.5 Corrective action 19
4.7.6 Impacted revisions 19

5 CATEGORY 3 ERRATA 20

5.1 Consecutive Breakpoints 20
5.1.1 Summary 20
5.1.2 Conditions 20
5.1.3 Workaround 21
5.1.4 Validation Coverage 21
5.1.5 Corrective Action 22
5.1.6 Impacted revisions 22

5.2 Incorrect behaviour when an interrupt (FIQ/IRQ) occurs in Debug state 23
5.2.1 Summary 23
5.2.2 Description 23
5.2.3 Implication 23
5.2.4 Corrective Action 23
5.2.5 Impacted revisions 23

5.3 Incorrect value of EmbeddedICE-RT version number read through JTAG interface 24
5.3.1 Summary 24
5.3.2 Description 24
5.3.3 Implication 24
5.3.4 Workarounds 24
5.3.5 Corrective Action 24
5.3.6 Impacted revisions 24

5.4 Incorrect EIS log file when the core is in Thumb state 25
5.4.1 Summary 25
5.4.2 Description 25
5.4.3 Implication 26
5.4.4 Corrective Action 26
5.4.5 Impacted revisions 28

5.5 SWI instruction and Prefetch Abort not properly handled if they follow a condition code failed Undefined Instruction 29
5.5.1 Summary 29
5.5.2 Description 29
5.5.3 Implication 29
5.5.4 Workaround 29
5.5.5 Corrective action 29
5.5.6 Impacted revisions 29

6 SYSTEM ERRATA 30

6.1 LDC/LDCL/STC/STCL instructions incorrectly decoded when non-indexed addressing mode is used 30
6.1.1 Conditions 30
6.1.2 Implications 30
6.1.3 Workaround 30
6.1.4 Impacted revisions 30

6.2 Sequential MRC instructions may not execute correctly 31
6.2.1 Conditions 31
6.2.2 Implications 32
6.2.3 Workaround 32
6.2.4 Impacted revisions 32

1 ABOUT THIS DOCUMENT

1.1 References

This document refers to the following documents.

Ref.	Document No	Author(s)	Title
1	ARM DDI 0234 B	ARM	ARM7TDMI-S (R4p3) Technical Reference Manual

1.2 Scope

This document describes all the errata discovered in the different implementations of the ARM7TDMI-S, 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
- the status of corrective action.

1.3 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AVS	ARM Validation Suite

2 CATEGORISATION 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 behaviour 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.

Implementation Errata that are of particular interest to those implementing the product and that have no software implications

System Errata or possible issues that have system implications and therefore should be considered by system designers

Testchip Errata that are identified with components in the Testchip RTL that make up the validation testbench such as the AHB components.

2.1 Errata Summary

The errata associated with this product are categorised in the following way. Numbers in brackets after the errata description indicate the order in which the errata were found chronologically.

		Rev2	Rev2a	Rev3	Rev3a	Rev4p0-p2	Rev4p3
Category 1	[2] Undefined exceptions falsely taken	X	Ok	X	Ok	Ok	Ok
Category 2	[3] Debug Request coincident with Exceptions	X	X	X	X	Ok	Ok
	[4] Breakpoints following Exceptions or Watchpointed store	X	X	X	X	Ok	Ok
	[5] Breakpoints following Multi-cycle Instructions	X	X	X	X	Ok	Ok
	[6] Watchpoint followed by Exceptions	X	X	X	X	Ok	Ok
	[7] Debug Entry coincident with Debug Request	X	X	X	X	Ok	Ok
	[12] Abort link register not properly updated on data aborts in Thumb state	X	X	X	X	X	Ok
	[13] PC is load with an incorrect value when DBGRQ is asserted asynchronously during the execution of a modify PC instruction.	X	X	X	X	X	X
Category 3	[1] Consecutive Breakpoints	X	X	Ok	Ok	Ok	Ok
	[8] Incorrect behaviour when an Interrupt occurs in Debug state	X	X	X	X	X	Ok
	[9] Incorrect value of EmbeddedICE-RT version number read through JTAG interface	Ok	Ok	Ok	Ok	X	Ok

	[10] Incorrect EIS log file when the core is in Thumb state	N/A	N/A	N/A	N/A	X	Ok
	[11] SWI Instruction and Prefetch Abort not properly handled if they follow a condition code failed Undefined Instruction	X	X	X	X	X	X
System	[14] LDC/LDCL/STC/STCL instructions incorrectly decoded when non-indexed addressing mode is used	X	X	X	X	X	X
	[15] Sequenial MRC instructions may not execute correctly	X	X	X	X	X	X

3 CATEGORY 1 ERRATA

3.1 Undefined exceptions falsely taken

3.1.1 Summary

The processor can falsely take the Undefined exception.

3.1.2 Conditions

A combination of three conditions is required.

1) An ALU operation which modifies the PC, or a BX, which will result in the processor being in Thumb State.
2) The second instruction in the branch shadow decodes as a Thumb undefined instruction.
3) There is an abort on the destination instruction, or the destination instruction is a Thumb SWI.

Only if **all three conditions are satisfied** can the problem occur. If all these conditions are met the processor will take the Undefined exception trap, instead of the SWI or Prefetch Abort exceptions.

Condition 1

The problem can only occur after a BX (Branch Exchange), or an arithmetic operation which modifies the PC, such that the processor will be in Thumb State following this instruction.

