ETM7 Rev 1a Errata

CPU Cores Division

Document number: **ETM7-PRDC-000627 3.0**
Date of Issue: 13 February 2003

Abstract

This document describes the known errata in the ETM7 Rev 1a design.

Keywords

ETM, ETM7, trace, 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

This document replaces document no. TRACE-0090-CUST-DEI-A.

Issue	Date	Change
A	2 May 2001	First Issue.
2.0	20 August 2001	Status changed to Non Confidential and new document format
3.0	13 February 2003	Upgraded “JTAG instruction register not connected to TDO during Shift-IR” to category 2 and added workaround.

1.2 References

This document refers to the following documents.

Document No	Author(s)	Title
ARM-IHI-0014	ARM	Embedded Trace Macrocell Specification

1.3 Scope

This document describes the errata discovered in the implementation of the ETM7 Rev 1a, categorized by level of severity. Each description includes:

- where the implementation deviates from the specification
- the conditions under which erroneous behavior occurs
- the implications of the erratum for users and tool vendors
- the application and limitations of a workaround where possible
- the status of corrective action

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.

Category 1	No known errata.
Category 2	Execution status unknown prior to an interrupt or prefetch abort Address range cannot include 0xFFFFFFFF Tracing from power-on reset not supported Stall cycles reported on wrong instruction JTAG instruction register not connected to TDO during Shift-IR
Category 3	Extra instruction traced prior to debug entry

3 CATEGORY 1 ERRATA

There are no category 1 errata.

4 CATEGORY 2 ERRATA

4.1 Execution status unknown prior to an interrupt or prefetch abort

4.1.1 Description

The trace port protocol allows for each instruction traced to be reported with a corresponding branch address to indicate the address of the next instruction. If this branch address is not output then the trace tools must calculate the address of the next instruction. If a branch address is output then the PIPESTAT pins of the trace port will indicate *Branch Executed* (BE) or *Branch with Data* (BD).

When an interrupt or prefetch abort occurs, the instruction which follows the last instruction to be executed should be traced (even though it did not execute) with a branch address to the relevant exception vector. In the case of interrupts this is often referred to as the interrupted instruction. The trace tools must recognize that the branch was to an interrupt or prefetch abort exception vector, and correctly treat that instruction as having not executed. In this way, the trace can reliably indicate if the last instruction executed failed its condition codes.

Sometimes the ETM does not trace this extra instruction but instead traces the last instruction executed as having branched to the exception vector.

Table 1 shows an example of incorrect trace caused by this erratum. Table 2 shows an example where this erratum does not occur.

Table 1 Example of incorrect trace behavior on interrupt

Actual instructions executed		**Correct behavior**	**Actual behavior**	
Address	**Instruction**	**Trace generated**	**Trace generated**	**Decompressed trace**
		Trace starts, address `0x00001000`	Trace starts, address `0x00001000`	
`0x00001000`	`ADD r1, r1, 1`	Instruction executed	Instruction executed	`0x00001000`
`0x00001004`	`B 0x00001020`	Instruction executed, branch to `0x00001020`	Instruction executed, branch to `0x00000018`	`IRQ!`
`(0x00001020)`	*(Not executed due to interrupt)*	Instruction executed, branch to `0x00000018`[a]		
`0x00000018` . . .	*IRQ hander* . . `SUBS pc, r14, #4`	IRQ handler traced . . Instruction executed, branch to `0x00001020`	IRQ handler traced . . Instruction executed, branch to `0x00001020`	`0x00000018` . . .
`0x00001020`	…	…	…	`0x00001020`

[a] Extra instruction traced to indicate interrupt

Table 2 Example of correct trace behavior on interrupt

Actual instructions executed		Actual behavior	
Address	**Instruction**	**Trace generated**	**Decompressed trace**
		Trace starts, address `0x00002000`	
`0x00002000`	`ADD r1, r1, 1`	Instruction executed	`0x00002000`
`0x00002004`	`BNE 0x00002020` Failed its condition codes	Instruction executed but failed its condition codes	`0x00002004` Failed its condition codes
`(0x00002020)`	*(Not executed due to interrupt)*	Instruction executed, branch to `0x00000018`[a]	`IRQ!`
`0x00000018` . . .	*IRQ hander* . . `SUBS pc, r14, #4`	IRQ handler traced . . Instruction executed, branch to `0x00002020`	`0x00000018` . . .
`0x00002020`	…	…	`0x00002020`

[a] Extra instruction traced to indicate interrupt

4.1.2 Conditions

The conditions under which this erratum occurs are not easily predicted. It never occurs if the last executed instruction failed its condition codes.

4.1.3 Implications

The last instruction executed before an interrupt or prefetch abort might be missing from the trace. Some tools always report the extra instruction traced as having executed, in which case the user will instead see an extra instruction reported before an interrupt or prefetch abort when this erratum does not occur.

