ARM7TDMI Rev4 Errata List

CPU Cores Division

Document number:	ARM7TDMI-PRDC-000580 6.0
Date of Issue:	1st August 2001
Author:	Jon Rijk/Steve Poole
Authorized by:	Keith Clarke

Abstract

This document describes the known errata in the ARM7TDMI Revision 4 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

1.1 Current History

Issue	Date	By	Change
6.0	1st August 2001	Steven Poole, CPU Project Manager Jon Rijk, CPU Validation Engineer	New issue.
6.0	23rd November 2001	Clive Watts, ARM7 Product Manager Jean-Philippe Martin	
4.4	29th November 2001	Clive Watts, ARM7 Product Manager	Added sections 4.4 and 5.1

1.2 References

This document refers to the following documents.

Ref.	Document No	Author(s)	Title
1	ARM DDI 0210A	ARM	ARM7TDMI (Rev4) Technical Reference Manual

1.3 Scope

This document describes the errata discovered in the implementation of the ARM7TDMI Rev4, 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.

1.4 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Bounced Coprocessor Instruction	An invalid coprocessor instruction that results in the Undefined Instruction trap being taken.
Breakpoint	A debugging mechanism used to halt execution due to an **instruction fetch**. For the breakpoint to cause debug state entry, the instruction must reach the execution stage of the pipeline, but it will be prevented from executing. This enables a debugger to observe the state prior to that instruction's execution. A breakpoint can be made to occur by either appropriate triggering of the Embedded ICE watchpoint units or by asserting the **BREAKPT** signal during an instruction fetch.
DBGBREAK	EmbeddedICE breakpoint/watchpoint indicator. This signal identifies the current memory access with a debug condition. If the memory access is an instruction fetch then a **breakpoint** is indicated, otherwise a **watchpoint** is indicated.
DBGRQ	Debug request. An input used to signal the processor to enter debug state once the current executing instruction completes.
Debugger/Debugging Tool	A debugging system that includes a program used to detect, locate and correct software faults, together with custom hardware that supports software debugging.
Single-stepping	A debugging operation used to step through the flow of program execution, instruction by instruction. This can be implemented by using a single **watchpoint unit** configured to cause a **breakpoint** on the next instruction to be executed.
Watchpoint	A debugging mechanism used to halt execution due to a memory **data access**. Watchpoints cause debug state entry once the instruction causing the data access fully completes; this may include accepting state changes due to pending exceptions. Watchpoints enable a debugger to observe memory changes, such as updates to variables. A watchpoint can be made to occur by either appropriate triggering of the Embedded ICE watchpoint units or by asserting the **BREAKPT** signal during a data memory access.
Watchpoint Unit	Custom EmbeddedICE hardware capable of triggering a breakpoint or watchpoint when all its comparators match. Each watchpoint unit has several registers to configure the type of comparison desired, enabling matches against any value on the address bus, and/or the data bus and/or various bus control signals.

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.

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.

Category 1	None
Category 2	Debug Request coincident with Aborting Store (1) Watchpoint and Prefetch Abort (2) Watchpoint coincident with Debug Request (3) Possible Internal DC-path (4)
Category 3	External Data Bus, D[], driven during debug state (5)

3 CATEGORY 1 ERRATA

There are no errata in this group.

4 CATEGORY 2 ERRATA

4.1 Debug Request coincident with Aborting Store (1)

If **DBGRQ** or the 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.

4.1.1 Conditions

A combination of two conditions is required to cause this erratum:

1) Debug Request is asserted and
2) A store instruction that causes a Data Abort on the final access is executing

If the above conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and execute from an incorrect address.

4.1.2 Implications

In this erratum, the Debug Request takes effect before the recognition of the abort. This may result in unreliable debug entry, the abort being missed and premature exit of debug state. Thus unintended or unpredictable device behaviour may result.

4.1.3 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. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only, therefore there is no plan to revise the design to resolve this erratum at the current time.