In ARM State this could be:

- An arithmetic operation with destination PC, and the 'S' bit set, such as:
  `MOVS PC, R14` or
  `SUBS PC, R14, #4`
  in which the stored T bit is set, such that the processor will transition into Thumb State.
- A BX which will transition into Thumb State.

In Thumb State this could be:

- An arithmetic operation with destination PC such as:
  `MOV PC, R14`
- A BX, which will not transition into ARM State, but will remain in Thumb State.

Condition 2 (ARM to Thumb State)

In the case of a transition from ARM State to Thumb State the next two ARM instructions are loaded into the pipe. The problem can occur if the second instruction can be decoded as a Thumb undefined instruction. In Little Endian mode the Thumb decoder will decode the lower half (bits 15:0) of the instruction, in Big Endian mode, the upper (bits 31:16) of the instruction. If this bit pattern is an undefined Thumb instruction then the problem can occur if condition 3 is also met.

Condition 2 (Thumb to Thumb State)

In the case of a transition from Thumb State to Thumb State (a Thumb MOV PC, LR, or a BX which doesn't change state) then the two Thumb instructions following the implicit branch are loaded into the pipe. If the second of these is an undefined Thumb instruction then the problem can occur if condition 3 is also met.

Condition 3

Finally the target instruction of the branch must be a SWI, or must be aborted when it is fetched from memory (Prefetch Abort).

If all three of these conditions are met the Processor will take the Undefined exception rather than the SWI or Prefetch Abort exceptions.

3.1.3 Workaround

There is no workaround for this erratum. An update to the RTL and some associated deliverables will be necessary. This will be made available to all partners as a maintenance update.

3.1.4 Validation Coverage

The existing validation suite does not introduce the specific conditions needed to show this erratum. The maintenance update will include additional tests that will fully explore the device behaviour under these conditions.

3.1.5 Corrective Action

For most existing implementations, it is likely that the above combination of conditions will never occur. However, application code can be scanned for code sequences that might result in this issue and code can be patched if required.

3.1.6 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev3

4 CATEGORY 2 ERRATA

4.1 Debug Request coincident with Exceptions

4.1.1 Summary

If **DBGRQ** or scan chain created debug request is asserted while an exception is being processed then the processor may not restart execution from the correct point after exiting Debug state.

4.1.2 Conditions

This erratum is triggered by two simultaneous conditions:

1) Debug Request is asserted and
2) An instruction that causes an exception is executing and the exception is one of:
    1) Undefined Instruction
    2) Bounced Coprocessor Instruction
    3) Prefetch Abort
    4) Data Abort.

If these conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may exit from debug and begin execution from an incorrect address.

4.1.3 Implications

For each exception the following implications are expected given the above conditions:

4.1.3.1 Undefined Instructions

On entry into debug state the undefined instruction has been recognised. However, the PC does not advance by the expected amount as debug entry proceeds. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour may result.

4.1.3.2 Bounced Coprocessor Instructions

This form of this erratum behaves in the same manner as the Undefined Instruction form.

4.1.3.3 Prefetch Aborts

This form of this erratum behaves in the same manner as the Undefined Instruction form.

4.1.3.4 Data Aborts

For store instructions, debug entry and exit occurs such that upon return, the correct instruction is executed, however, the recognition of the abort does not happen prior to debug entry and so the abort is missed. This will probably result in unpredictable behaviour as the data that should have eventually been stored (assuming the abort was recoverable) will be incorrect.

For load instructions, debug entry occurs for a single cycle, then occurs again as the abort is recognised. This multiple entry into debug affects the value of the PC such that the standard return from debug results in the wrong instruction being executed on return and thus unintended or unpredictable device behaviour may result.

4.1.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action.

4.1.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a

4.2 Breakpoints following Exceptions or Watchpointed Store

4.2.1 Summary

If a breakpoint occurs whilst an exception is being processed or after a watchpointed store instruction then the processor may not restart execution from the correct point after exiting debug state.

4.2.2 Conditions

There are two scenarios that can cause this erratum.

The conditions for the first scenario are:

An instruction that causes an exception is executing, the exception is one of:

1) Undefined Instruction
2) Bounced Coprocessor Instruction
3) Data Abort caused by a store instruction

and the following instruction causes a breakpoint to occur.

The conditions for the second scenario are:

1) A store instruction that causes a watchpoint has executed;
2) and the second following instruction causes a breakpoint to occur.

If these conditions are met then incorrect debug entry may occur, in a way that deviates from the defined manner, which may result in the device returning to execute instructions at the wrong address.

4.2.3 Implications

For each exception the following implications are expected given the above conditions:

4.2.3.1 Undefined Instructions

In this form the breakpointed instruction causes premature entry to debug. The expected behaviour is to take the undefined instruction exception and only after returning from the handler should the breakpoint cause debug entry. When the premature debug entry occurs the device has recognised the undefined instruction, but the PC does not advance by the expected amount. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour will result.

4.2.3.2 Bounced Coprocessor Instructions

This form of this erratum behaves in the same manner as the Undefined Instruction form.

4.2.3.3 Data Aborts caused by a Store Instruction

This case is specific to store instructions. In this form debug entry occurs for a single cycle, then occurs again as the abort is recognised. This multiple entry into debug affects the value of the PC such that the standard return from debug results in the wrong instruction being executed upon return and thus unintended or unpredictable device behaviour may result.

