ETM9™

Revision: r2p2

Technical Reference Manual

ARM®

ARM DDI 0157G

ETM9
Technical Reference Manual

Release Information

Change history

Date	Issue	Confidentiality	Change
14 December 1999	A	Open Access	ETM9 (Rev 0) release.
28 February 2000	B	Open Access	ETM9 (Rev 0a) release.
6 September 2000	C	Open Access	ETM9 (Rev 1) release.
15 January 2001	D	Open Access	ETM9 (Rev 2) release to NDA signatories only.
6 June 2001	E	Open Access	ETM9 (Rev 2a) release. Open Access.
20 August 2002	F	Open Access	ETM9 Revision: r2p2.
27 September 2006	G	Non-Confidential	Added content for DFT.

Proprietary Notice

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents
ETM9 Technical Reference Manual

List of Tables
ETM9 Technical Reference Manual

List of Figures
ETM9 Technical Reference Manual

Preface

This preface introduces the *ARM9 Embedded Trace Macrocell* (ETM9) (r2p2) and its reference documentation. It contains the following sections:

About this document

This document is the ETM9 (r2p2) Technical Reference Manual. The product is referred to as ETM9 throughout this manual.

Intended audience

This document has been written for hardware and software engineers who want to incorporate an ARM core based product having instruction and data trace capability into their hardware and/or software design.

This manual does not describe the operation of the ETM9. For information on how to use the ETM9 read the *Embedded Trace Macrocell Specification*.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this document, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Previous revisions are referred to using the old product revision status format.

Using this manual

This document is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the ETM9. This chapter includes the ETM9 block and functional diagrams.

Chapter 2 *Accessing ETM9 Registers*

Read this chapter for details of how to access the ETM9 registers.

Chapter 3 *Integrating the ETM9*

Read this chapter for details of how to integrate the ETM9 with an ARM9 microprocessor.

Chapter 4 *Memory Map Decode Interface*

Read this chapter for details of the Memory Map Decode interface used to integrate the ETM9 with memory-mapped peripherals.

Chapter 5 *Test Wrapper*

Read this chapter for details of the ETM9 test wrapper.

Chapter 6 ***ETM Integration Testing***

Read this chapter for details of how to validate the Trace macrocell in an ASIC design.

Chapter 7 ***Software Considerations for Trace***

Read this chapter for a description of the software issues that relate to the ETM9.

Chapter 8 ***Physical Trace Port Signal Guidelines***

Read this chapter for guidelines that ensure correct operation of the ETM and the trace tools.

Appendix A ***Signal Descriptions***

Read this appendix for a description of the ETM9 signals.

Appendix B ***Integrating the EtmMuxDemux Block into ETM9***

Read this appendix for details of how to integrate the EtmMuxdemux block into an ETM9.

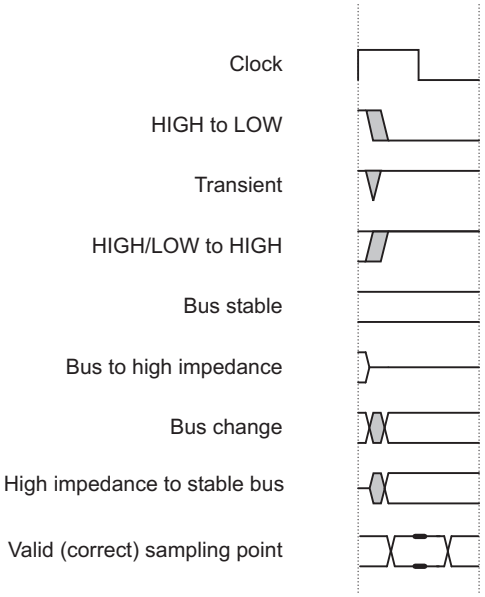
Typographical conventions

The following typographical conventions are used in this book:

bold	Highlights ARM processor signal names, and interface elements, such as menu names and buttons. Also used for terms in descriptive lists, where appropriate.
italic	Highlights special terminology, cross-references, and citations.
mono	Denotes text that can be entered at the keyboard, such as commands, file names and program names, and source code.
mono`space`	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
monospace italic	Denotes arguments to commands or functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.

Timing diagram conventions

This manual contains several timing diagrams. The following key explains the components used in these diagrams. Any variations are clearly labeled when they occur. Therefore, no additional meaning should be attached unless specifically stated.



Key to timing diagram conventions

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

This book contains information that is specific to the ETM9. Refer to the following documents for other relevant information:

- *Embedded Trace Macrocell Specification* (ARM IHI 0014)
- *ETM9 Revision:r2p2 Implementation Guide* (ARM DII 0001)
- ARM7 Embedded Trace Macrocell (ETM7) Technical Reference Manual (ARM DDI 0158)
- *Multi-ICE User Guide* (ARM DUI 0048)
- *ARM Architecture Reference Manual* (ARM DDI 0100).

Other publications

This section lists relevant documents published by third parties.

- Trace Port Analysis for ARM ETM Users Guide (Agilent Publications - publication number E5903-97002).

Feedback

ARM Limited welcomes feedback on the ETM9, and on the documentation.

Feedback on the ETM9

If you have any comments or suggestions about this product, please contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this book

If you have any comments about this document, please send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the ETM9 (r2p2). It contains the following section:

1.1 About the ETM9

The ETM9 provides instruction and data trace for the ARM9 family of microprocessors. This document describes the interface between an ARM9 Thumb family processor and ETM9. For details of the interface between an ARM7 processor and ETM7, refer to the *ARM7 Embedded Trace Macrocell (ETM7) Technical Reference Manual*. Elements of the ETM that are common to both ETM7 and ETM9, such as the trace protocol and the physical interface to the *Trace Port Analyzer* (TPA), are described in the *Embedded Trace Macrocell Specification*. Where the expression ETM appears in the text it refers to a nonspecific ETM (ETM7 or ETM9).

This document assumes that the ETM9 is an r2p2 version. For details of differences between different revisions, see *Product revisions* on page 1-4.

The block diagram of the ETM9 is shown in Figure 1-1.

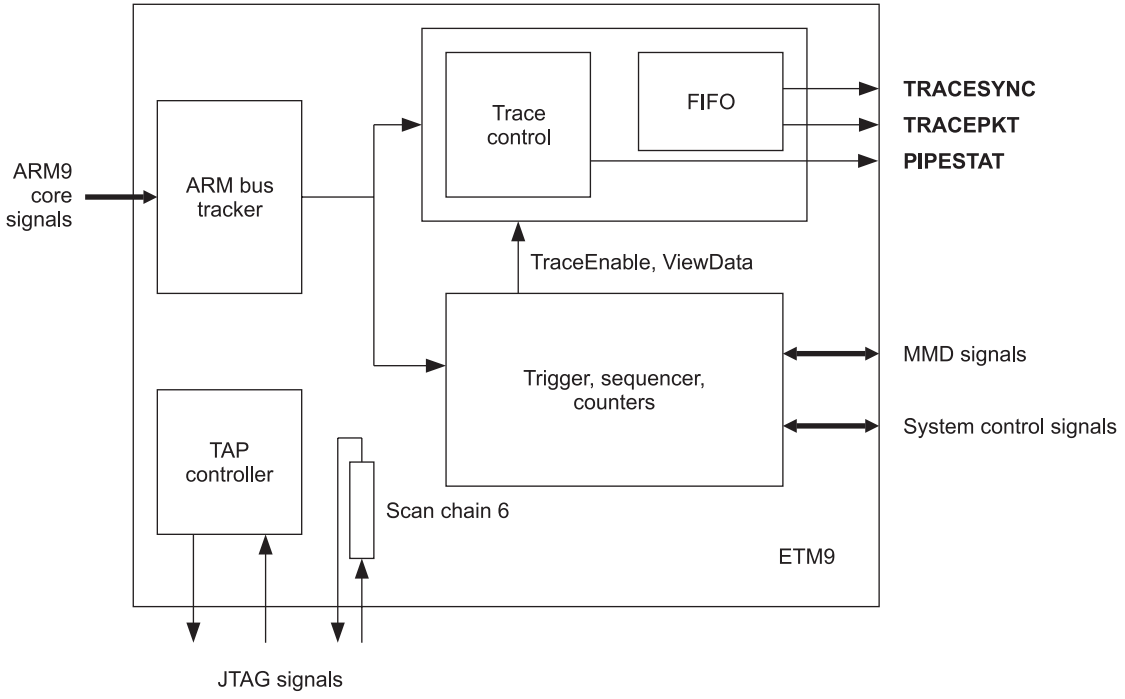
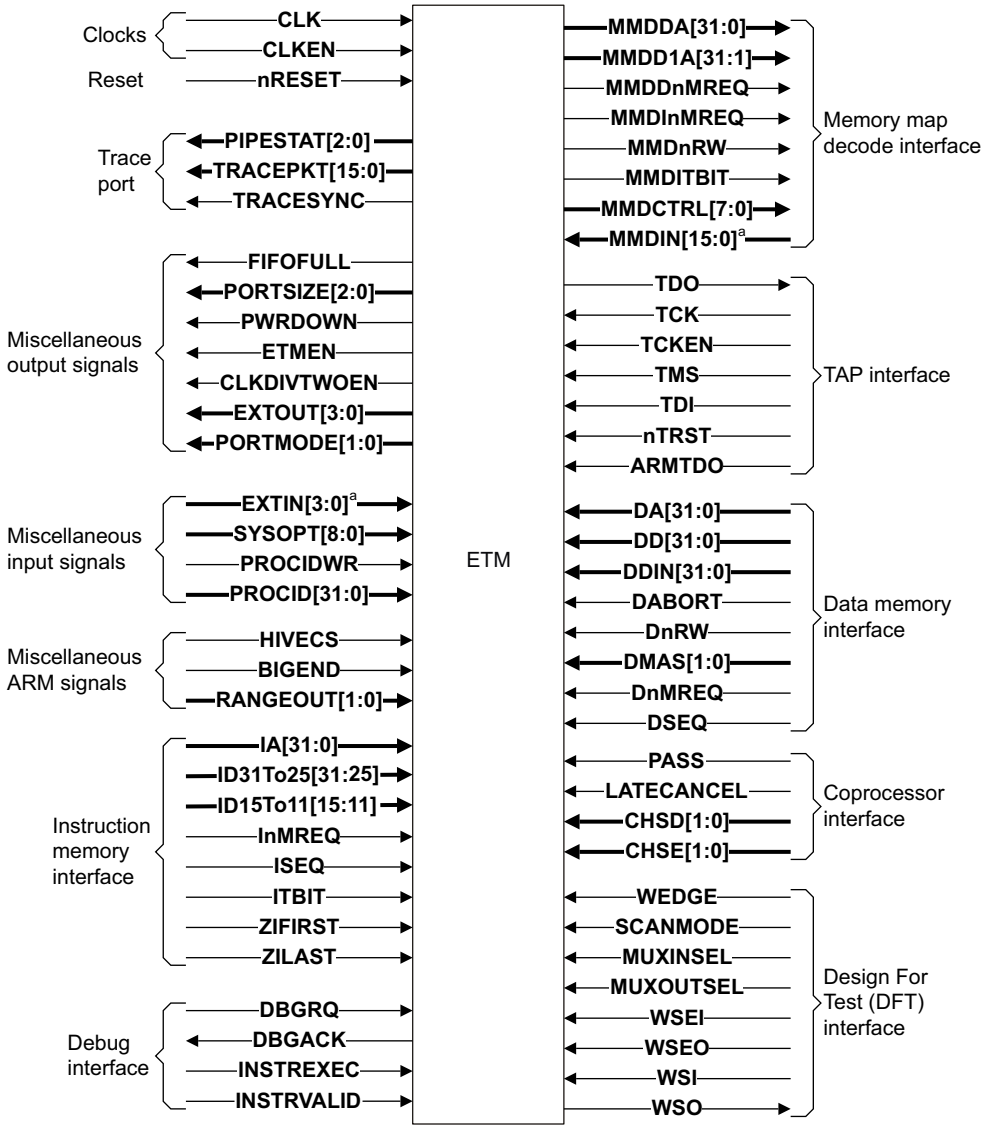


Figure 1-1 Block diagram of the ETM9

The signals diagram of the ETM9 is shown in Figure 1-2 on page 1-3 .



[a] The width of these inputs depends on the ETM configuration.

Figure 1-2 Signal diagram

1.2 Product revisions

This manual is for r2p2 of the ETM9.

This sections includes the following information about previous revisions:

- *ETM versions and variants*
- *Pin names* on page 1-5
- *Changes to the programmer's model in Rev 0a* on page 1-8
- *Changes to the programmer's model in Rev 1* on page 1-9
- *Changes to the programmer's model in Rev 2* on page 1-10
- *Changes to the programmer's model in Rev 2a* on page 1-10
- *Changes to the programmer's model in r2p2* on page 1-11.

1.2.1 ETM versions and variants

The ETM is subject to continuous improvement in conjunction with the development of the ARM processors. The history of ETM9 versions and variants is shown in Table 1-1.

Table 1-1 ETM versions and variants

ETM name	Protocol version	Architecture
ETM9 Rev 0	0	ETMv1.0
ETM9 Rev 0a	1	ETMv1.1
ETM9 Rev 1	2	ETMv1.2
ETM9 Rev 2	3	ETMv1.3
ETM9 Rev2a	5	ETMv1.3
ETM9 Rev 2p2	7	ETMv1.3

Reference to protocol versions is deprecated in ETMs supporting architecture ETMv2 and above. Protocol versions mentioned in previous issues of this document are now referred to by the corresponding architecture as shown in Table 1-1.

To indicate later revisions:

- ETM9 protocol versions 5 and 7 are equivalent to ETM9 protocol version 3.

1.2.2 Pin names

Table 1-2 shows the pin names on the different revisions of ETM9.