If tracing continues until after the exception handler returns, the user can use the address of the first instruction executed upon returning from the exception to determine the actual behavior.

4.1.4 Workaround for tools vendors

Since the execution status is unknown, development tools cannot reliably report the last instruction executed before an abort or an interrupt.

The development tools must display the last instruction traced with its execution status shown as ‘unknown’, rather than suppress it as defined in the trace port protocol.

4.1.5 Implications of workaround

It is not possible to determine which instruction was executed immediately before an interrupt or prefetch abort, without referring to the last instruction traced before the interrupt.

4.1.6 Correction

No correction is planned. This erratum is documented in the ETM specification.

4.2 Address range cannot include 0xFFFFFFFF

4.2.1 Description

There is no way to define a range which includes `0xFFFFFFFF`. Ranges are defined to be exclusive of the upper address, so a range with an upper address of `0xFFFFFFFF` only includes addresses up to `0xFFFFFFFE`.

4.2.2 Conditions

All.

4.2.3 Implications

The ability to monitor accesses to this address is seriously reduced.

4.2.4 Workaround

To monitor accesses to `0xFFFFFFFF`, the upper address on the range must be set to `0xFFFFFFFF`, with the size mask set to clear bits 1:0 (bits 4:3 of the access type register must be set to *b11*). This causes the upper bound of the address range never to match, and so the range has no upper limit and accesses to `0xFFFFFFFF` are monitored correctly.

4.2.5 Implications of workaround

No implications.

4.2.6 Correction

This will become part of the ETM specification.

4.3 Tracing from power-on reset not supported

4.3.1 Description

The first instructions executed by the core after the ETM is reset cannot be traced. Invalid trace will be produced if this is attempted. This is because the ETM's pipeline follower is not reset when the ETM is reset, saving area and increasing speed.

4.3.2 Conditions

This erratum only occurs if tracing from power-on reset is attempted, as illustrated in Table 3.

Table 3 Example of tracing from power-on reset

Core nRESET	ETM nRESET	Comments
low	low	Core and ETM taken into reset
low	high	ETM released from reset ETM registers programmed to enable tracing
high	high	Core released from reset Tracing begins

Tracing through core reset when the ETM has already seen instructions traced is supported. Table 4 illustrates this case.

Table 4 Example of tracing through reset

Core nRESET	ETM nRESET	Comments
high	high	Core running
low	high	Core reset ETM registers programmed to enable tracing
high	high	Core released from reset Tracing begins

4.3.3 Implications

The trace produced by the ETM under these conditions is unpredictable.

4.3.4 Workaround

Always release the core from reset before enabling trace, if necessary forcing another reset immediately afterwards to begin tracing. Table 5 illustrates an example of this workaround.

Table 5 Example of Workaround

Core nRESET	ETM nRESET	Comments
low	low	Core and ETM taken into reset
high	high	Core released from reset ETM released from reset Core running briefly to initialize ETM
low	high	Core reset ETM registers programmed to enable tracing
low	high	ETM registers programmed to enable tracing
high	high	Tracing begins

4.3.5 Implications of workaround

Some extra instructions will be executed. Since the number that must be executed is small, this is unlikely to be significant in time or behavior.

4.3.6 Correction

No correction is planned.

4.4 Stall cycles reported on wrong instruction

4.4.1 Description

The timestamps or cycle numbers, if captured, might not truly reflect the number of cycles taken by each instruction. Stall cycles which cause an instruction to take longer than the minimum time to execute might be reported on an earlier instruction.

It is occasionally useful to be able to determine the exact cycles on which instructions are executed, for example to perform basic performance analysis or to diagnose problems with the memory system. The information can be preserved by one of two methods:

- By selecting *cycle-accurate mode*, the ETM can be configured to cause trace to be captured on every cycle during a trace region.
- By using a trace port analyzer or logic analyzer which is capable of saving timestamps with each cycle of trace.

Under some circumstances, extra instruction execution cycles are reported alongside a previous instruction, making it appear that that instruction took longer to execute instead. While the number of cycles that a sequence of instructions takes to execute is correctly reported, the number of cycles taken by individual instructions is not.

This erratum does not affect ETM9 Rev 0a, although ARM strongly recommends that the latest revision of ETM9 is used in all new designs. The erratum was introduced as a result of improvements to the effectiveness of the FIFOFULL mechanism, although it is occurs whether FIFOFULL is enabled or not.

Table 6 shows an example where a stall caused by an LDR is traced with the wrong instruction. Note that the ETM reports an instruction when the following instruction begins execution. Therefore the time taken by an instruction is the time between that instruction and the previous instruction in the trace.