4.2.3.4 Watchpoint caused by a Store Instruction

This case is specific to store instructions. In this form debug entry occurs due to the watchpoint, but the PC does not advance by the expected amount. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour results.

4.2.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action.

4.2.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a

4.3 Breakpoints following Multi-cycle Instructions

4.3.1 Summary

If a multi-cycle instruction is followed by two breakpointed instructions then the processor may not restart execution from the correct point after exiting debug state.

4.3.2 Conditions

1) A non-branching multicycle instruction has executed;
2) and execution of the following two instructions cause breakpoints to occur.

Note: Examples of non-branching multicycle instructions are Data Processing with Register Shift; Multiplies; Load or Store instructions.

If these conditions are met then incorrect debug entry may occur, in a way that deviates from the defined manner. This may result in the device executing instructions at the wrong address upon return.

4.3.3 Implications

In this erratum the two successive breakpoints in the pipeline affect the debug entry such that on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour results.

4.3.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action.

4.3.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a

4.4 Watchpoint followed by Exceptions

4.4.1 Summary

If an exception occurs while a watchpoint is being processed then the processor may not restart execution from the correct point after exiting debug state.

4.4.2 Conditions

There are three scenarios in which this erratum occurs:

1) A load instruction is executed, triggers a watchpoint and the following instruction causes a prefetch abort or
2) A store instruction is executed, triggers a watchpoint and the second following instruction causes a prefetch or abort
3) An interrupt (FIQ or IRQ) occurs on the final cycle of a load instruction that triggers a watchpoint.

If any of these conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and begin execution from the incorrect address.

4.4.3 Implications

For each exception the following implications are expected given the above conditions:

4.4.3.1 Prefetch Aborts

In this form, the Prefetch Abort is recognised prematurely, ie. before debug entry occurs. The PC is therefore not advanced by the expected amount as debug entry proceeds. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour may result.

4.4.3.2 Interrupts

In this form, the Interrupt causes the watchpointing load instruction to postpone for two further cycles after its completion. The PC is therefore not advanced by the expected amount as debug entry proceeds. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour may result.

4.4.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action.

4.4.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a

4.5 Debug Entry coincident with Debug Request

4.5.1 Summary

If **DBGRQ** or scan chain created debug request is asserted as debug entry is occurring then incorrect operation will result.

4.5.2 Conditions

There are two scenarios that can cause this erratum.

1) Debug Request is asserted during debug entry following a breakpointed instruction or
2) Debug Request is asserted whilst an instruction that causes a watchpoint is executing.

If either of these conditions is met then the debug entry mechanism may fail to behave in the defined manner and the device may return from debug and execute from the incorrect address.

4.5.3 Implications

For each exception the following implications are expected given the above conditions:

4.5.3.1 Breakpoints

In this form, debug entry proceeds as if for a breakpoint, but from a debugger's perspective it may appear that debug entry was due to Debug Request since the debugger has insufficient information to be certain of the method of debug entry. Correspondingly, on return from debug the standard return address calculation may produce an incorrect address and thus unintended or unpredictable device behaviour may result.

4.5.3.2 Watchpoints

In this form, if the watchpoint was due to a store instruction, then debug entry proceeds as if for Debug Request. The debugger can see conflicting information since the Debug Request bit in the **Debug Status Register** indicates entry due to Debug Request and bit 32 of Scan Chain 1 indicates that debug entry was due to a watchpoint. Returning from debug as if due to Debug Request will result in correct device behaviour, however the occurrence of the watchpoint will have been lost. This behaviour is unintended.

If the watchpoint was due to a load instruction, then debug entry proceeds as if for a watchpoint. The debugger can again see conflicting information. Returning from debug as if due to a watchpoint will result in correct device behaviour. However, returning from debug as if due to Debug Request would result in unintended or unpredictable device behaviour.

4.5.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action.

4.5.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a

4.6 Abort link register not properly updated on data aborts in Thumb state

4.6.1 Summary

If the processor is in Thumb state and executing the code sequence STR, STMIA or PUSH followed by a PC-relative load, and the STR, STMIA or PUSH is aborted, the PC is saved to the abort link register in only word resolution, instead of half-word resolution.

4.6.2 Conditions

The processor must be in Thumb state, and the following sequence must occur:

```
<any instruction>

<STR, STMIA, PUSH>    <---- data abort on this instruction

LDR rn, [pc,#offset]
```

In this case the PC is saved to the link register R14_abt in only word resolution, not half-word resolution. The effect is that the link register holds an address that could be #2 less than it should be, so any abort handler could return to one instruction earlier than intended.

4.6.3 Implications

In a system that does not use Thumb state, there will be no problem.

In a system that uses Thumb state but does not use data aborts, or does not try to use data aborts in a recoverable manner, there will be no problem.

In a system that uses Thumb state, and uses data aborts in a recoverable manner, such as in a demand paging environment, where the STR, STMIA or PUSH aborts for paging reasons, the code will patch up the MMU and re-execute the STR, STMIA or PUSH. What matters in this case is the instruction preceding the STR etc, and whether this can be re-executed harmlessly. As this cannot be predicted with certainty, it is likely to be a problem.

4.6.4 Workaround

The workaround is to ensure that a STR, STMIA or PUSH cannot precede a PC-relative load. One method for this is to add a NOP before any PC-relative load instruction. However this is not something currently supported by our software tools, and would have to be done manually.