Table 1-2 ETM9 pin names

Type	ETM9 (Rev 0) signal	ETM9 (Rev 0a) signal	ETM9 (Rev 1) signal	ETM9 (Rev 2) and ETM9 (Rev 2a) signal ETM9 (Rev 2p2) signal
Input	**TCK**	**TCK**	**TCK**	**TCK**
Input	**TCKEN**	**TCKEN**	**TCKEN**	**TCKEN**
Input	**nTRST**	**nTRST**	**nTRST**	**nTRST**
Input	**TDI**	**TDI**	**TDI**	**TDI**
Input	**TMS**	**TMS**	**TMS**	**TMS**
Input	**ARMTDO**	**ARMTDO**	**ARMTDO**	**ARMTDO**
Input	**nRESET**	**nRESET**	**nRESET**	**nRESET**
Input	**BIGEND**	**BIGEND**	**BIGEND**	**BIGEND**
Input	**GCLK**	**CLK**	**CLK**	**CLK**
Input	**HIVECS**	**HIVECS**	**HIVECS**	**HIVECS**
Input	**nWAIT**	**CLKEN**	**CLKEN**	**CLKEN**
Input	**IA[31:1]**	**IA[31:1]**	**IA[31:1]**	**IA[31:0]**
Input	**InMREQ**	**InMREQ**	**InMREQ**	**InMREQ**
Input	**ISEQ**	**ISEQ**	**ISEQ**	**ISEQ**
Input	**ITBIT**	**ITBIT**	**ITBIT**	**ITBIT**
Input	-	-	-	**IJBIT**
Input	**IABORT**	-	-	-
Input	**ID31To24[31:24]**	**ID31To25[31:25]**	**ID31To25[31:25]**	**ID31To25[31:25]**
Input	**ID15To8[15:8]**	**ID15To11[15:11]**	**ID15To11[15:11]**	**ID15To11[15:11]**
Input	**DA[31:0]**	**DA[31:0]**	**DA[31:0]**	**DA[31:0]**
Input	**DD[31:0]**	**DD[31:0]**	**DD[31:0]**	**DD[31:0]**

Table 1-2 ETM9 pin names (continued)

Type	ETM9 (Rev 0) signal	ETM9 (Rev 0a) signal	ETM9 (Rev 1) signal	ETM9 (Rev 2) and ETM9 (Rev 2a) signal ETM9 (Rev 2p2) signal
Input	**DMAS[1:0]**	**DMAS[1:0]**	**DMAS[1:0]**	**DMAS[1:0]**
Input	**DMORE**	**DMORE**	-	-
Input	**DnMREQ**	**DnMREQ**	**DnMREQ**	**DnMREQ**
Input	**DnRW**	**DnRW**	**DnRW**	**DnRW**
Input	**DSEQ**	**DSEQ**	**DSEQ**	**DSEQ**
Input	**CHSD[1:0]**	**CHSD[1:0]**	**CHSD[1:0]**	**CHSD[1:0]**
Input	**CHSE[1:0]**	**CHSE[1:0]**	**CHSE[1:0]**	**CHSE[1:0]**
Input	**LATECANCEL**	**LATECANCEL**	**LATECANCEL**	**LATECANCEL**
Input	**PASS**	**PASS**	**PASS**	**PASS**
Input	**DABORT**	**DABORT**	**DABORT**	**DABORT**
Input	**DDIN[31:0]**	**DDIN[31:0]**	**DDIN[31:0]**	**DDIN[31:0]**
Input	**DBGACK**	**DBGACK**	**DBGACK**	**DBGACK**
Input	**INSTREXEC**	**INSTREXEC**	**INSTREXEC**	**INSTREXEC**
Input	-	-	**INSTRVALID**	**INSTRVALID**
Input	-	-	-	**ZIFIRST**
Input	-	-	-	**ZILAST**
Memory Map Decoder signals				
Input	**Mmd[16:1]**	**MMDIN[x:0]** x = 15 for large configuration x = 7 for mediumplus configuration x = 7 for medium configuration x = 3 for small configuration.	**MMDIN[x:0]** x = 15 for large configuration x = 7 for mediumplus configuration x = 7 for medium configuration x = 3 for small configuration.	**MMDIN[x:0]** x = 15 for large configuration x = 7 for mediumplus configuration x = 7 for medium configuration x = 3 for small configuration.

Table 1-2 ETM9 pin names (continued)

Type	ETM9 (Rev 0) signal	ETM9 (Rev 0a) signal	ETM9 (Rev 1) signal	ETM9 (Rev 2) and ETM9 (Rev 2a) signal ETM9 (Rev 2p2) signal
Output	**MemMapRegO[7:0]**	**MMDCTRL[7:0]**	**MMDCTRL[7:0]**	**MMDCTRL[7:0]**
Output	**IAFeSetupO[31:1]**	**MMDIA[31:1]**	**MMDIA[31:1]**	**MMDIA[31:1]**
Output	**ITBITFeSetupO**	**MMDITBIT**	**MMDITBIT**	**MMDITBIT**
Output	**InMREQFeSetupO**	**MMDInMREQ**	**MMDInMREQ**	**MMDInMREQ**
Output	**DAMeSetupO[31:0]**	**MMDDA[31:0]**	**MMDDA[31:0]**	**MMDDA[31:0]**
Output	**DnRWMeSetupO**	**MMDDnRW**	**MMDDnRW**	**MMDDnRW**
Output	**DnMREQMeSetupO**	**MMDDnMREQ**	**MMDDnMREQ**	**MMDDnMREQ**
EmbeddedICE signals				
Input	**EmbdIce1**	**RANGEOUT[0]**	**RANGEOUT[0]**	**RANGEOUT[0]**
Input	**EmbdIce2**	**RANGEOUT[1]**	**RANGEOUT[1]**	**RANGEOUT[1]**
External input signals				
Input	**ExtIn[4:1]**	**EXTIN[x:0]** x = 3 for large configuration x = 3 for mediumplus configuration x = 3 for medium configuration x = 1 for small configuration.	**EXTIN[x:0]** x = 3 for large configuration x = 3 for mediumplus configuration x = 3 for medium configuration x = 1 for small configuration.	**EXTIN[x:0]** x = 3 for large configuration x = 3 for mediumplus configuration x = 3 for medium configuration x = 1 for small configuration.
Miscellaneous input signals				
Input	-	-	**SYSOPT[7:0]**	**SYSOPT[8:0]**
Input	-	-	**PROCID[31:0]**	**PROCID[31:0]**
Input	-	-	**PROCIDWR**	**PROCIDWR**
External output signals				
Output	**ExtOut[4:1]**	**EXTOUT[3:0]**	**EXTOUT[3:0]**	**EXTOUT[3:0]**

Table 1-2 ETM9 pin names (continued)

Type	ETM9 (Rev 0) signal	ETM9 (Rev 0a) signal	ETM9 (Rev 1) signal	ETM9 (Rev 2) and ETM9 (Rev 2a) signal ETM9 (Rev 2p2) signal
Output	**DBGRQ**	**DBGRQ**	**DBGRQ**	**DBGRQ**
Output	**TDO**	**TDO**	**TDO**	**TDO**
Output	**EtmPwrDwn**	**PWRDOWN**	**PWRDOWN**	**PWRDOWN**
Output	**ETMEN**	**ETMEN**	**ETMEN**	**ETMEN**
Output	**TRACECLK**	-	-	-
Output	**PIPESTAT[2:0]**	**PIPESTAT[2:0]**	**PIPESTAT[2:0]**	**PIPESTAT[2:0]**
Output	**TRACEPKT[15:0]**	**TRACEPKT[15:0]**	**TRACEPKT[15:0]**	**TRACEPKT[15:0]**
Output	**TRACESYNC**	**TRACESYNC**	**TRACESYNC**	**TRACESYNC**
Output	**ETMFIFOFULL**	**FIFOFULL**	**FIFOFULL**	**FIFOFULL**
Output	-	-	**PORTMODE[1:0]**	**PORTMODE[1:0]**
Output	**ETMPORTSIZE[2:0]**	**PORTSIZE[2:0]**	**PORTSIZE[2:0]**	**PORTSIZE[2:0]**
Output	-	**CLKDIVTWOEN**	**CLKDIVTWOEN**	**CLKDIVTWOEN**

Note

The ETM9 r2p2 macrocell can implement **WSI**, **WSO**, **MUXINSEL**, **MUXOUTSEL**, **WSEI**, **WSEO**, **WEDGE**, and **SCANMODE**. These are DFT signals and their use is implementation defined. They are not directly tested by the test programs, but must be tested if you intend to use these connections. See Table 5-1 on page 5-4.

1.2.3 Changes to the programmer's model in Rev 0a

Rev 0a adds a small number of features to the programmer's model described in this section and fully documented in the third and fourth releases of the *Embedded Trace Macrocell Specification*. The version is identified by the protocol version field of the ETM configuration code register.

Half-rate clocking

When you use half-rate clocking, the TPA samples trace data signals on both the rising and falling edges of **TRACECLK**.

A pin, controlled by bit 13 of the ETM control register, is provided to enable the trace clock to be divided by two when half-rate clocking is required. The external pin is **CLKDIVTWOEN**, which enables the **TRACECLK** pin to be controlled. You must set bit 13, using the trace software tools, to select the clocking mode required. On a TAP reset this bit is LOW (see TAP reset on page 3-10).

Status register

ETM register 000 0100 has been added. This read-only register holds a single bit, the pending overflow status bit, that indicates that an overflow has occurred, but that a restart due to FIFO overflow reason code has not yet been issued as a result. The debugger can use this bit to check for an overflow occurring around the time of a breakpoint.

1.2.4 Changes to the programmer's model in Rev 1

Rev 1 adds a small number of features to the programmer's model. These are described in this section and fully documented in the *Embedded Trace Macrocell Specification*. The version is identified by the protocol version field of the ETM configuration code register.

Context ID

You can use the **PROCID** bus and the **PROCIDWR** signal to allow tracing of different context IDs or overlay numbers.

System options

Rev 1 of the ETM9 enables you to input system configuration options using the **SYSOPT** bus. You can use these hard-wired inputs to change the operation of your trace debug tools according to the ETM9 system configuration.

Trace port mode

You can configure the ETM9 outputs for three modes of operation:

- normal mode
- multiplexed mode
- demultiplexed mode.

These enable you to optimize the use of trace output pins in your particular ASIC design.

Instruction tracing on/off

Using the EnOnOff bit of **TraceEnable** control register 1, in conjunction with preset instruction addresses, enables you to turn tracing on and off at preset instruction addresses. You can use this to inhibit or enable tracing for individual sections of code, such as for subfunctions, ensuring that you only trace the instructions that you have to trace.

Data controlled instruction tracing

You can now enable and disable the tracing of instructions based on the data access being performed. Before Rev 1, if **TraceEnable** was based on data addresses the exact instruction tracing was unpredictable. You can now trace all instructions that access a particular address, without having to permanently enable instruction tracing.

1.2.5 Changes to the programmer's model in Rev 2

ETM9 Rev 2 includes support for Java tracing. The changes made to the programmer's model for ETM9 (Rev 2) are fully documented in the *Embedded Trace Macrocell Specification*. In summary, the changes are as follows:

ETM configuration code register

Protocol version `0011` supported.

System configuration register

Expanded to include bit 8, used to indicate **FIFOFULL** support.

Bit 7 of fifth address packet

Asserted when branching into Java state.

1.2.6 Changes to the programmer's model in Rev 2a

ETM9 Rev 2a fixes several errata encountered in previous versions. The only change to the programmer's model is as follows:

ETM configuration code register

Protocol version `0101` supported.

1.2.7 Changes to the programmer's model in r2p2

ETM9 r2p2 fixes several errata encountered in previous versions. The only change to the programmer's model is as follows:

ETM configuration code register

Protocol version `0111` supported.

Chapter 2
Accessing ETM9 Registers

This chapter describes the mechanism for programming the registers used to set up the trace and triggering facilities of the ETM9. It contains the following sections:

2.1 TAP interface

The ETM9 is programmed using a TAP interface. The structure of the TAP interface is shown in Figure 2-1.

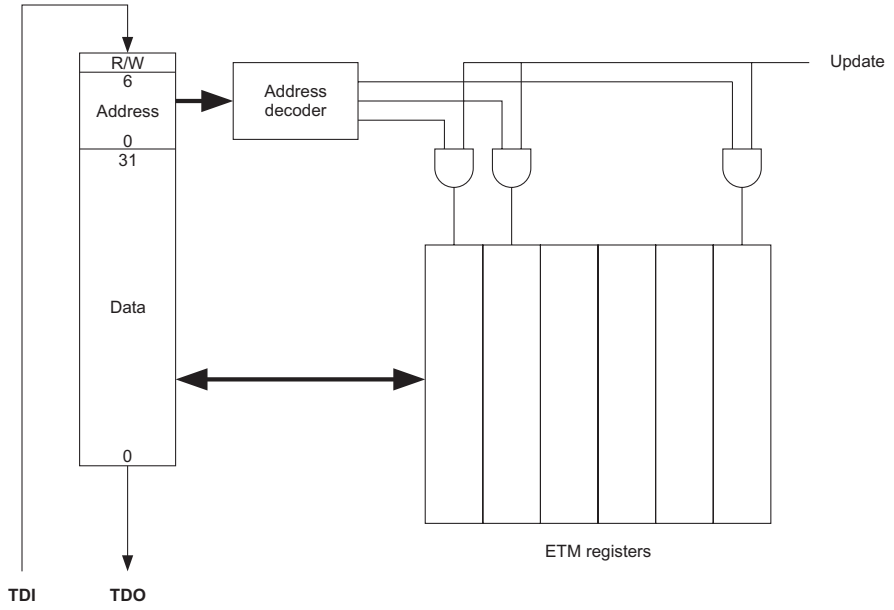


Figure 2-1 ETM9 TAP structure

The ETM9 TAP is logically part of the ARM core that it is connected to. That is, Multi-ICE only detects one TAP in a single ARM9 ETM system.

The ETM9 uses scan chain 6. For details, refer to the *Embedded Trace Macrocell Specification*.