Table 6 Example of correct and incorrect trace in the presence of stalls

Cycle	Address	Instruction	Actual trace	Expected trace	Notes
500	1000	NOP			Instruction enters ETM pipeline
501	1004	NOP			1004 triggers tracing of 1000, 9 cycles later
502	1008	NOP			
503	100C	NOP			
504	1010	NOP			
505	1014	NOP			
506	1018	NOP			
507	101C	LDR			
508		*(LDR stalled)*			Most of ETM pipeline stalled along with processor pipeline
509	1020	NOP			
510	1024	B 1040	IE (instruction executed) for 1000	IE for 1000	
511		*(Branch delay)*	WT (wait)	IE for 1004	Stall in ETM pipeline observed after only 3 cycles
512		*(Branch delay)*	IE for 1004	IE for 1008	
513	1040	NOP	IE for 1008	IE for 100C	
514			IE for 100C	IE for 1010	
515			IE for 1010	IE for 1014	
516			IE for 1014	IE for 1018	
517			IE for 1018	WT	Stall should be observed in line with other instructions

518			IE for 101C	IE for 101C	
519			IE for 1020	IE for 1020	
520			WT	WT	Stalls caused by the processor are correctly reported
521			WT	WT	
522			IE for 1024	IE for 1024	Instruction traced upon completion

4.4.2 Conditions

As shown in Table 6, only external stalls that are caused by deasserting CLKEN (such as memory stalls) are misreported. All stalls caused by the processor, including branch delays, register interlocks and coprocessor busy-waits, are reported correctly. External (CLKEN) stalls are reported 6 cycles early.

4.4.3 Implications

It is hard to use the timestamp information to gain information on stalls caused by the memory system. While this is beyond the scope of the original design aims of the ETM, precise stall information has proven to be extremely useful to some users in debugging system level issues.

4.4.4 Workaround

It is often possible to model the behavior of the core as it would behave if there were no external stalls to determine which stalls are external and which are internal. To do this in all situations would require a complete cycle-accurate model of the core. However, in most cases the number of cycles required by each instruction can be easily predicted, and is documented in the Technical Reference Manual for the core.

It is therefore possible to determine where external stall cycles have been inserted, and to count forward 6 cycles from that point to find where they should have been inserted, not counting other external stall cycles. This is a complex procedure which cannot easily be automated, and cannot be regarded as a complete workaround.

Table 7 shows an example of this technique. Note that, as in Table 6, the trace reports stall cycles before the corresponding instruction is traced, not after.

Table 7 Example of how to calculate the correct location of a stall

Cycle	Address	Instruction	Actual trace	Reconstructed trace	Notes
600	2000	MOV r1,r0			
601	2004	LDR r2,var0			
602		*(interlock on r2)*			This is an internal stall and can be predicted
603	2008	ADD r2,r2,#1			
604	200C	B 2018			
605		*(Branch delay)*			These cycles are also predictable internal stalls
606		*(Branch delay)*			
607	2018	ADD r2,r2,#1			
608	201C	LDR r3,var1			
609		*(LDR stalled)*			An external stall

610	2020	LDR r4,var2	IE (instruction executed) for 2000	IE for 2000	
611	2024	ADD r2,r2,#1	WT (wait)	WT	Only one stall was predicted for 2004. One must be an external stall. Counting forward 6 from the first puts it on 2018, the second on 201C. Remove it and try to place later.
612			WT	IE for 2004	
613			IE for 2004	IE for 2008	
614			IE for 2008	WT	
615			WT	WT	
616			WT	IE for 200C	
617			IE for 200C	IE for 2018	The stall could correspond to 2018 in theory, but in this system only the memory system can cause external stalls, and the code is in fast instruction memory, so an ADD could not cause an external stall.
618			IE for 2018	WT	The stall is far more likely to correspond to 201C, as this causes a LDR to on-chip memory. We can therefore deduce that this LDR caused precisely 1 stall cycle.
619			IE for 201C	IE for 201C	
620			IE for 2020	IE for 2020	The stall could not correspond to this load, as it is not 6 cycles ahead of a detected external stall in the trace.
621			IE for 2024	IE for 2024	Note that the total number of cycles taken for the sequence of instructions is reported correctly.

4.4.5 Correction

No correction is planned.

4.5 JTAG instruction register not connected to TDO during Shift-IR

4.5.1 Description

When the JTAG TAP controller is in the Shift-IR (0xA) state and the BYPASS instruction is selected, serial data is transferred from TDI to TDO through the BYPASS register with a delay of one clock cycle. The instruction register is supposed to be selected as the serial path between DBGTDI and DBGTDO in this situation. If the ETM TDO output is used to connect the TAP controller of another device in series with the ETM/ARM, then during Shift-IR, an undefined instruction can be shifted into the device connected.