4.6.5 Impacted revisions

This erratum impacts all revisions of ARM7TDMI-S up to and including r4p2, and also the products which use this processor - ARM720T r4p0 to r4p2 and the SC100 rev0, rev0 rel2 and r1p0.

4.7 PC is load with an incorrect value when DBGRQ is asserted asynchronously during the execution of a modify PC instruction.

4.7.1 Summary

The Core Debug Logic in the ARM7TDMI-S r4 causes the PC to be loaded with an incorrect value under certain conditions.

4.7.2 Conditions

If the CPU is executing a data processing instruction which modifies the PC (MOV PC, LDR PC etc) and DBGRQ is asserted asynchronously, at a particular point during the 3-cycles it takes to execute this instruction, the PC may be loaded with an unexpected value.

4.7.3 Implication

If the sole use of DBGRQ is for the debugger to asynchronously stop the processor, then it is very unlikely that this erratum will be seen, due to the relative infrequency of this event.

All other debug features are unaffected by this erratum.

The effect of this erratum is to potentially cause incorrect operation of a program, by restarting a program at the wrong address.

4.7.4 Workaround

It is very unlikely that this erratum will be seen when DBGRQ is asserted by the debugger since it is a very infrequent event, and hence no workaround will be required.

However if required for users of RVD the following workaround to stop the processor is possible, which avoids the use of DBGRQ. When debug entry is desired the user can put the core into the debug state safely by manually creating a hardware breakpoint. This is performed by entering the following values into the watchpoint registers:

```
WPn control mask = 0xFFFFFEF7  (Opcode fetch)
WPn control value = 0x00000100 (Enable this watchpoint unit)
WPn address mask = 0xFFFFFFFF  (Break on any address)
WPn data mask = 0xFFFFFFFF     (Break on any data value)

(Where WPn = Watchpoint unit 'n'; n being 0 or 1)
```

This causes the CPU to break on any instruction at any address. To continue execution the user must disable the watchpoint and select ‘run’. Further ‘breaks’ can be performed by enabling the watchpoint via the debugger. (The watchpoint unit does not need reprogramming). The previous contents of the watchpoint can be restored once debug state has been entered if desired by the user.

Note that this workaround is not applicable for AXD users. There is no workaround for users implementing DBGRQ as an asynchronous debug interrupt source.

4.7.5 Corrective action

No correction is planned for this erratum.

4.7.6 Impacted revisions

This erratum impacts the following revision of the product: rev4p0-p3* *Note previous revisions have not been investigated for this defect.*

5 CATEGORY 3 ERRATA

5.1 Consecutive Breakpoints

5.1.1 Summary

If a single EmbeddedICE comparator is required to flag a watchpoint or breakpoint in successive cycles, the second is not flagged correctly.

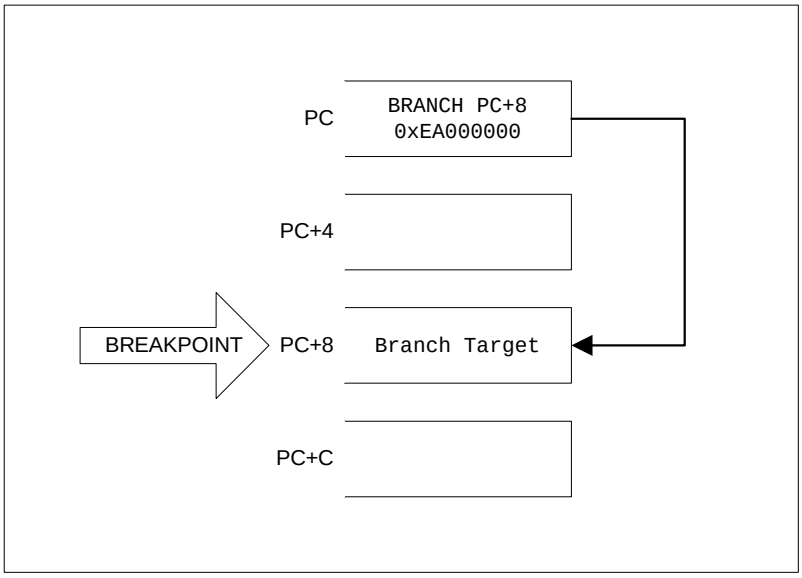
5.1.2 Conditions

This can be encountered in two cases:

1) Branch to PC+8

This case only affects the Branches to pc+8 in ARM State, and Branches to pc+4 in Thumb State. Other branch instructions are unaffected. The problem will only be encountered if the Branch is taken. If the branch is not taken, the breakpoint will be taken correctly. This branch could either be caused by an explicit branch instruction, or an instruction, which directly changes the PC, such as MOV PC, or LDR PC.