2.2 Programming and reading ETM9 registers

All registers in the ETM9 are programmed through a JTAG interface. The interface is an extension of the ARM TAP controller, and is assigned scan chain 6.

The scan chain consists of a 40-bit shift register comprising:

- a 32-bit data field
- a 7-bit address field
- a read/write bit.

The general arrangement of the ETM9 JTAG registers is shown in Figure 2-1 on page 2-2.

The data to be written is scanned into the 32-bit data field, the address of the register into the 7-bit address field, and a 1 into the read/write bit.

A register is read by scanning its address into the address field and a 0 into the read/write bit. The 32-bit data field is ignored.

A read or a write takes place when the TAP controller enters the UPDATE-DR state.

For further details of ETM9 registers, see the *Embedded Trace Macrocell Specification*.

Chapter 3
Integrating the ETM9

This chapter describes how the ETM9 macrocell is integrated with an ARM9 microprocessor. It contains the following sections:

3.1 About integrating the ETM9

The ETM9 is designed to be connected directly to the ARM core that it is tracing, and not to the main AMBA system bus. This is because it must closely track the instructions that the ARM core is executing, and this information is only available on the ARM processor pins.

Cached and other ARM products, such as ARM946E-S, ARM966E-S, and ARM920T (Rev 1) provide a trace interface to bring out the required trace signals from the ARM core to the periphery of the macrocell. This enables an ETM9 to be connected directly to it without further modifications being required.

The trace interface is described in *ETM9 to ARM9 connection guide* on page 3-3.

A small amount of glue logic is required to connect the ETM9 to an ARM processor. For example, glue logic is required if clock-gating is implemented (see *Using the PWRDOWN output* on page 3-21), or to OR debug requests (see *Debug request output wiring* on page 3-20). This glue logic is provided for most processors in the ETM Integration Kit described in Chapter 6 *ETM Integration Testing*.

3.2 ARM interfacing

ARM interfacing is described under the following headings:

- *ETM9 to ARM9 connection guide*
- *Data buses* on page 3-9
- *Coprocessor data bus connections* on page 3-9.

3.2.1 ETM9 to ARM9 connection guide

The ETM9 port names are a mixture of those from the ARM9TDMI, ARM9E-S, and ARM9EJ-S macrocells.

Table 3-1 on page 3-4 and Table 3-2 on page 3-6 show how the ETM9 must be connected to different members of the ARM9 processor family.

Table 3-1 on page 3-4 shows ETM9 connections for the following macrocells:

- ARM9E-S
- ARM9TDMI
- ARM920T (Rev 1)
- ARM922T
- the generic trace interface on ARM946E-S and ARM966E-S.

Note

- The ARM966E-S connections shown in Table 3-1 on page 3-4 are for a (Rev 1) ARM966E-S (Rev 1) processor. If you are using a ARM966E-S (Rev 0) processor, you must tie off the unused ETM9 inputs as described in the following sections:
  - *INSTRVALID* on page 3-9
  - *Using the context ID signals* on page 3-23.
- The connections for the ARM922T (Rev 0) and ARM920T (Rev 1) processors are identical.
- The External Inputs **EXTIN[3:0]** must be synchronous to the ETM9 **CLK**

Table 3-1 Signal connections between ETM9 and ARM9 cores without Jazelle extensions

ETM9 signal name	ARM processor signal name			
	ARM9E-S	ARM9TDMI	ARM920T (Rev 1) and ARM922T	ARM946E-S and ARM966E-S (Rev 1) (generic trace interface)
ARMTDO [a]	-	-	-	-
BIGEND	**CFGBIGEND**	**BIGEND**	**ETMBIGEND**	**ETMBIGEND**
CHSD[1:0]	**CHSD[1:0]**	**CHSD[1:0]**	**ETMCHSD[1:0]**	**ETMCHSD[1:0]**
CHSE[1:0]	**CHSE[1:0]**	**CHSE[1:0]**	**ETMCHSE[1:0]**	**ETMCHSE[1:0]**
CLK [bc]	**CLK**	**GCLK**	**ETMCLOCK**	**CLK**
CLKEN [c]	**CLKEN**	**nWAIT**	**ETMnWAIT**	**ETMnWAIT**
DA[31:0]	**DA[31:0]**	**DA[31:0]**	**ETMDA[31:0]**	**ETMDA[31:0]**
DABORT	**DABORT**	**DABORT**	**ETMDABORT**	**ETMDABORT**
DBGACK	**DBGACK**	**DBGACK**	**ETMDBGACK**	**ETMDBGACK**
DBGRQ [d]	**EDBGRQ**	**EDBGRQ**	**EDBGRQ**	**EDBGRQ**
DD[31:0]	**WDATA[31:0]**	**DD[31:0]**	**ETMDD[31:0]**	**ETMWDATA[31:0]**
DDIN[31:0]	**RDATA[31:0]**	**DDIN[31:0]**	**ETMDD[31:0]**	**ETMRDATA[31:0]**
DMAS[1:0]	**DMAS[1:0]**	**DMAS[1:0]**	**ETMDMAS[1:0]**	**ETMDMAS[1:0]**
DnMREQ	**DnMREQ**	**DnMREQ**	**ETMDnMREQ**	**ETMDnMREQ**
DnRW	**DnRW**	**DnRW**	**ETMDnRW**	**ETMDnRW**
DSEQ	**DSEQ**	**DSEQ**	**ETMDSEQ**	**ETMDSEQ**
ETMEN [e]	-	-	-	-
FIFOFULL	-	-	-	**FIFOFULL** [f]
HIVECS	**CFGHIVECS**	**HIVECS**	**ETMHIVECS**	**ETMHIVECS**
IA[0] [g]	GND	GND	GND	GND
IA[31:1]	**IA[31:1]**	**IA[31:1]**	**ETMIA[31:1]**	**ETMIA[31:1]**

Table 3-1 Signal connections between ETM9 and ARM9 cores without Jazelle extensions (continued)

ETM9 signal name	ARM processor signal name			
	ARM9E-S	ARM9TDMI	ARM920T (Rev 1) and ARM922T	ARM946E-S and ARM966E-S (Rev 1) (generic trace interface)
ID15To11[15:11]	**INSTR[15:11]**	**ID[15:11]**	**ETMID15To8[15:11]**	**ETMID15To8[15:11]**
ID31To25[31:25]	**INSTR[31:25]**	**ID[31:25]**	**ETMID31To24[31:25]**	**ETMID31To24[31:25]**
IJBIT[g]	GND	GND	GND	GND
InMREQ	**InMREQ**	**InMREQ**	**ETMInMREQ**	**ETMInMREQ**
INSTREXEC	**DBGINSTREXEC**	**INSTREXEC**	**ETMINSTREXEC**	**ETMINSTREXEC**
INSTRVALID	**DBGINSTRVALID**	V_{DD}	V_{DD}	**ETMINSTRVALID**
ISEQ	**ISEQ**	**ISEQ**	**ETMISEQ**	**ETMISEQ**
ITBIT	**ITBIT**	**ITBIT**	**ETMITBIT**	**ETMITBIT**
LATECANCEL	**LATECANCEL**	**LATECANCEL**	**ETMLATECANCEL**	**ETMLATECANCEL**
nRESET [h]	**DBGnTRST**	**nTRST**	**nTRST**	**DBGnTRST**
nTRST	**DBGnTRST**	**nTRST**	**nTRST**	**DBGnTRST**
PASS	**PASS**	**PASS**	**ETMPASS**	**ETMPASS**
PROCID[31:0]	-	-	-	**ETMPROCID[31:0]**
PROCIDWR	-	-	-	**ETMPROCIDWR**
PWRDOWN [i]	-	-	**ETMPWRDOWN**	**!ETMEN**[e]
RANGEOUT[0]	**DBGRNG[0]**	**RANGEOUT[0]**	**ETMRNGOUT[0]**	**ETMRNGOUT[0]**
RANGEOUT[1]	**DBGRNG[1]**	**RANGEOUT[1]**	**ETMRNGOUT[1]**	**ETMRNGOUT[1]**
TCK	**CLK**	**TCK**	**TCK**	**CLK**
TCKEN	**DBGTCKEN**	V_{DD}	V_{DD}	**DBGTCKEN**
TDI	**DBGTDI**	**TDI**	**TDI**	**DBGTDI**
TDO	**DBGSDOUT**	**SDOUTBS**	**SDOUTBS**	**DBGSDOUT**

Table 3-1 Signal connections between ETM9 and ARM9 cores without Jazelle extensions (continued)

ETM9 signal name	ARM processor signal name			
	ARM9E-S	ARM9TDMI	ARM920T (Rev 1) and ARM922T	ARM946E-S and ARM966E-S (Rev 1) (generic trace interface)
TMS	**DBGTMS**	**TMS**	**TMS**	**DBGTMS**
ZIFIRST [g]	GND	GND	GND	GND
ZILAST [g]	GND	GND	GND	GND

a. See *TAP interface wiring* on page 3-16.
b. See *Trace signal output timing* on page 3-29.
c. See *CLK and CLKEN* on page 3-12.
d. See *Debug request output wiring* on page 3-20.
e. In cores supporting the generic trace interface, the **ETMEN** input to the core must be connected to an inverted version of **PWRDOWN**. The **ETMEN** output is not connected to the core.
f. In revisions where **FIFOFULL** is supported.
g. See *Memory interface signals on ETM9 (Rev 2 and above)* on page 3-11.
h. See *Clocks and resets* on page 3-12.
i. See *Using the PWRDOWN output* on page 3-21.

Table 3-2 shows ETM9 connections for:

- ARM9EJ-S cores
- the generic trace interface on ARM926EJ-S.

Table 3-2 ETM9 and Jazelle-enabled ARM9 cores signal connections

ETM9 signal name	ARM processor signal name	
	ARM9EJ-S	ARM926EJ-S (generic trace interface)
ARMTDO [a]	-	-
BIGEND	**CFGBIGEND**	**ETMBIGEND**
CHSD[1:0]	**CHSD[1:0]**	**ETMCHSD[1:0]**
CHSE[1:0]	**CHSE[1:0]**	**ETMCHSE[1:0]**
CLK [bc]	**CLK**	**CLK**
CLKEN [c]	**CLKEN**	**ETMnWAIT**
DA[31:0]	**DA[31:0]**	**ETMDA[31:0]**

Table 3-2 ETM9 and Jazelle-enabled ARM9 cores signal connections (continued)

ETM9 signal name	ARM processor signal name	
	ARM9EJ-S	ARM926EJ-S (generic trace interface)
DABORT	DABORT	ETMDABORT
DBGACK	DBGACK	ETMDBGACK
DBGRQ [d]	EDBGRQ	EDBGRQ
DD[31:0]	WDATA[31:0]	ETMWDATA[31:0]
DDIN[31:0]	RDATA[31:0]	ETMRDATA[31:0]
DMAS[1:0]	DMAS[1:0]	ETMDMAS[1:0]
DnMREQ	DnMREQ	ETMDnMREQ
DnRW	DnRW	ETMDnRW
DSEQ	DSEQ	ETMDSEQ
ETMEN [e]	-	-
FIFOFULL	-	FIFOFULL
HIVECS	CFGHIVECS	ETMHIVECS
IA[0] [f]	IA[0]	ETMIA[0]
IA[31:1]	IA[31:1]	ETMIA[31:1]
ID15To11[15:11]	INSTR[15:11]	ETMID15To8[15:11]
ID31To25[31:25]	INSTR[31:25]	ETMID31To24[31:25]
IJBIT [f]	IJBIT	ETMIJBIT
InMREQ	InMREQ	ETMInMREQ
INSTREXEC	DBGINSTREXEC	ETMINSTREXEC
INSTRVALID	DBGINSTRVALID	ETMINSTRVALID
ISEQ	ISEQ	ETMISEQ
ITBIT	ITBIT	ETMITBIT
LATECANCEL	LATECANCEL	ETMLATECANCEL
nRESET [g]	DBGnTRST	DBGnTRST

Table 3-2 ETM9 and Jazelle-enabled ARM9 cores signal connections (continued)

ETM9 signal name	ARM processor signal name	
	ARM9EJ-S	ARM926EJ-S (generic trace interface)
nTRST	**DBGnTRST**	**DBGnTRST**
PASS	**PASS**	**ETMPASS**
PROCID[31:0]	-	**ETMPROCID[31:0]**
PROCIDWR	-	**ETMPROCIDWR**
PWRDOWN [h]	-	**!ETMEN**[e]
RANGEOUT[0]	**DBGRNG[0]**	**ETMRNGOUT[0]**
RANGEOUT[1]	**DBGRNG[1]**	**ETMRNGOUT[1]**
TCK	**CLK**	**CLK**
TCKEN	**DBGTCKEN**	**DBGTCKEN**
TDI	**DBGTDI**	**DBGTDI**
TDO	**DBGSDOUT**	**DBGSDOUT**
TMS	**DBGTMS**	**DBGTMS**
ZIFIRST [f]	**ZIFIRST**	**ETMZIFIRST**
ZILAST [f]	**ZILAST**	**ETMZILAST**

a. See *TAP interface wiring* on page 3-16.
b. See *Trace signal output timing* on page 3-29.
c. See *CLK and CLKEN* on page 3-12.
d. See *Debug request output wiring* on page 3-20.
e. In cores supporting the generic trace interface, the **ETMEN** input to the core must be connected to an inverted version of **PWRDOWN**. The **ETMEN** output is not connected to the core.
f. See *Memory interface signals on ETM9 (Rev 2 and above)* on page 3-11.
g. See *Clocks and resets* on page 3-12.
h. See *Using the PWRDOWN output* on page 3-21.

Note

Earlier revisions of ETM9 (Rev 0a and below) also included the **DMORE** signal. This signal is no longer required.

3.2.2 INSTRVALID

If you are using an ARM processor in which this signal is not available, such as ARM9TDMI, ARM920T, or ARM966E-S (Rev 0), you must tie this ETM input HIGH.