4.2 Watchpoint and Prefetch Abort (2)

If a watchpoint occurs followed by a prefetch aborted instruction then the processor may not restart execution from the correct point after exiting debug state.

4.2.1 Conditions

A combination of two conditions is required to cause this erratum:

1) An instruction that causes a watchpoint has been executed and
2) The second following instruction would Prefetch Abort if executed.

If the above conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and execute from the incorrect address.

4.2.2 Implications

In this erratum, the Prefetch Abort gets recognised prematurely, that is before debug entry occurs. This causes the PC to 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.

4.2.3 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. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only, therefore there is no plan to revise the design to resolve this erratum at the current time.

4.3 Watchpoint coincident with Debug Request (3)

If **DBGRQ** or the scan chain created debug request is asserted as debug entry due to a watchpoint is occurring then the processor may not restart execution from the correct point after exiting debug state.

4.3.1 Conditions

A combination of two conditions is required to cause this erratum:

1) Debug Request is asserted and
2) a load that causes a watchpoint is executing.

If the above conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and execute from the incorrect address.

4.3.2 Implications

In this erratum, the Debug Request takes effect before the recognition of the watchpoint. This may result in unreliable debug entry, the watchpoint being missed and premature exit of debug state. Thus unintended or unpredictable device behaviour may result.

4.3.3 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. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only, therefore there is no plan to revise the design to resolve this erratum at the current time.

4.4 Possible Internal DC-path (4)

When the core is stopped in debug state it is possible to activate an internal DC path

4.4.1 Conditions

It is possible to cause the internal DC-path if the following conditions are met:

1) scan chain 0,1 or 4 is selected by the tap controller.
2) TCK input is low.
3) the tap controller state machine is in state 5, Update-DR.
4) the tap controller instruction register is not EXTEST, CLAMP or CLAMPZ.
5) the core clock, MCLK is low, or nWAIT is low.
6) the core is performing a write onto the external data bus, ie nRW is high.

These conditions can occur if debugging is being used, for example using Multi-ICE, whenever a non system speed STR or STM is executed while the core is halted in debug state. A non system speed STR or STM would be used in debug state to make the register contents of the core visible to the debugger host.

Note that if debug is not being used these conditions will not occur.

4.4.2 Implications

Although no functional problem will be noticed if the DC-path is activated, it is possible that the lifetime of the part will be reduced.

4.4.3 Workarounds

None at this time.

This issue will be fixed in release 4.1 of the core.

5 CATEGORY 3 ERRATA

5.1 External Data Bus, D[], driven during debug state (5)

5.1.1 Description

When the core is stopped in debug state, the external data bus, D[], can be driven when external logic may not be expecting it.

Normally external logic would use the BUSDIS pin to determine if ARM7TDMI is driving the external data bus during test operations. Alternatively external logic would use a combination of nMREQ, nENOUT and nENOUTI to determine if ARM7TDMI is driving the external data bus during normal operation. In this revision of ARM7TDMI it is not possible to determine that the external data bus is being driven during debug by only looking at the BUSDIS, nMREQ, nENOUT and nENOUTI output pins.

5.1.2 Conditions

1) Scan chain 1 selected

2) The tap controller instruction register contains the INTEST instruction.

These conditions occur when the ARM7TDMIr4 is stopped in debug state and debug instructions are shifted into scan chain 1 (MultiICE uses this method to examine the state of the core).

5.1.3 Implications

External data bus clashes can occur when the core is stopped in debug state, scan chain 1 is selected and the tap controller instruction register contains the INTEST instruction.

5.1.4 Workaround

It is possible to decode that ARM7TDMI is driving the external data bus during these conditions by additional decode logic on the SC[3:0] and IR[3:0] output pins of ARM7TDMI. The following equation can be used to determine that ARM7TDMI is driving:

SC[3:0]==”0001” AND IR[3:0]==”1100”

This issue will be fixed in release 4.1 of the core.