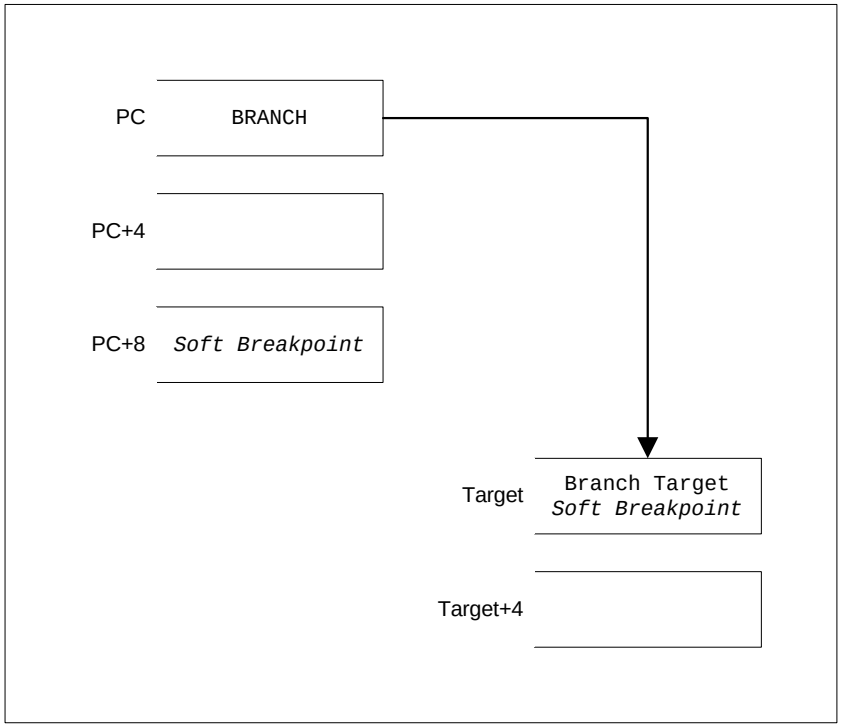
If a breakpoint is set on the destination of a branch to pc+8 (or pc+4 in Thumb State) the breakpoint will be ignored by the ARM7TDMI-S. This is due to the breakpointed instruction being fetched twice in consecutive cycles. The first is prefetch beyond the branch, and is discarded. The second is to re-fill the pipeline and is executed. The second breakpoint is not flagged correctly and the breakpoint is ignored. See diagram below.



This case can be encountered either if the user sets a breakpoint on the branch target, or if the user is single stepping through the case of a branch to pc+8.

2) Soft Breakpoints

If multiple soft breakpoints are set in the memory, this condition can also be encountered. The case occurs where a soft breakpoint is set 2 instructions after a branch, and another soft breakpoint is set on the branch target. In this case the soft breakpoint value will be executed and the processor will not be stopped. See the following diagram.



With the current tools release users should be aware of this issue. If they suspect that a breakpoint has not been taken, they should set a breakpoint on the following instruction instead. When single stepping users should be aware that a branch to pc+8 cannot be single stepped, and place a breakpoint on the instruction following the branch target.

5.1.3 Workaround

'Soft-breakpoints' are patterns that the debug agent plants in memory to allow the debugger to gain control when the instruction reaches the execute stage of the CPU pipeline. In a debug monitor such breakpoints would normally be planted as UNDEFINED instruction patterns which cause the CPU to take the Undefined instruction trap at Vector 0x04.

A workaround has been developed for the ARM Multi-ICE driver which also uses UNDEFINED instruction patterns and a hardware breakpoint is set on the Undefined Instruction trap vector. A special variant of Multi-ICE DLL then detects this 'breakpoint entry' condition, restores the PC to the breakpointed instruction location and then passes control to the normal debugger interface. The only side effect is the use of the private Undefined mode link register and processor status flags (R14_undef, SPRS_undef). The limitation is that one cannot single-step in UNDEF mode (see NOTE below).

Subject to the limitation below, this workaround will allow the user to set soft breakpoints, and single step though code without encountering this problem.

NOTE: this workaround is only suitable where undefined instruction trap handling is NOT being used (i.e. emulation of coprocessor instructions that are not handled in hardware).

5.1.4 Validation Coverage

A validation test has been written to cover this case. It tests:

- Breakpoint on Branch target and in prefetch after branch for small branches
- Watchpoints (soft breakpoints) set near branches, and branch targets

In each case the test strobes the breakpoint across the branch and branch target.

5.1.5 Corrective Action

A software workaround is available for this release.

5.1.6 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a

5.2 Incorrect behaviour when an interrupt (FIQ/IRQ) occurs in Debug state

5.2.1 Summary

Interrupts are not masked correctly when the core is in debug state, so that when the core attempts to execute 'at-speed' an instruction that lasts more then 6 cycles (LDM, STM) whilst an interrupt is asserted, this instruction cannot finish properly.

5.2.2 Description

When the following conditions are reached:

- The core is in debug state,
- An interrupt (FIQ or IRQ) is asserted at any time when the core is in debug state and is still asserted when the core goes into 'at speed' state.
- The core starts to execute at speed a LDM or STM that lasts more then 6 cycles (instruction loaded via JTAG),

The core should ignore the interrupt and go back to halt mode, once the LDM or STM is finished. However, the core starts to execute the LDM instruction, and after 6 data transfers, it comes back to halt debug mode without finishing the data transfers.

The *core module* ignores properly the interrupt request so that it does not enter interrupt mode, but the interrupt is not masked correctly in the *debug module* and corrupts the execution of at least 7-cycles instructions. For shorter instructions, the problem is not observed.

5.2.3 Implication

When the debugger needs to download some code by using LDM/STM instructions, the code might not be properly loaded if an interrupt occurs. Note that all ARM debug tools disable interrupts during code download – the errata will never be seen if using ARM tools.

5.2.4 Corrective Action

Two possible corrective actions are possible:

5.2.4.1 Hardware correction

The A7S hardware can be modified by masking the interrupts in debug module.