3.2.3 Data buses

The ETM requires visibility of any data from either the memory system or coprocessors to select between the two data buses based on the transfer direction, defined by **DnRW**.

The **DDIN** bus is sampled when:

- **DnRW** is LOW (for `LDC` data transfers, `MRC` register transfers, and `LDR/LDM` data transfers)
- **DnMREQ** and **PASS** are both LOW (for `LDC` and `STC` data transfers).

The **DD** bus is sampled at all other times, such as for `STR/STM` transfers.

When ARM coprocessors are present in the system, you must ensure that a copy of the coprocessor data transferred to and from memory is seen on the **DD** bus. See *Coprocessor data bus connections* for details.

3.2.4 Coprocessor data bus connections

Figure 3-1 on page 3-10 shows the coprocessor data bus connections for an ARM9E-S-based system having an ETM9.

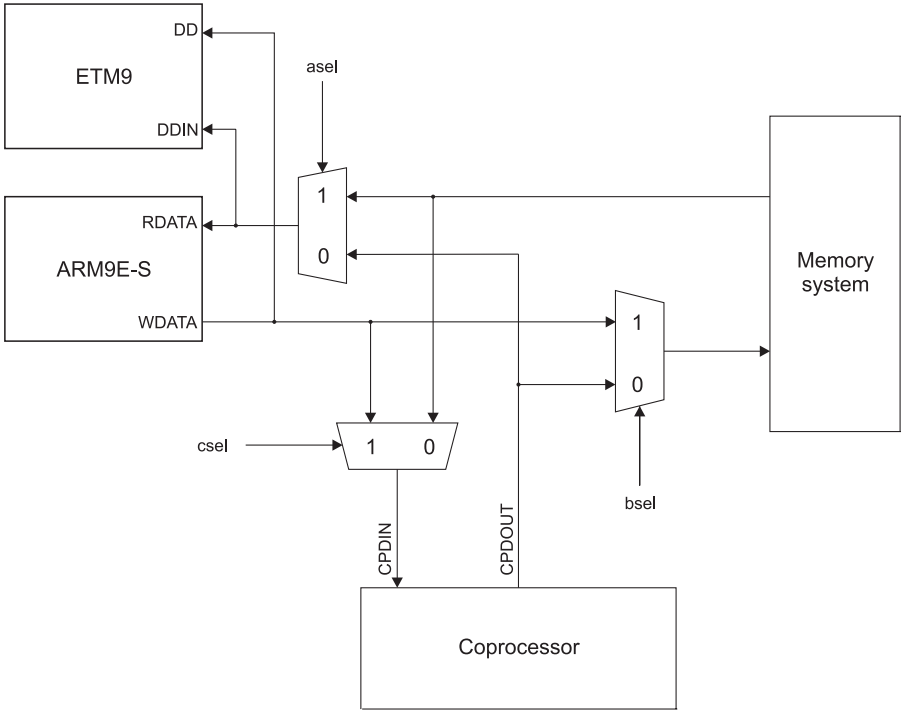


Figure 3-1 Coprocessor data bus connections

The logic that drives asel, bsel, and csel from the relevant ARM9TDMI, ARM9E-S, or ARM9EJ-S pins is as follows:

```
on RISING CLK
    asel = not(DnMREQ and DSEQ) and (not DnRW)
    bsel = (not DnMREQ) and (not PASS)
    csel = DnMREQ and DSEQ
```

You do not have to make any coprocessor data bus connections for the following processors because the necessary support logic is included in the ARM macrocell:

- ARM920T
- ARM922T
- ARM926EJ-S
- ARM946E-S
- ARM966E-S.

Note

The **RDATA** enable term, asel, is specially constructed to select the coprocessor output data during MRC and STC operations. This is to allow the connection of the ARM ETM module to the **RDATA** and **WDATA** buses of the ARM9E-S or the ARM9EJ-S cores while still allowing tracing of MRC and STC data.

To ensure that tracing of coprocessor instructions functions correctly it is essential that:

- the ETM is directly connected to the data buses of the ARM9 processor core, as described in Table 3-2 on page 3-6
- all output data from the coprocessor is visible on the data input bus of the processor.

3.2.5 Memory interface signals on ETM9 (Rev 2 and above)

The Java trace support in ETM9 (Rev 2 and above) includes the following additional memory interface signals:

- **IA[0]**
- **IJBIT**
- **ZIFIRST**
- **ZILAST**.

If these signals are not available on the trace port interface of the core, you must tie the unused ETM inputs LOW.

3.3 Clocks and resets

The ETM9 has two sets of clock, clock enable, and reset inputs:

Main system controls:
- **CLK**
- **CLKEN**
- **nRESET**.

Scan chain controls:
- **TCK**
- **TCKEN**
- **nTRST**.

These are described under the following headings:
- *CLK and CLKEN*
- *ETM reset* on page 3-13
- *TCK and TCKEN* on page 3-14
- *TAP reset* on page 3-14.

3.3.1 CLK and CLKEN

CLK is the master clock for the ETM9. The ETM9 only uses the rising edge of **CLK,** so **CLK** is the inverse of **GCLK** when used with an ARM9TDMI, and **CLK** when used with an ARM9E-S or an ARM9EJ-S macrocell.

CLKEN is a synchronous enable for **CLK**. All ARM9 interface signals are sampled on the rising edge of **CLK** when **CLKEN** is HIGH, and all trace port outputs are generated off the rising edge of **CLK**.

Note

The trace port outputs can change on clock cycles when **CLKEN** is LOW.

If you dynamically stretch or gate the **CLK** signal, you cannot infer elapsed time from the number of elapsed cycles when cycle-accurate tracing is enabled.

Connecting to an ARM9E-S or an ARM9EJ-S macrocell

You must connect the ETM9 signals, **CLK** and **CLKEN**, directly to the same signals that drive **CLK** and **CLKEN** on the ARM9E-S or the ARM9EJ-S macrocell.

Connecting to an ARM9TDMI macrocell

When connecting ETM9 directly to an ARM9TDMI macrocell, you must make the following connections:

- connect **CLK** to the inverse of **GCLK**
- connect **CLKEN** to **nWAIT** inputs.

Connecting to an ARM920T or ARM922T macrocell

When connecting ETM9 to an ARM920T macrocell, or ARM922T macrocell you must make the following connections:

- connect **CLK** to the inverse of **ETMCLOCK**
- connect **CLKEN** to the **ETMnWAIT** ARM920T output.

The signals in the ARM920T ETM interface are driven off the rising edge of **ETMCLOCK**. However, the signals cannot be captured off the rising edge of **ETMCLOCK** because **ETMCLOCK** is exported from the ARM920T and is subject to further insertion delays within the ETM. These timing problems are usually solved by inverting **ETMCLOCK**, but timing analysis of the system is still required.

Connecting to an ARM946E-S, an ARM966E-S, or an ARM926EJ-S macrocell

When connecting ETM9 to an ARM946E-S, an ARM966E-S, or an ARM926EJ-S macrocell, you must make the following connections:

- connect **CLK** to the same signal that drives the core **CLK** input
- connect **CLKEN** to the **ETMnWAIT** processor output.

3.3.2 ETM reset

nRESET resets all of the ETM9 state, with the exception of the ETM control register, and flushes the trace FIFO. You can connect **nRESET** to the same reset as the processor. However, this prevents trace being used during a warm reset of the ARM9. ARM Limited strongly recommends that you use the TAP reset, **nTRST/DBGnTRST**, to reset the ETM9 state.

In systems where **CLK** and **TCK** are asynchronous, **nTRST** must be synchronized to **CLK**. You can do this using the arrangement shown in Figure 3-2 on page 3-14.

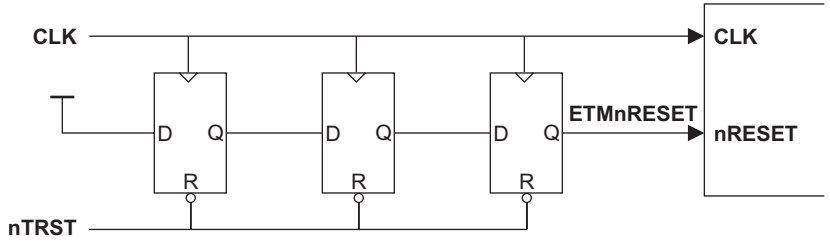


Figure 3-2 Synchronizing reset

Synchronizing **nTRST** to **CLK** enables **nTRST** to reset the ETM even when **CLK** is running slowly, or is stopped.

3.3.3 TCK and TCKEN

TCK is the master clock for the ETM9 JTAG interface. **TCKEN** is a synchronous clock enable signal.

In a system with an ARM9TDMI, ARM920T, or ARM922T macrocell, **TCK** is generally a free-running clock, asynchronous to **CLK**. In these systems you must tie **TCKEN** HIGH.

In systems containing the following processors, a single clock is used as both the main system clock and the JTAG clock:

- ARM9E-S
- ARM946E-S
- ARM966E-S
- ARM9EJ-S
- ARM926EJ-S.

In these systems you must connect the single main clock to both **CLK** and **TCK**. You can then use **TCKEN** to control the JTAG interface.

ETM9 is designed to function with fully asynchronous **CLK** and **TCK** inputs. Synchronizing logic is included in the design to prevent metastability problems between the two clock domains when running with asynchronous clocks.

3.3.4 TAP reset

nTRST is the TAP controller reset, used to asynchronously reset the TAP controller. It also resets the ETM control register.

Note

ARM Limited recommends that either:

- **TCK** is inhibited when **nTRST** is deasserted
- **nTRST** is deasserted synchronously to **TCK**.

This ensures that the TAP state machine cannot enter an unknown state because of reset hold violations. You must hold **TMS** HIGH during a TAP reset.

3.4 TAP interface wiring

Both the ARM9 microprocessor and the ETM9 provide scan chain expansion inputs. These are:

- **SDOUTBS** on the ARM9TDMI macrocell (**DBGSDOUT** on the ARM9E-S macrocell and the ARM9EJ-S macrocell)
- **ARMTDO** on ETM9.

In each case this input is routed through to **TDO** when an unimplemented scan chain is selected. This enables the ARM9 and ETM9 TAP controllers to run in parallel, with a single **TDO** output.

ARMTDO is connected to **TDO** when the ETM scan chain, scan chain 6, is not selected.

ARM Limited recommends connectivity as shown in Figure 3-3.

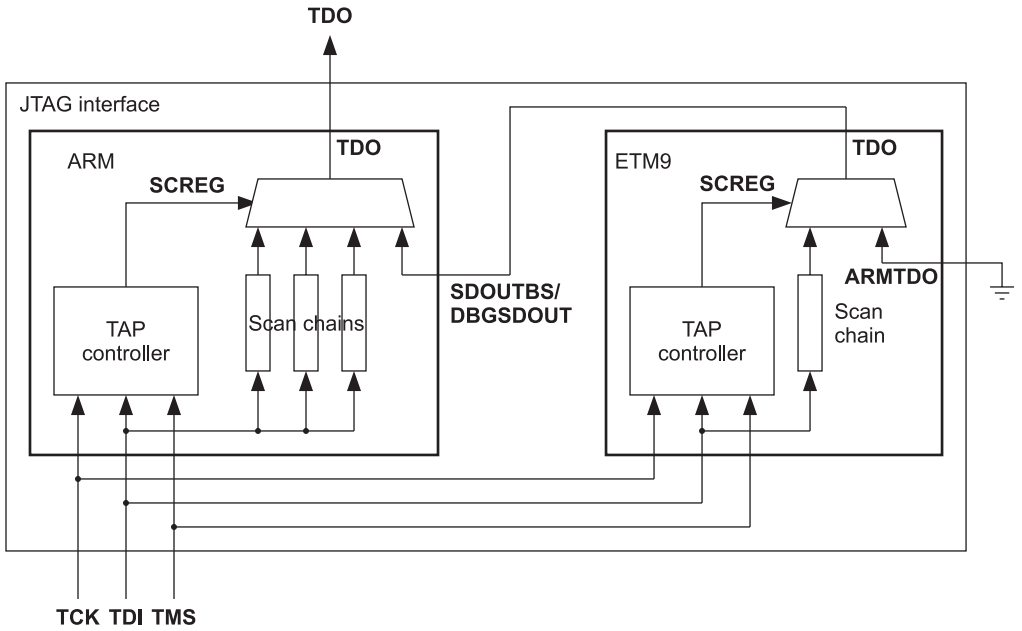


Figure 3-3 TAP interface structure

Note

For clarity, **nTRST** is omitted from figures relating to the TAP interface. You must connect **nTRST** to all TAPs on the chip. See the *Multi-ICE User Guide* for details.

If your ASIC includes a further scan chain controlled by the ARM TAP controller, then the **TDO** of this scan chain can be connected into the otherwise unused **ARMTDO** input on the ETM9. This is shown in Figure 3-4.

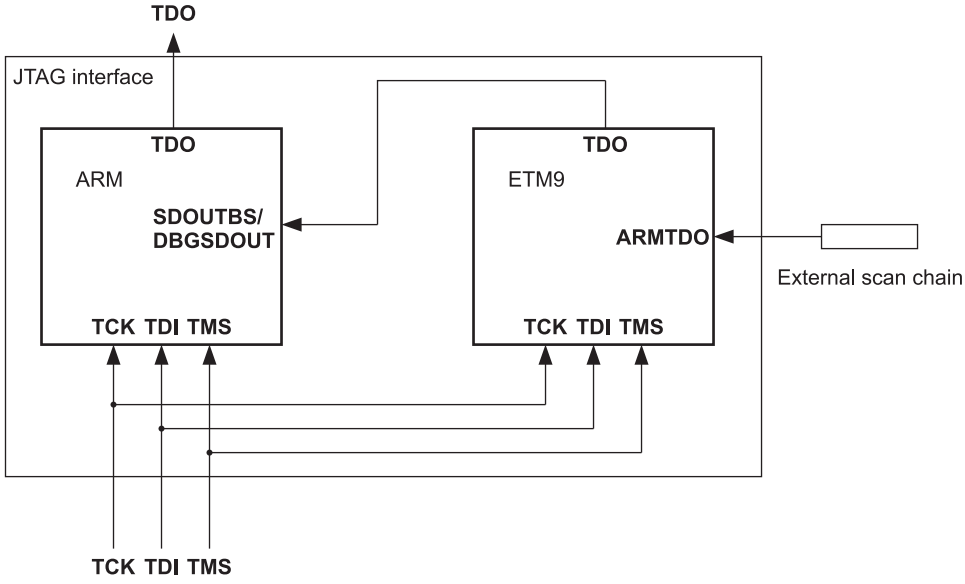


Figure 3-4 Using ETM9 and ARM9 with an external scan chain

3.4.1 IEEE 1149.1 compatibility

The **TDO** output from the ETM changes on the rising edge of **TCK**, when **TCKEN** is HIGH. For ARM9TDMI and ARM920T systems, you must add an external falling edge D-type flip-flop, as shown in Figure 3-5 on page 3-18, to ensure full compatibility with IEEE 1149.1.