This errata can only be observed when the ETM7 is used with an ARM7TDMI-S core, or an ARM7TDMI-S based core such as ARM720T Rev 4.

When connected to the SDOUTBS input of an ARM core, the ETM TDO output is ignored during Shift-IR. However the ARM7TDMI-S core and derivatives (for example ARM720T Rev 4) do not have an SDOUTBS input, so the ARM7TDMI-S TDO output must be connected to the ARMTDO input of the ETM instead.

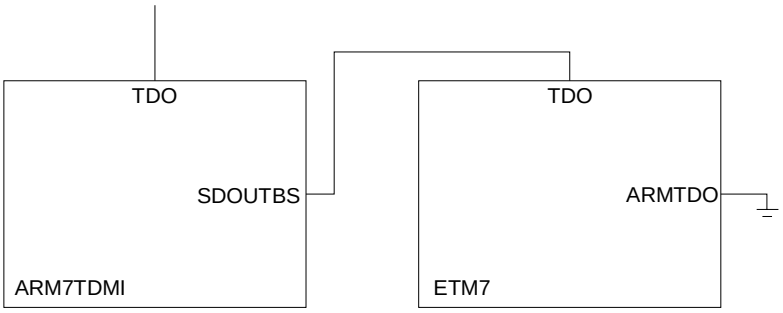

Figure 1: ARM7TDMI TDO connections

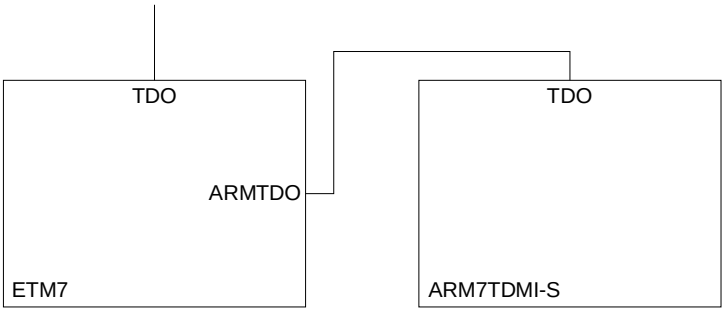

Figure 2: ARM7TDMI-S TDO connections

4.5.2 Conditions

If another device is connected in series with the ETM Tap controller, but after it in the JTAG chain then an undefined/wrong instruction could be shifted into the device if the ETM Tap controller is in the following state:

- The BYPASS instruction is selected, AND,
- The ETM TAP controller is in the Shift-IR state.

This only applies when using an ARM7TDMI core if the ETM and the ARM core are not connected as described in the ETM7 TRM, or when using an ARM7TDMI-S core and derivatives (e.g. ARM720T rev4).

4.5.3 Implications

The device connected in series with the ETM TAP controller but after it in the JTAG scan path will get incorrect instructions if using an ARM7TDMI-S core or derivative (for example ARM720T Rev 4).

4.5.4 Workaround

4.5.4.1 ARM7TDMI and ARM720T (Rev 3) based systems

The ETM should be connected as described in the ETM7 TRM.

4.5.4.2 ARM7TDMI-S and ARM720T (Rev 4) based systems

Tools vendors are recommended to follow the following steps:

- Instead of using BYPASS, use the CLAMP instruction. This causes the ARM7TDMI-S core to behave as if the BYPASS instruction is selected, but causes the ETM7 macrocell to behave as if the INTEST instruction is selected.
- Before using CLAMP, ensure that scan chain 6 has been deselected. This causes the ETM7 macrocell to route the ARM7TDMI-S TDO to its TDO, and in effect selects the core BYPASS register.

Alternatively, silicon implementers can follow the following steps:

- Implement a shadow TAP controller which shares its TDI, TMS, CLK, TCKEN and nTRST inputs with the ETM7 macrocell.
- Implement a multiplexer on the ETM7 and ARM7TDMI-S TDO outputs. When the shadow TAP controller is in BYPASS, select the ARM7TDMI-S TDO output, otherwise select the ETM7 TDO output.

4.5.5 Implications of workaround

None.

4.5.6 Correction

No correction is planned.

5 CATEGORY 3 ERRATA

5.1 Extra instruction traced prior to debug entry

5.1.1 Description

An instruction selected as a breakpoint may be traced prior to debug entry, even though it was not executed. It is flagged as having failed its condition codes, regardless of whether it is conditional.

5.1.2 Conditions

The conditions under which this erratum occurs are not easily predicted.

5.1.3 Implications

It will appear to the user that the breakpoint occurred at the wrong time.

5.1.4 Workaround

No workaround is known.

5.1.5 Implications of workaround

No implications.

5.1.6 Correction

No correction is planned.