A new signal that indicates the core is at-speed state masks the interrupts, so that any executed instruction finishes properly before the core re-enters debug halt mode.

5.2.4.2 Software correction

A software workaround may be possible by disabling the interrupts in debug mode.

In other words, before loading LDM instructions via JTAG, the debugger should first of all, set INTDIS bit in Debug Control Register via JTAG so that all interrupts are disabled in debug mode.

5.2.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev2, rev2a, rev3, rev3a, rev4p0-p2

5.3 Incorrect value of EmbeddedICE-RT version number read through JTAG interface

5.3.1 Summary

The EmbeddedICE-RT version is 7 for the ARM7TDMI-S rev4, but the value read via JTAG is 4.

5.3.2 Description

Bits 31:28 of the Debug Communications Channel Control Register contain a fixed pattern that denotes the EmbeddedICE-RT version number implemented within the ARM7TDMI-S r4p2. This should have a value of b0111, whether read through JTAG or CP14.

However, a read of this register through JTAG scan chain 2 results in a value of b0100 being read, which is incorrect.

5.3.3 Implication

This has an implication for debug tools only

5.3.4 Workarounds

No workaround is necessary as it still works with debug tools

5.3.5 Corrective Action

No corrective action is performed on rev4p0-p2.

Correct operation of the ARM7TDMI-S r4p2 with this bug has been demonstrated with the following debug tools:

- Multi-ICE with armsd/ADW/AXD
- RealView-ICE with RealViewDebugger

5.3.6 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev4p0-p2

5.4 Incorrect EIS log file when the core is in Thumb state

5.4.1 Summary

During simulation, all the instructions executed by the processor are logged into a file (EIS log file) containing the address, the hexadecimal opcode, and the mnemonic.

The EIS log file contains the wrong information when the processor is in Thumb state. The problem occurs only in simulation, as the module generating the log files is not synthesised.

When in Thumb state, the log file contains a half word of the corresponding expanded ARM instruction. However, the disassembler returns wrong mnemonic. The address of the instruction and whether it is executed (CCFAIL) is correct.

The ARM7TDMI-S r4p2 netlist is NOT affected by this problem, nor is the ETM.

As the ETM returns only the address of the instructions, not the opcode, the comparison between ETM and this log file is still correct.

5.4.2 Description

To correct the problem, the disassembler must receive the compressed Thumb instruction instead of expended ARM instruction.

Here is an example of EIS log file with the error:

```
>      102511 ns                  00001050  E12FFF10  BX r0
>      102591 ns                  00001059      800E  STRH r6,[r1,#0]
>      102611 ns                  0000105B      7001  STRB r1,[r0,#0]
>      102631 ns                  0000105D      0001  LSL r1,r0,#0
>      102651 ns                  0000105F      0000  LSL r0,r0,#0
>      102671 ns                  00001061      0000  LSL r0,r0,#0
>      102691 ns     CCFAIL       00001063      004D  LSL r5,r1,#1
>      102711 ns     CCFAIL       00001065      004C  LSL r4,r1,#1
>      102731 ns     CCFAIL       00001067      004B  LSL r3,r1,#1
>      102751 ns     CCFAIL       00001069      004A  LSL r2,r1,#1
>      102771 ns                  0000106B      0007  LSL r7,r0,#0
```

After the correction, the EIS log contains now:

```
>      102511 ns                  00001050  E12FFF10  BX r0
>      102591 ns                  00001059      46F0  MOV r8,r14
>      102611 ns                  0000105B      2701  MOV r7,#1
>      102631 ns                  0000105D      2001  MOV r0,#1
>      102651 ns                  0000105F      3800  SUB r0,#0
>      102671 ns                  00001061      0000  LSL r0,r0,#0
>      102691 ns     CCFAIL       00001063      D64D  BVS 0x1101
>      102711 ns     CCFAIL       00001065      D34C  BCC 0x1101
>      102731 ns     CCFAIL       00001067      D04B  BEQ 0x1101
>      102751 ns     CCFAIL       00001069      D44A  BMI 0x1101
>      102771 ns                  0000106B      42B8  CMP r0,r7
```

Furthermore, the disassembler (disass module) used to print out the EIS log file does not translate correctly the BL and BLX instructions in thumb state. The last version of disass modelgen corrects this problem.

Here is an example of EIS log file with the previous version of disass modelgen:

```
>     235571 ns                  00003450  E12FFF14  BX r4
>     235651 ns                  00003455      F000  DCI 0xf000 ; ? Undefined
>     235691 ns                  00003457      F801  DCI 0xf801 ; ? Undefined
>     235791 ns                ( 0000345A      0000  LSL r0,r0,#0 )
```

With the new version of disass, we now have:

```
>     235571 ns                  00003450  E12FFF14  BX r4
>     235651 ns                  00003455      F000  BL (1st part: LDR r14,=0x3459)
>     235691 ns                  00003457      F801  BL 0x345d
>     235791 ns                ( 0000345A      0000  LSL r0,r0,#0 )
```

5.4.3 Implication

In RTL simulations, the EIS log file will be incorrect whenever the core is in Thumb state.