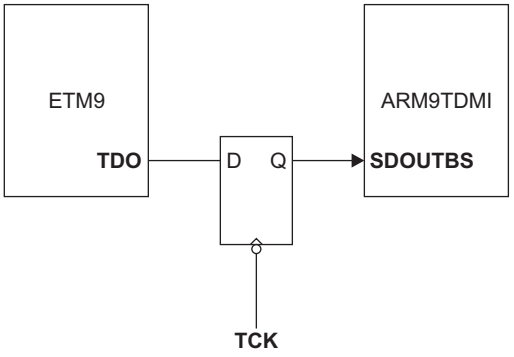


Figure 3-5 TDO output retiming for IEEE 1149.1 compatibility

You do not have to do anything for systems based on the ARM9E-S macrocell or the ARM9EJ-S macrocell, because the synchronization and adaptive clocking logic ensures that this register is not required.

You must take care if you add this register and use the **ARMTDO** input to the ETM. This input must change around the rising edge of **TCK**. If it does not, it is incorrectly delayed by one cycle.

3.4.2 Multiprocessor TAP structure

If you want your multiprocessor-compatible run control products, such as Multi-ICE, to work correctly when used with more than one ARM processor on a chip, ARM Limited recommends that you connect the processors as a serial TAP structure. The presence of an ETM9 on any or all of the ARM processors does not affect this. The TAP structure for a dual-processor ARM processor system is shown in Figure 3-6 on page 3-19.

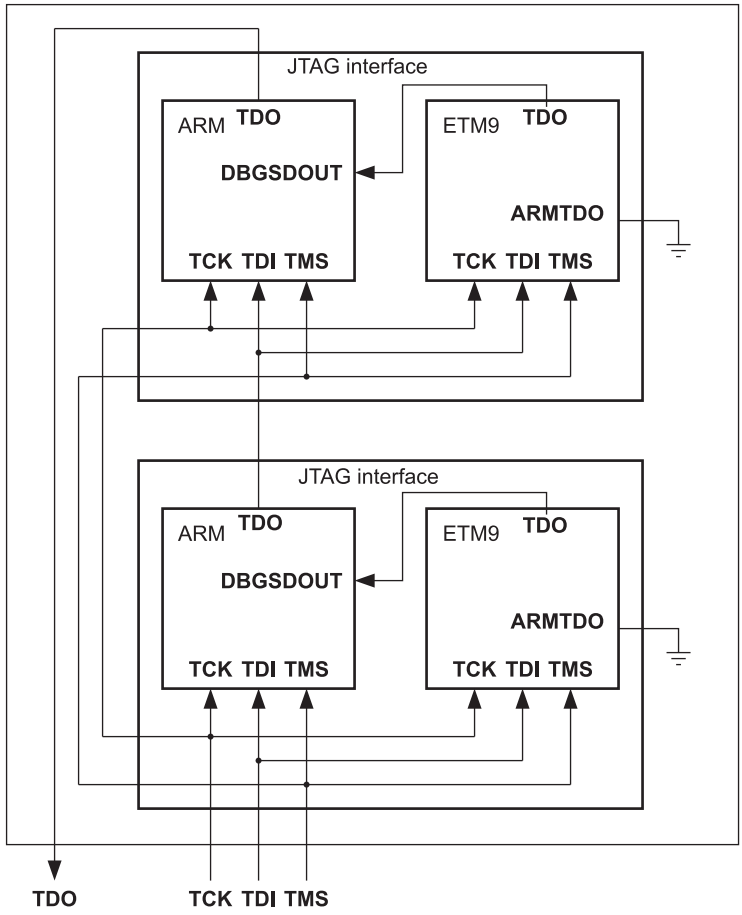


Figure 3-6 Multiprocessor TAP structure

Note

For clarity, **nTRST** is omitted from figures relating to the TAP interface. You must connect **nTRST** to all TAPs on the chip. See the *Multi-ICE User Guide* for details.

3.5 System control signals

System control signal interfacing is described under the following headings:

- *Debug request output wiring*
- *Using the PWRDOWN output* on page 3-21
- *FIFOFULL* on page 3-23
- *Using the context ID signals* on page 3-23
- *Using the system options bus* on page 3-24.

3.5.1 Debug request output wiring

When the trigger condition occurs, you can set the ETM9 to assert **DBGRQ** until **DBGACK** is observed under the control of bit 9 in the ETM control register.

Note

ARM9 processors take at least one cycle to respond to **EDBGRQ**. This means that the ARM processor can execute a few instructions after the trigger condition is detected but before the system has stopped. Some debug tools can report an unrecognized breakpoint as a result.

You must connect the **DBGRQ** output of the ETM9 to the **EDBGRQ** input of the ARM processor. If this input is already in use, for example a **DBGRQ** input is present on the device, the **DBGRQ** signals must be ORed together as shown in Figure 3-7.

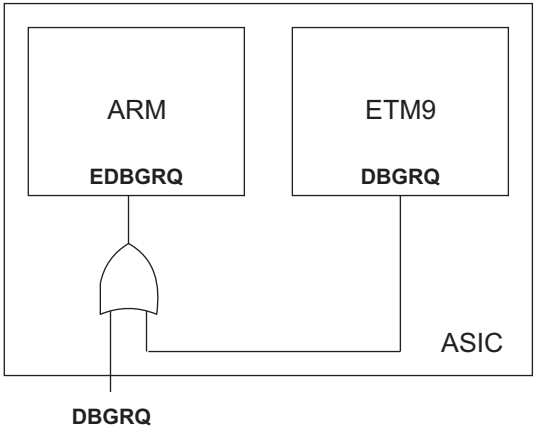


Figure 3-7 Combining DBGRQ inputs

3.5.2 Using the PWRDOWN output

The ETM9 provides an output called **PWRDOWN**. When HIGH this indicates that the ETM is not currently enabled, so you can stop the **CLK** input and hold the other ETM9 signals stable. You can use this to reduce system power consumption when trace is not being used. When a TAP reset (**nTRST**) occurs, **PWRDOWN** is automatically forced HIGH until the ETM9 control register has been programmed.

You can use the **PWRDOWN** output directly to gate the ETM9 **CLK** input. This is shown in Figure 3-8.

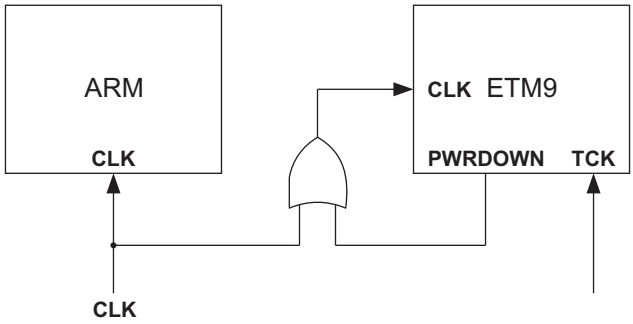


Figure 3-8 Clock gating the ETM9

The **PWRDOWN** signal is changed synchronously to **TCK**. Because **PWRDOWN** changes many cycles before trace is enabled, this does not cause any metastability problems if you use **PWRDOWN** to gate the ETM9 clock. If using **PWRDOWN** in this way causes problems with static timing analysis, you can synchronize **PWRDOWN** to **CLK** before using it to gate the ETM9 clock.

The **PWRDOWN** output is controlled by the ARM debug tools, and is automatically cleared at the start of a debug session.

The ARM920T (Rev 1), and ARM922T macrocells implement this logic internally by providing an **ETMPWRDOWN** input to the trace interface, causing the clock and data outputs to the ETM to be stopped.

You must connect the **PWRDOWN** output from ETM9 to the **ETMPWRDOWN** input to these cores, as shown in Figure 3-9 on page 3-22.

Figure 3-9 Connecting PWRDOWN to cores that do not support the generic trace interface

The following cores support the generic trace interface:

- ARM926EJ-S
- ARM946E-S
- ARM966E-S.

On these cores, the trace outputs are prevented from switching using an **ETMEN** core input. You must tie the **ETMEN** input to the inverted **PWRDOWN** output from ETM9, as shown in Figure 3-10.

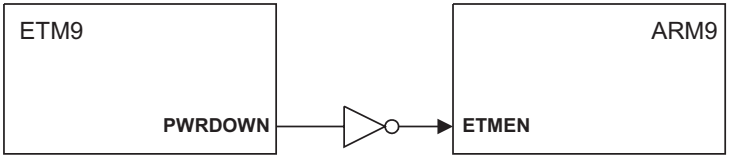


Figure 3-10 Connecting PWRDOWN to cores that support the generic trace interface

For more information, see the Technical Reference Manual for your processor.The ETMEN output from ETM9 must only be used to disable the trace port, not to disable ETM9 itself. See *Dual-processor tracing* on page 3-27 for more information. Table 3-3 lists the possible combinations of ETMEN and PWRDOWN.

Table 3-3 ETMEN and PWRDOWN combinations

ETMEN	PWRDOWN	Description
0	1	ETM in low power state. All functionality disabled.
0	0	ETM trace port disabled. Other ETM functionality, such as **EXTOUT** generation and assertion of **DBGRQ** on trigger, continues as normal.
1	0	All ETM functionality enabled.
1	1	UPREDICTABLE

3.5.3 FIFOFULL

This signal changes on the rising edge of **CLK** and is active HIGH. When asserted it indicates that:

- the trace tools user has enabled the ETM FIFO full detection
- the FIFO currently has less than a programmed number of bytes of space available.

You can use **FIFOFULL** to stall the ARM core, so that more trace data is not generated until the FIFO has drained. It is recommended that you implement this by controlling the **CLKEN** input to the ARM core, rather than gating the clock. If the clock is gated there is a risk of system lock-up, because stopping the clock prevents the FIFO from draining, and prevents **FIFOFULL** from being deasserted.

If **CLKEN** is supported you can increase the accuracy of the tracing by slowing down the processor when the trace port bandwidth is exceeded. This enables you to slow down non real-time areas of code while critical regions remain unaffected. The *Embedded Trace Macrocell Specification* describes how this is achieved in more detail. Briefly however, you specify, within the ETM, the address regions in which **FIFOFULL** can be asserted.

If the system designer is not able to support the use of this signal no harm results, even if the **FIFOFULL** logic inside the ETM is programmed and enabled, because the logic does not have any direct effect on the behavior of the ETM. However, if **FIFOFULL** is not used, there is a risk of some trace data being lost while the FIFO drains.

Note

To maintain interrupt response time in the system, you might have to override **FIFOFULL** assertion when **nIRQ** and/or **nFIQ** are asserted.

3.5.4 Using the context ID signals

ETM9 (Rev1) supports tracing of context IDs or overlay identifiers, using the **PROCID[31:0]** and **PROCIDWR** signals.

Note

Context ID was previously known as process ID. This has been changed to avoid confusion with the *Fast Context Switch Extensions* (FCSE) field, sometimes referred to as the FCSE process ID.

A change in Context ID is indicated by asserting **PROCIDWR** in the same cycle as the new Context ID is provided on **PROCID[31:0]** as shown in Figure 3-11 on page 3-24.

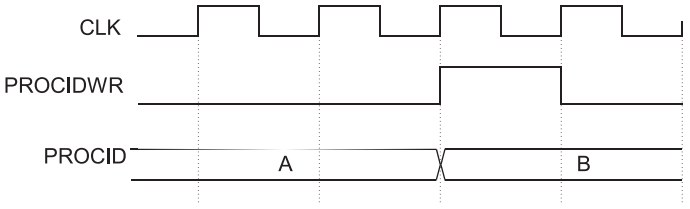


Figure 3-11 Change in Context ID

The following processors provide a 32-bit register in the system control coprocessor (CP15) that contains the current context ID:

- ARM926EJ-S
- ARM946E-S
- ARM966E-S (Rev1).

The macrocell has output signals that correspond to this register called **ETMPROCID[31:0]** and **ETMPROCIDWR**. These must be connected directly to the **PROCID[31:0]** and **PROCIDWR** ETM inputs.Earlier ARM cores do not provide these signals directly, but it is possible for the context ID register to be put into a peripheral or another coprocessor, subject to software toolkit compatibility.

Note

If you are using a processor that does not provide these signals, you must tie the unused ETM9 inputs LOW.

Future versions of the ARM trace debug tools will support the context ID extensions, to allow tracing of dynamically loaded memory and overlay systems. See Chapter 7 *Software Considerations for Trace* for more details.

3.5.5 Using the system options bus

The system options bus is a 9-bit input bus called **SYSOPT[8:0]**. This is provided on ETM (Rev 1) and above. It enables you to specify whether certain trace features, such as half-rate clocking, are implemented on the ASIC. You must tie each of the bits of the

bus to either GND or V_{DD}, depending on the features supported. The trace debug tools must read the state of this input bus using the JTAG interface, and adapt the user options offered accordingly. The bit meanings of the **SYSOPT** bus are shown in Table 3-4.