5.4.4 Corrective Action

New deliverables correct the EIS log file problem in verilog and vhdl:

- AT080-MN-22100-r4p2-04rel0 replaces AT080-MN-22100-r4p2-03rel0 to update A7S.v and A7S_log_status.v files
- AT080-MN-22101-r4p2-04rel0 replaces AT080-MN-22101-r4p2-03rel0 to update disass modelgen (disass.so and disass.v files)
- AT080-MN-23100-r4p2-04rel0 replaces AT080-MN-23100-r4p2-03rel0 to update A7S.vhd, A7S_log_status.vhd, A7S_dataio.vhd and A7S_components.vhd files)
- AT080-MN-23101-r4p2-04rel0 replaces AT080-MN-23101-r4p2-03rel0 to update disass modelgen (disass.so and disass.vhd files)

README files explain how to patch the ARM7TDMI-S r4p2.

Here is the first part of README_verilog delivered with the verilog tar files:

```
> AT080-MN-22100-r4p2.04rel0.tar replaces AT080-MN-22100-r4p2.03rel0.tar
> AT080-MN-22101-r4p2.04rel0.tar replaces AT080-MN-22101-r4p2.03rel0.tar
-----------------------------------------------------------------------------------------------
NOTE: to make sure that all new files are updated properly, you should, first of all, delete your old
directories:
> rm -rf <A7S_root_dir>/verilog/ARM7DTMIS
> rm -rf <A7S_root_dir>/verilog/modelgen

and then install the new deliverables:
> tar xvf AT080-MN-22100-r4p2.04rel0.tar
> tar xvf AT080-MN-22101-r4p2.04rel0.tar
-----------------------------------------------------------------------------------------------

These new deliverables are intended to correct the problem with the EIS log file.
************************** IMPORTANT ************************************
```

```
*   These modifications will NOT change your netlist, as the module
*    A7S_log_status is not synthesised. It is instanciated between the lines :
*      // synopsys translate_off
*      // synopsys translate_on
*
*
*   These modifications only affect the file log.eis created during
*   simulations, and will NOT modify anything else.
*************************** IMPORTANT ************************************

The following files are different from the previous release :
    ./A7S.v
    ./A7S_log_status.v
```

Here is the first part of README_vhdl delivered with the vhdl tar files:

```
> AT080-MN-23100-r4p2.04rel0.tar replaces AT080-MN-23100-r4p2.03rel0.tar
> AT080-MN-23101-r4p2.04rel0.tar replaces AT080-MN-23101-r4p2.03rel0.tar
-------------------------------------------------------------------------------------------------------
NOTE: to make sure that all new files are updated properly, you should, first of all, delete your old
directories:
> rm -rf <A7S_root_dir>/vhdl/ARM7DTMIS
> rm -rf <A7S_root_dir>/vhdl/modelgen

and then install the new deliverables:
> tar xvf AT080-MN-23100-r4p2.04rel0.tar
> tar xvf AT080-MN-23101-r4p2.04rel0.tar
-------------------------------------------------------------------------------------------------------


These new deliverables are intended to correct the problem with the EIS log file.

*************************** IMPORTANT ************************************
*   These modifications will NOT change your netlist, as the module
*    A7S_log_status is not synthesised. It is instanciated between the lines :
*      // synopsys translate_off
*      // synopsys translate_on
*
*
*   These modifications only affect the file log.eis created during
*   simulations, and will NOT modify anything else.
*************************** IMPORTANT ************************************


The following files are different from the previous release :
    ./A7S.vhd
    ./A7S_components.vhd
    ./A7S_dataio.vhd
    ./A7S_log_status.vhd
```

5.4.5 Impacted revisions

This erratum impacts the following revision of ARM7TDMI-S: rev4p0-p2
All previous revisions do not have an EIS module.

5.5 SWI instruction and Prefetch Abort not properly handled if they follow a condition code failed Undefined Instruction

5.5.1 Summary

If a SWI instruction follows a condition code failed undefined instruction, the UNDEFINED trap is taken instead of the SWI exception. If a Prefetch Abort occurs on the instruction following a condition code failed undefined instruction, the UNDEFINED trap is taken instead of the Prefetch Abort handler.

5.5.2 Description

In case of the following instruction sequences:

CCFAIL Undef

SWI

or

CCFAIL Undef

Aborted instruction

the Undefined instruction is not executed because of its failing condition code. But the UNDEFINED trap is taken instead of taking the SWI exception (SWI) or the Prefetch Abort handler (Aborted instruction).

Note that coprocessor instructions leading to the UNDEFINED trap to be taken (for example if no coprocessor is able to execute such an instruction) are properly handled: if their condition code fails, they are not executed and the core properly executes the following SWI or aborted instruction.

5.5.3 Implication

No undefined instruction should occur in normal application. If a sequence as described above happens, this means that the application is not written properly and taking the UNDEFINED trap is safe in this case as it will always be taken. It is a deviation from the ARM Reference Manual but without any fatal implication.

5.5.4 Workaround

Not applicable as the instruction sequence described above should never happen.

5.5.5 Corrective action

No correction is planned for this erratum.

5.5.6 Impacted revisions

This erratum impacts the following revision of the product: rev2, rev2a, rev3, rev3a and rev4p0-p3.

6 SYSTEM ERRATA

6.1 LDC/LDCL/STC/STCL instructions incorrectly decoded when non-indexed addressing mode is used

6.1.1 Conditions

The core is unable to decode LDC/LDCL/STC/STCL instructions without base register write back and non-indexed addressing mode. Base Register write-back is controlled by bit-21 (W-bit) of the instruction opcode.

6.1.2 Implications

These specific Coprocessor load and store instructions will fail for systems that implement external coprocessors.