Table 3-4 SYSOPT bus settings

Bit number	Description
8	If HIGH, system stalling using **FIFOFULL** is supported
7	If HIGH, demultiplexed trace data format is supported
6	If HIGH, multiplexed trace data format is supported
5	If HIGH, normal trace data format is supported
4	If HIGH, full-rate clocking is supported
3	If HIGH, half-rate clocking is supported
2:0	Maximum port width supported:000 = 4-bit only001 = 4/8-bit only010 = 4/8/16-bit

Note

If the correct input is not supplied to the **SYSOPT** bus, the operation of the trace tools might be unreliable.

3.6 Trace port interfacing

Trace port interfacing is described under the following headings:

- *Trace port logic*
- *Single-processor tracing*
- *Dual-processor tracing* on page 3-27
- *Trace signal output timing* on page 3-29
- *PCB design guidelines* on page 3-31.

See *Modes of operation of the trace port* on page 3-32 for details of trace port operation.

3.6.1 Trace port logic

The trace information from the ETM9 is broadcast on the following signals:

- **PIPESTAT**
- **TRACESYNC**
- **TRACEPKT**.

Note

During simulation, Xs might be seen **on TRACESYNC** and **TRACEPKT**, even when tracing is enabled. This is normal behavior.

In addition, three configuration signals are provided:

- **ETMEN**
- **PORTSIZE**
- **PORTMODE**.

You can use these to configure the external logic connected to the trace port, under the control of the debugger.

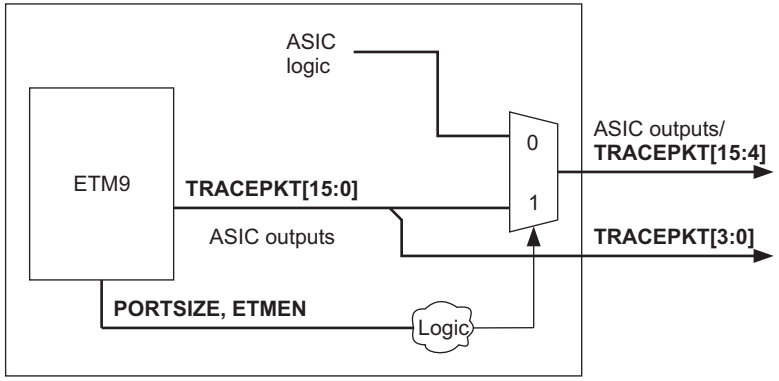
3.6.2 Single-processor tracing

Some chips might not dedicate 16 pins to the **TRACEPKT** bus. Under some circumstances you might be able to reuse miscellaneous output signals from the chip as trace port pins. To enable this the ETM9 has the following outputs:

- **ETMEN**
- **PORTSIZE[2:0]**.

Example 3-1 on page 3-27 shows one way in which the **TRACEPKT** pins can be shared with the ASIC pins.

Example 3-1 Re-using TRACEPKT pins



You can use the **PORTSIZE** and **ETMEN** signals to control on-chip logic to select between the normal ASIC output signals and the ETM9 trace port signals. This enables you to control the port width of the trace, and the number of pins used, from the debugger.

At reset, the ETM9 is disabled (**ETMEN** LOW) and a 4-bit port is selected (**PORTSIZE** = 000). This ensures that normal operation of the ASIC is undisturbed.

When the debug session starts, the debug tools can control **ETMEN** and **PORTSIZE** by programming the ETM control register.

The configuration in Example 3-1 does not multiplex the **TRACEPKT[3:0]** or other trace port signals. This provides a minimum-size trace port at all times. In addition, routing for the normal ASIC logic outputs on the board is prevented from compromising the carefully-controlled routing requirements for the high-speed trace port signals (**TRACECLK** in particular).

3.6.3 Dual-processor tracing

Where there are multiple ARM processors on a single chip, it is recommended that each ARM processor has its own dedicated ETM.

The principle of controlling the port width, described in *Single-processor tracing* on page 3-26, can be extended to support dual-processor systems without dedicating a large number of pins to the trace signals.

The recommended dual trace configuration uses 21 pins on the ASIC, because this matches the 20 data pins and 1 clock pin defined in the trace connector specification. These pins are configured as 20 data pins and a single clock pin (assuming that both processors are clocked off a single clock).

This enables a number of configurations. Possible configurations for a single processor are shown in Table 3-5.

Table 3-5 Single-processor configurations

TRACEPKT	PIPESTAT	TRACESYNC	Total
16 trace packet	3 status	1 sync	20 data pins
8 trace packet	3 status	1 sync	12 data pins
4 trace packet	3 status	1 sync	8 data pins

You can, therefore, set a single trace port to enable the configurations shown in Table 3-6.

Table 3-6 Dual-processor trace port configurations

Processor A	Processor B
20 data	No trace
12 data	8 data
8 data	12 data
No trace	20 data

Pseudo-HDL to implement this is as follows:

```
if (PORTSIZE_B = 21)
    TRACE_DATA <= {PIPESTAT_B, TRACESYNC_B, TRACEPKT_B[15:0]}
else if (PORTSIZE_A = 13) and (PORTSIZE_B = 9)
    TRACE_DATA <= {PIPESTAT_B, TRACESYNC_B, TRACEPKT_B[3:0],
                   PIPESTAT_A, TRACESYNC_A, TRACEPKT_A[7:0]}
else if (PORTSIZE_A = 9) and (PORTSIZE_B = 13)
    TRACE_DATA <= {PIPESTAT_A, TRACESYNC_A, TRACEPKT_A[3:0],
                   PIPESTAT_B, TRACESYNC_B, TRACEPKT_B[7:0]}
else
    -- select A as the "master" for all other combinations.
    TRACE_DATA <= {PIPESTAT_A, TRACESYNC_A, TRACEPKT_A[15:0]}
end if
```

The *Embedded Trace Macrocell Specification* documents the target system connector pin allocations for single and dual-processor configurations. Support for the dual-processor pinouts is dependent on the debug tools and the TPA.

It is not recommended that you connect a single ETM9 to multiple ARM9 processors, because there is no general mechanism available to control the logic that selects which processor is connected to the single ETM.

3.6.4 Trace signal output timing

The trace connection to the TPA requires a clock, **TRACECLK,** to be exported from the ASIC. This is not generated by the ETM9, but must be generated by the system implementer. It is essential that you balance the clock to provide sufficient hold time on the trace data signals. The required hold times are defined in the *Embedded Trace Macrocell Specification*. It is essential that you maintain these hold times to guarantee reliable trace functionality.

It is recommended that the trace data signals are shifted by a clock phase from **TRACECLK**. This ensures that, on **TRACECLK** transitions, the trace data signals are stable, with a sufficient setup and hold time around the clock edge. Most TPAs require approximately 3ns of data valid time and a hold time in the range 1 to 2ns, for reliable acquisition.

The ETM also supports a half-rate clocking mode, controlled by the **CLKDIVTWOEN** ETM9 output. When asserted, you should drive **TRACECLK** from the ETM clock, **CLK**, divided by two. When the debugger selects this mode, it also tells the TPA that it must sample the trace data signals on both edges of the clock, instead of only the rising edge.

Note

You do not have to implement half-rate clocking, and for low-speed systems (for example, less than 50MHz) the normal clocking mode is adequate. The primary purpose of half-rate clocking is to reduce the signal transition rate on the **TRACECLK** pin of the ASIC. This might be necessary to reduce electrical interference, to maintain **TRACECLK** signal integrity when using low drive strength pads, or for systems with very high clock speeds.

Figure 3-12 on page 3-30 shows an example circuit that implements both half-rate clocking and shifting of the trace data with respect to the clock.

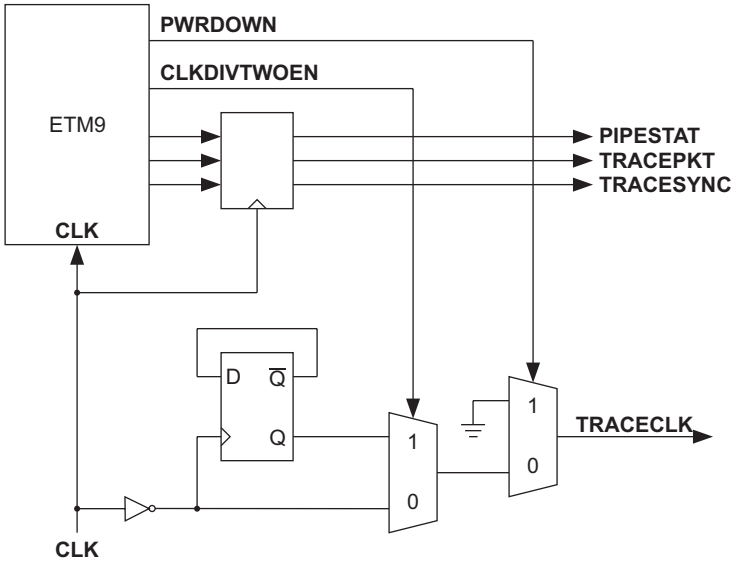


Figure 3-12 Trace output circuit with inverted clock

If your design flow does not enable you to invert the clock, you can also use falling edge D-types to retime the trace data signals, as shown in Figure 3-13.

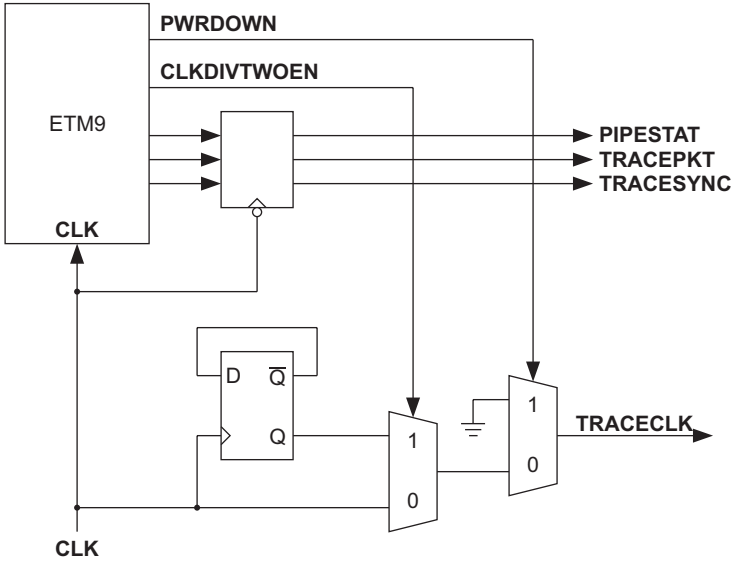


Figure 3-13 Trace output circuit using falling-edge D-types

It is recommended that you analyze carefully the timing of the trace data and clock signals to ensure the optimum setup and hold timing on the pins of the ASIC. It is also advisable to do detailed simulations of the output pads, package (for example, bond wires), PCB tracking, and logic analyzer loads to ensure the setup and hold times and signal integrity are met for the analyzer.

3.6.5 PCB design guidelines

See Chapter 8 *Physical Trace Port Signal Guidelines* for information about output pad selection and PCB design, including:

- trace signal termination
- PCB track lengths
- pad drive strength.

3.7 Modes of operation of the trace port

Normal mode tracing is the only mode of operation directly supported by Rev 0/0a of the ETM.The **PORTMODE** output bus, which is available in ETM9 Rev.1 and later revisions, provides a copy of the contents of bits [17:16] of the ETM control register. This bus enables the optional on-chip logic (the EtmMuxDemux block) to configure how the trace port signals from the ETM (**PIPESTAT**, **TRACEPKT**, and **TRACESYNC**) are mapped onto the trace port pins of the ASIC. The three modes of operation are described in the following sections:

- *Normal trace port signals*
- *Multiplexed trace port signals* on page 3-33
- *Demultiplexed trace port signals* on page 3-34.

Note

The HDL for configuring the trace port in any one of these three modes of operation is not part of the ETM9 macrocell HDL. It is provided in an additional file `ETM9/EtmMuxDemux.v` in the ETM9 HDL directory. To include the `EtmMuxDemux` block with your ETM9 refer to Appendix B *Integrating the EtmMuxDemux Block into ETM9*.

Table 3-3 on page 3-22 shows the behavior of **TRACECLK** in the available modes.

Table 3-7 TRACECLK behavior in available modes

PORTMODE	CLKDIVTWO	TRACECLK	Data capture
00 (Normal)	0	**CLK** speed	Rising edge of **TRACECLK**
00 (Normal)	1	Half **CLK** speed	Both edges of **TRACECLK**
01 (Multiplexed)	0	**CLK** speed	Both edges of **TRACECLK**
10 (Demultiplexed)	0	Half **CLK** speed	Rising edge of **TRACECLK**
10 (Demultiplexed)	1	Quarter **CLK** speed	Both edges of **TRACECLK**

3.7.1 Normal trace port signals

Normal mode tracing is the only mode of operation directly supported by Rev 0/0a of the ETM. Both normal and half-rate clocking can be supported in this mode, and for very high speed designs (greater than 100MHz) half-rate clocking is recommended to maintain the signal integrity of the clock.

3.7.2 Multiplexed trace port signals

This mode of operation multiplexes two trace data signals onto a single trace output pin. This means that the TPA must capture the data on both edges of **TRACECLK**. This scheme is only recommended for low-frequency designs (for example, less than 50MHz). This is because it is difficult to maintain the required setup and hold between **TRACECLK** and the trace data signals to the TPA.Half-rate clocking is not supported in this mode, because it already relies on the TPA capturing the state of the trace data pins twice per trace clock cycle.

The *Embedded Trace Macrocell Specification* provides the trace connector pinout for this mode of operation.

You must pair up the trace signals as shown in Table 3-8. Each row contains a separate pair of signals. One signal occurs on the rising edge of **TRACECLK** and the other on the falling edge.

Table 3-8 Paired signals in a multiplexed trace port connector

Connector groups			Signals sampled on the rising edge of TRACECLK (B, D, F)	Signals sampled on the falling edge of TRACECLK (A, C, E)
All signals are paired for a 10-pin trace port connector	These signals are paired for a 4-pin trace port connector	These pins are paired for a 6-pin trace port connector	**PIPESTAT[0]**	**TRACESYNC**
			PIPESTAT[1]	**TRACEPKT[1]**
			PIPESTAT[2]	**TRACEPKT[2]**
			TRACEPKT[0]	**TRACEPKT[3]**
			TRACEPKT[4]	**TRACEPKT[5]**
			TRACEPKT[6]	**TRACEPKT[7]**
			TRACEPKT[8]	**TRACEPKT[9]**
			TRACEPKT[10]	**TRACEPKT[11]**
			TRACEPKT[12]	**TRACEPKT[13]**
			TRACEPKT[14]	**TRACEPKT[15]**

Figure 3-14 on page 3-34 shows the logic to implement multiplexed data trace signals.

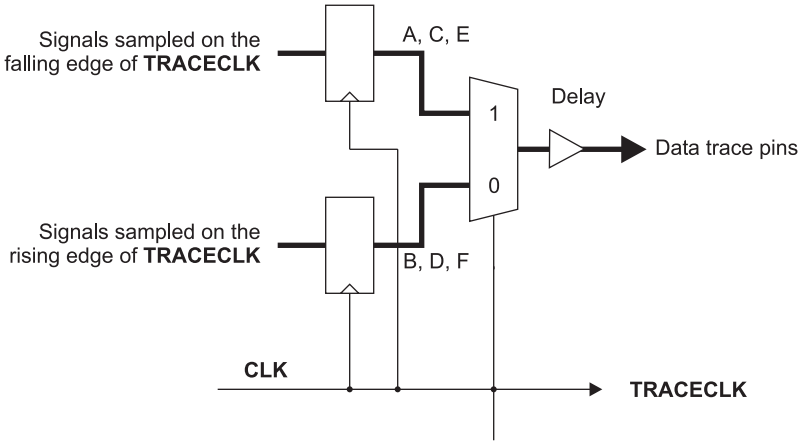


Figure 3-14 Multiplexing data trace signals

Figure 3-15 shows the timing of the multiplexed signals.

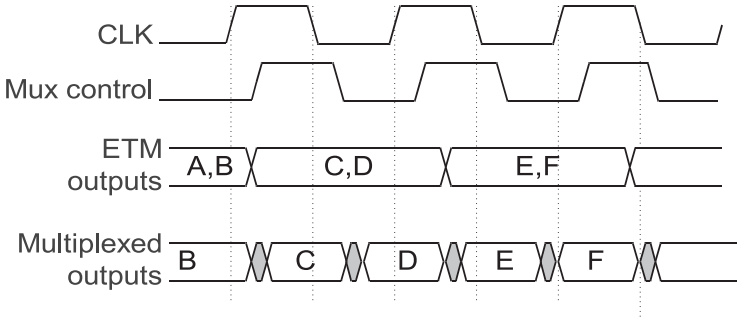


Figure 3-15 Multiplexed signal timing

Sufficient delays must be present in the switching of the trace data pins with respect to both edges of **TRACECLK**. You can achieve this by ensuring that **TRACECLK** is taken from the root of the ASIC clock tree. It is recommended that you carry out careful analysis to verify the timing on the pins of the ASIC.

3.7.3 Demultiplexed trace port signals

This scheme is recommended in systems that reuse general ASIC pins for trace, or for high-speed systems where the switching frequency of the off-chip trace signals is unacceptable. Figure 3-16 on page 3-35 shows logic to implement a demultiplexed trace port.

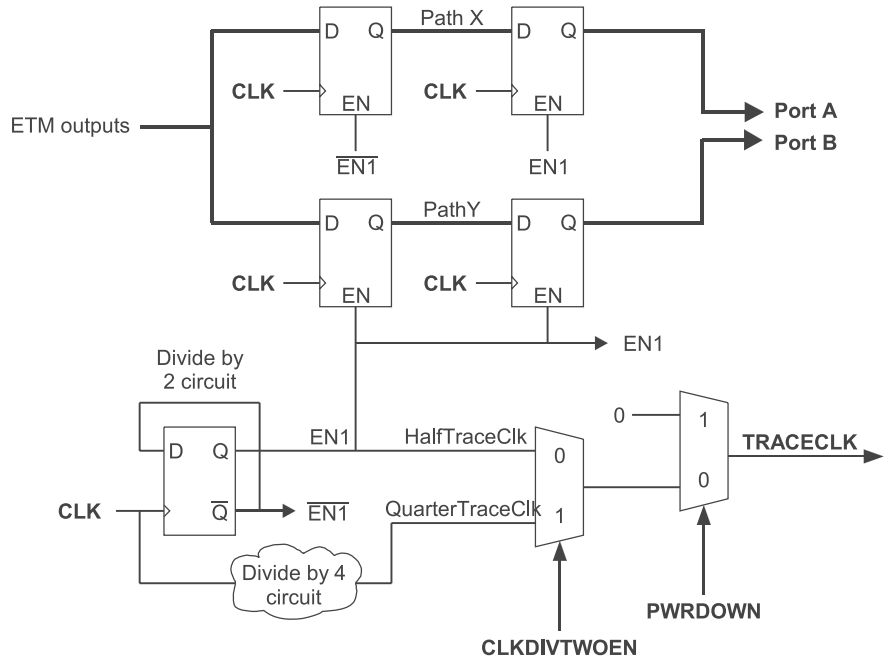


Figure 3-16 Demultiplexing trace data signals

Figure 3-17 on page 3-36 shows the timings for demultiplexed trace data signals.

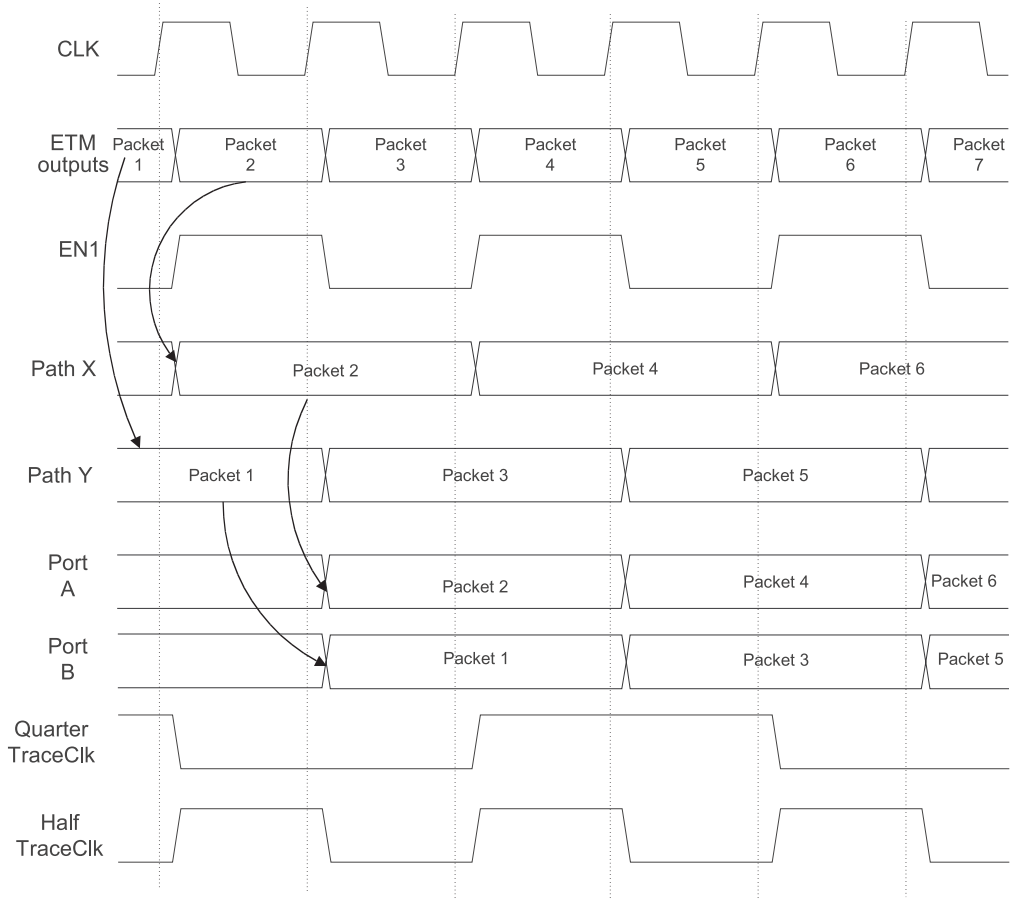


Figure 3-17 Demultiplexed signal timing

Quarter-rate clocking can be supported with demultiplexed trace data signals. However, you must take care to produce a clean trace clock. The following Verilog is an example of how a quarter-rate clock might be produced:

```
always @(posedge CLK or negedge nRESET)
  if (nRESET == 1'b0)
   begin
    State <= 2'b00;
    QuarterTraceClk <= 1'b0;
    end
  else
   case (State[1:0])
      2'b00  : begin
```

```
            State <= 2'b01;
            HalfTraceClk <= 1'b0;
            QuarterTraceClk <= 1'b0;
            end
2'b01  : begin
            State <= 2'b10;
            HalfTraceClk <= 1'b1;
            QuarterTraceClk <= 1'b1;
         end
2'b10  : begin
            State <= 2'b11;
            HalfTraceClk <= 1'b0;
            QuarterTraceClk <= 1'b1;
         end
2'b11  : begin
            State <= 2'b00;
            HalfTraceClk <= 1'b1;
            QuarterTraceClk <= 1'b0;
         end
default : begin
            State <= 2'bXX;
            HalfTraceClk <= 1'bX;
            QuarterTraceClk <= 1'bX;
         end
```

This scheme ensures that the delay from the system clock to **TRACECLK** is minimized and ensures that there are no registers clocked from the data output of other registers. This also ensures transitions of QuarterTraceClk occur on the rising edge of HalfTraceClk. This helps static timing analysis.

In demultiplexed mode, the TPA must examine the two cycles of trace data in parallel to determine whether a trigger has occurred. It must also check for the trace disabled pipeline status in both cycles of data. In each pair of trace cycles, the first cycle is output on Port B and the second on Port A.

3.7.4 Operation with asynchronous TCK

You can use the ETM9 in systems that have a fully asynchronous **TCK** and **CLK**. All synchronization issues are taken care of in the ETM9. All groups of signals are synchronous to the relevant clock:

- ARM9 interface **CLK**
- Trace port **CLK**
- JTAG port **TCK**.

Slow system clock speeds

The ETM9 contains synchronizing D-types to synchronize between the **TCK** timing domain and the **CLK** timing domain. When the system clock speed is very slow, this synchronization time causes a delay of several cycles before you can disable tracing.

Chapter 4
Memory Map Decode Interface

This chapter discusses the Memory Map Decode Interface used in the ETM9. It contains the following sections:

4.1 About the memory map decode interface

When you implement an ASIC or ASSP, there are usually a number of memory-mapped peripherals and areas of external and internal RAM, ROM, and flash, for example.

The *Memory Map Decode* (MMD) outputs allow simple, low-cost decoding of this address map using ASIC-specific logic. This logic drives the **MMDIN** inputs to the ETM, making them available to you as ETM resources, in a similar way to the address comparator and address range comparator resources.

The structure of the MMD logic is shown in Figure 4-1.

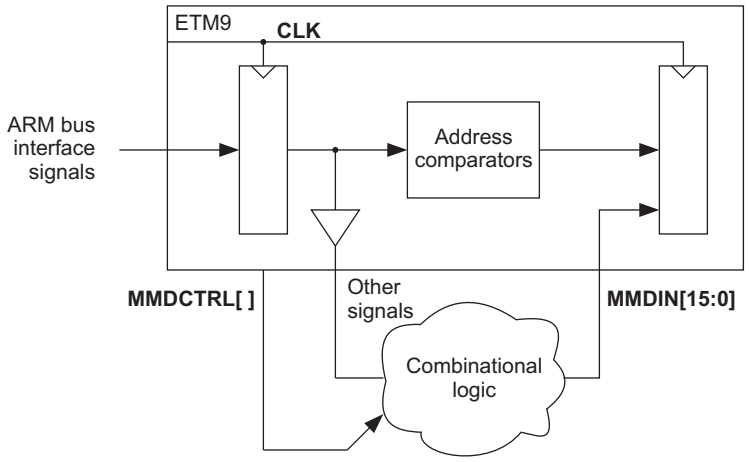


Figure 4-1 Memory map decode logic structure

In Figure 4-1, *Other signals* are:

- **MMDIA[]**
- **MMDDA[]**
- **MMDDnMREQ**
- **MMDDnRW**
- **MMDInMREQ**.

If no MMD logic is implemented, you must tie the **MMDIN** inputs to ground. The **MMDCTRL** bus comes from the memory map decode control register in the ETM, programmed by the Trace debug tools. These allow you to specify the value to be programmed into this 8-bit register.

4.1.1 Signal descriptions

The MMD signals are as follows:

MMDDnMREQ Pipelined version of **DnMREQ**.

MMDDnRW Pipelined version of **DnRW**.

MMDInMREQ Pipelined version of **InMREQ**.

MMDDA[31:0] Pipelined version of **DA[31:0]**.

MMDIA[31:1] Pipelined version of **IA[31:1]**.

MMDCTRL[7:0] Memory map decode control signals.

MMDIN[15:0] Decoded MMD resource inputs.

4.2 Memory map decode example

Figure 4-2 shows a memory map decode example based on an ARM966E-S system.