6.1.3 Workaround

Use an LDC/LDCL/STC/STCL with immediate post-indexed addressing mode. This should then be followed by a ADD (U bit-23 clear) or SUB (U bit-23 set) instruction to add or subtract four times the value of the immediate offset to or from the base register. This will incur a cost of 1 additional instruction per affected LDC/LDCL/STC/STCL instruction.

6.1.4 Impacted revisions

This erratum impacts all revisions of ARM7TDMI-S up to and including r4p3.

6.2 Sequential MRC instructions may not execute correctly

6.2.1 Conditions

If two or more MRC instructions are fetched sequentially, the second and any subsequent MRC instructions may not be correctly executed by the core.

An MRC instruction takes the form of:

MRC{<cond>} <coproc>, <opcode_1>, <Rd>, <CRn>, <CRm>{, <opcode_2>}

The bug is only encountered if the value of <opcode_1> is set to 3'bx1x and only if CPA/CPB are being driven low by the coprocessor a cycle earlier with respect to CPnOPC/CPnMREQ going low.

Figure 1, shows how CPA/CPB are driven in the AVS testbench, causing the bug not to trigger.

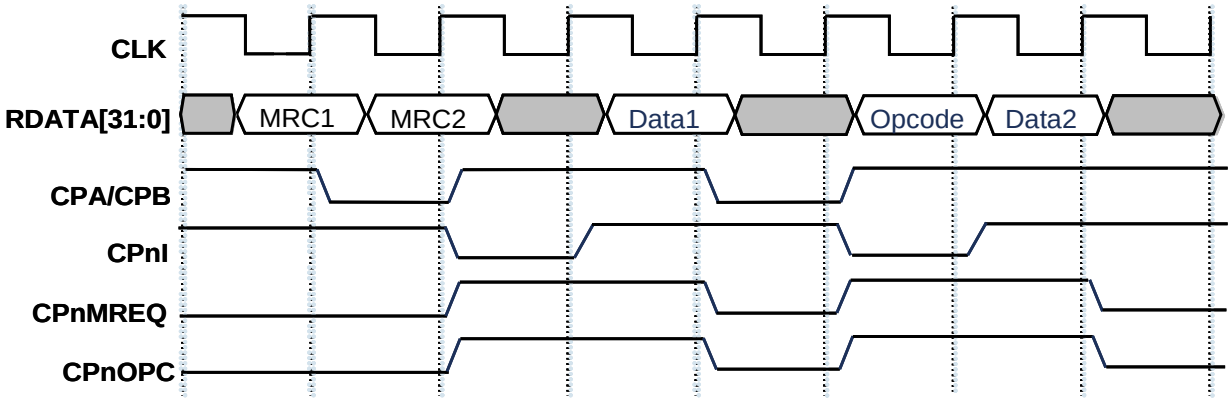


Fig 1: CPA/CPB as driven in the AVS testbench

Figure 2, shows how the bug is triggered, e.g. when CPA/CPB are driven a cycle earlier by the coprocessor.

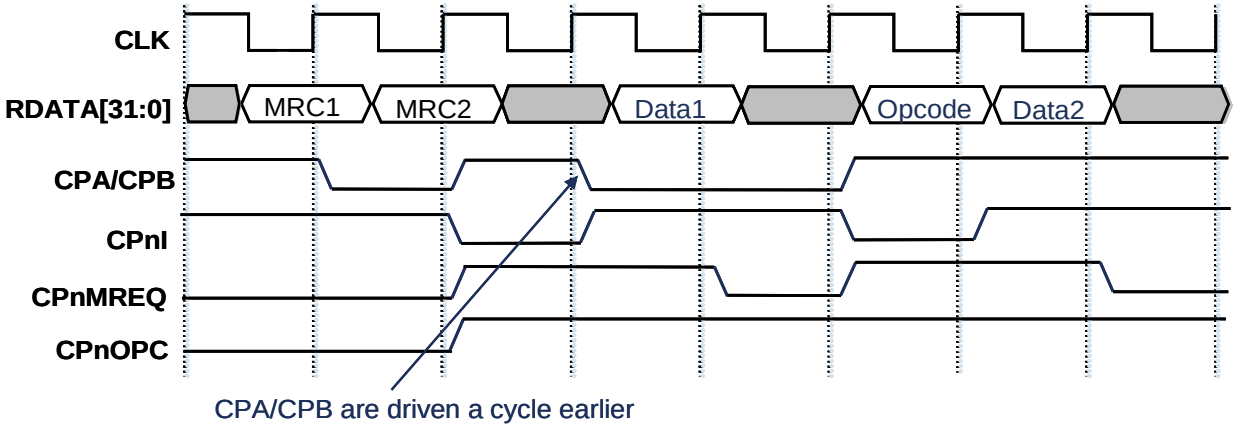


Fig 2: CPA/CPB driven a cycle earlier

6.2.2 Implications

This only has implications when external coprocessors are used.

6.2.3 Workaround

There are two possible workarounds:

1: Delay CPA/CPB inputs by a full clock cycle, e.g. so that they are taken low when CPnOPC/CPnMREQ go low.

2: Insert a NOP instruction between each MRC instruction that appear in program code, there will be an overhead of an additional instruction for each sequential MRC instruction fetched and executed by the core.

6.2.4 Impacted revisions

This erratum impacts all revisions of ARM7TDMI-S up to and including r4p3.