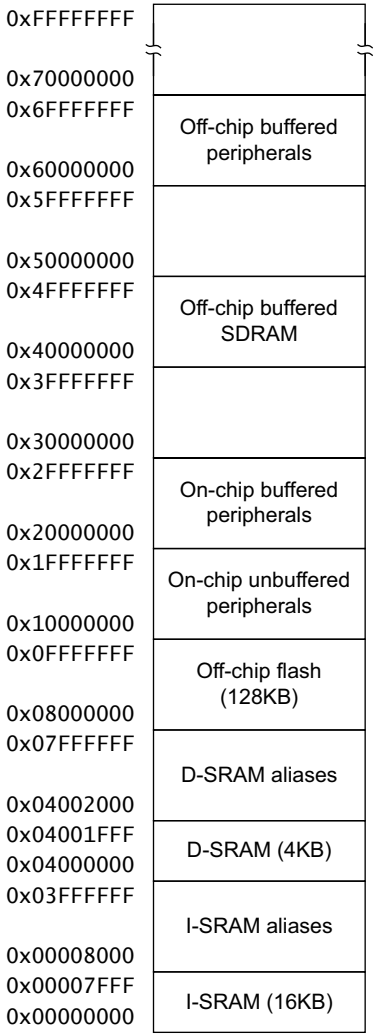


Figure 4-2 Memory map decode example based on an ARM966E-S core

The combinational logic used to decode memory map addresses in Figure 4-2 on page 4-4 is shown in pseudo-HDL format in Table 4-1.

Table 4-1 Memory map decode example pseudo-HDL

Logic expression	Comment
MMDIN[0] = (MMDIA[31:24] = 0x00) AND NOT(MMDInMREQ)	I-SRAM
MMDIN[1] = (MMDIA[31:24] = 0x04) AND NOT(MMDInMREQ)	D-SRAM
MMDIN[2] = (MMDDA[31:24] = 0x08) AND NOT(MMDDnMREQ)	Data access to flash
MMDIN[3] = (MMDIA[31:24] = 0x08) AND NOT(MMDInMREQ)	Instruction access to flash
MMDIN[4] = (MMDDA[31:20] = 0x100) AND NOT(MMDDnMREQ)	Three unbuffered peripherals
MMDIN[5] = (MMDDA[31:20] = 0x101) AND NOT(MMDDnMREQ)	
MMDIN[6] = (MMDDA[31:20] = 0x102) AND NOT(MMDDnMREQ)	
MMDIN[7] = (MMDDA[31:20] = 0x200) AND NOT(MMDDnMREQ)	Two buffered peripherals
MMDIN[8] = (MMDDA[31:20] = 0x201) AND NOT(MMDDnMREQ)	
MMDIN[9] = (MMDDA[31:28] = 0x4) AND NOT(MMDDnMREQ)	Off-chip SDRAM
MMDIN[10] = (MMDDA[31:28] = 0x600) AND NOT(MMDDnMREQ)	Off-chip buffered peripherals
MMDIN[11] = (MMDDA[31:28] = 0x601) AND NOT(MMDDnMREQ)	
MMDIN[12] = (MMDDA[31:28] = 0x602) AND NOT(MMDDnMREQ)	
MMDIN[13] = (MMDDA[31:28] = 0x603) AND NOT(MMDDnMREQ)	
MMDIN[14] = (MMDDA[31:28] = 0x604) AND NOT(MMDDnMREQ)	

Chapter 5
Test Wrapper

This chapter describes the ETM9 test wrapper. It contains the following section:

- *About the ETM9 test wrapper* on page 5-2.

5.1 About the ETM9 test wrapper

The scan chain in the ETM9 test wrapper can be used in conjunction with other scan chains that have been inserted in the design for manufacturing test coverage using *Automatic Test Pattern Generation* (ATPG) techniques.

5.1.1 Scan insertion and ATPG

This technique is covered in detail in the *ETM9 Revision2p2 Implementation Guide*. Scan insertion requires that all register elements are replaced by scannable versions that are then connected up into a number of scan chains as shown in Figure 5-1. These scan chains are used to set up data patterns on the combinatorial logic between the registers, and capture the logic outputs. The logic outputs are then scanned out while the next data pattern is scanned in.

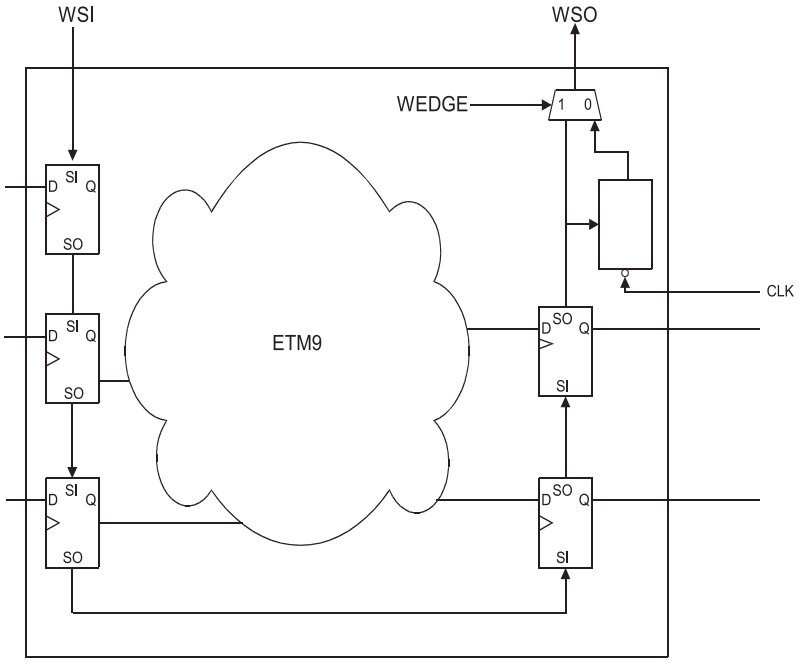


Figure 5-1 Scannable registers in ETM9

ATPG tools are used to create the necessary scan patterns to test the logic, when the scan insertion has been performed. This technique enables very high fault coverage to be achieved for the standard cell combinatorial logic, typically in the 95-99% range.

Scan insertion does have an impact on the area and performance of the synthesized design, because of the larger scan register elements and the serial routing between them. To minimize these effects the scan insertion is performed early in the synthesis cycle and the design re-optimized with the scan elements in place.

ETM9 test wrapper

A test wrapper can be used to improve test coverage where an ASIC contains a black-box macrocell, that is, where there is no internal visibility of the ETM9. For logic to be testable, the input to the logic must be controllable using a scan chain, and so must be driven by a register. The output of the logic must also be observable through a scan chain, and so must be registered.

If the ETM9 is integrated into an ASIC as a black box, the test tools do not have visibility of the internal core scan chains and cannot create vectors to cover any logic between the last register in the core and the next register in the ASIC. This is known as a *test shadow* and leads to a reduction in test coverage.

The addition of a test wrapper enables this shadow logic to be tested. The test wrapper is a scan chain around the periphery of the ETM9 that connects to each input and output. The test wrapper scan chain can be used in two modes:

- INTEST
- EXTEST.

In INTEST mode, all ETM9 inputs are driven using the test wrapper scan chain, and all ETM9 outputs are observable through the test wrapper scan chain. This enables a complete set of ATPG vectors to be produced for the macrocell in isolation.

In EXTEST mode, all ETM9 outputs are driven using the test wrapper scan chain. All ETM9 inputs are observable through the test wrapper scan chain. This enables the logic surrounding the ETM9 to be tested without the test tools requiring internal visibility of the macrocell.

Figure 5-2 on page 5-4 shows the DFT features available through the test wrapper.

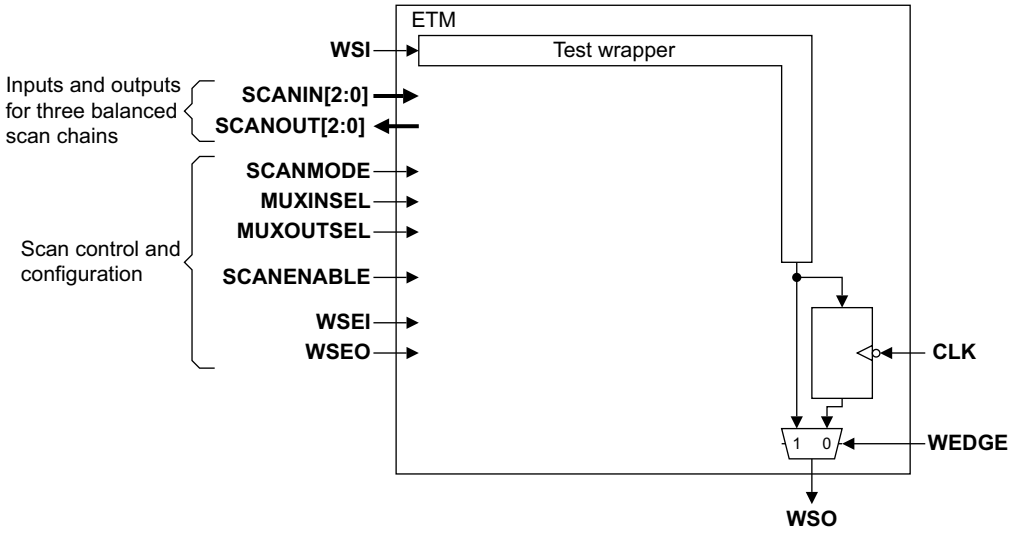


Figure 5-2 Test wrapper

Table 5-1 shows how to set the scan configuration pins.

Table 5-1 Scan configuration pins

Signal	Normal	INTEST	EXTEST	Description
SCANENABLE	0	T	0	**SCANENABLE** enables scan logic inside the ETM only.
WSEI	0	T	T	**WSEI** enables input signals and, in conjunction with **WSEO**, permits launch capture speed testing of synthesizeable devices.
WSEO	0	T	1	**WSEO** enables output signals and, in conjunction with **WSEI**, permits launch capture speed testing of synthesizeable devices.
MUXINSEL	0	1	0	**MUXINSEL** is similar to the **INTEST** pin on the core and enables INTEST mode.
MUXOUTSEL	0	0	1	**MUXOUTSEL** is similar to the **EXTEST** pin on the core and enables EXTEST mode.
SCANMODE	0	1	1	**SCANMODE** = 1 selects the clock signal **CLK** for all registers.
SCANIN[2:0]	0	T	0	**SCANIN[2:0}** are the inputs to the three balanced scan chains in the device. There are approximately 800 cells in each chain.
WSI	0	T	T	**WSI** is the serial input data for the test wrapper scan chain.

Table 5-1 Scan configuration pins (continued)

Signal	Normal	INTEST	EXTEST	Description
WEDGE	0	1	1	**WEDGE** changes on which edge of the clock **WSO** changes. The default value is: **WEDGE** = 1, **WSO** changes on a rising clock edge.
nTRST[a]	1	T	T	**nTRST** is the TAP controller reset.
nRESET[b]	1	T	T	**nRESET** is the ETM reset signal.

a. See *TAP reset* on page 3-14 and Chapter 6 *ETM Integration Testing*.
b. See *ETM reset* on page 3-13 and Chapter 6 *ETM Integration Testing*.

Note

In Table 5-1 on page 5-4:

- A T indicates that these inputs are controlled by the tester. The tester determines the state of these signals.
- Table 5-1 on page 5-4 does not show the test interface outputs, **SCANOUT[2:0]**.

Figure 5-3 shows how to use test wrappers to test SoC logic with both the ETM and a processor.

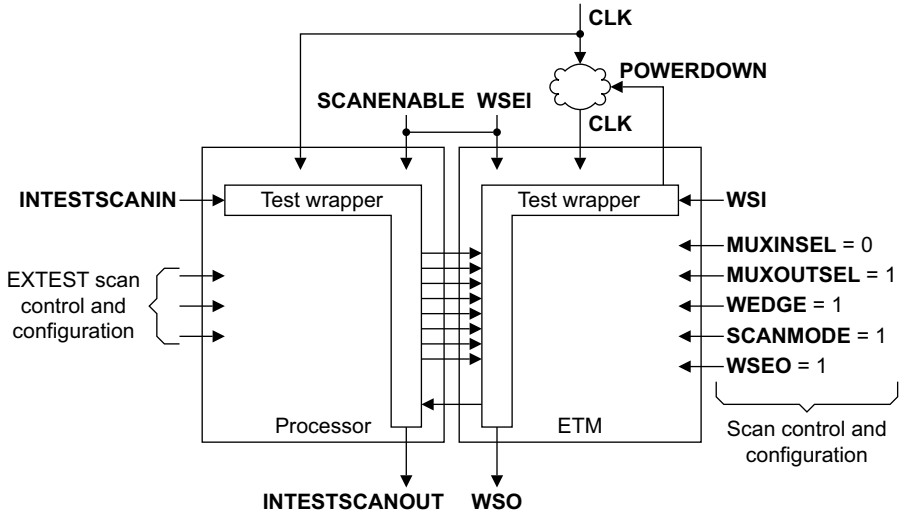


Figure 5-3 Testing SoC logic with test wrappers

Chapter 6
ETM Integration Testing

ETMIK is an environment to demonstrate the correct integration of an ETM with an ARM core. This chapter describes the kit.

Note

This chapter describes the operation of the ETM Integration Kit for the ETM7 and ETM9 but this chapter should only be used for ETM9 integration testing. Refer to the appropriate ETM7 Technical Reference Manual if you wish to perform ETM7 integration testing. The procedure for using the ETM7 Integration Kit is not maintained in the ETM9 Technical Reference Manual.

This chapter contains:

6.1 About the ETM Integration Kit

The ETMIK is an environment to demonstrate the correct integration of an ETM with an ARM core. The kit includes examples for various ARM cores and also shows how to connect the ETM to an ARM *Embedded Trace Buffer* (ETB).

Note

This chapter refers to ETM7 and ETM9 components of the ETMIK.

ETMIK comprises:

- the HDL files for each configuration
- an example test bench based on an AMBA AHB system
- test programs to verify the correct integration of the system.

The cores, ETM, ETB, and AHB wrappers are not included with the ETMIK. These are supplied separately.

6.1.1 Design flow

An outline of the design flow when using the ETMIK is shown in Figure 6-1. The stages shown are described in Chapter 3 *Integrating the ETM9*.

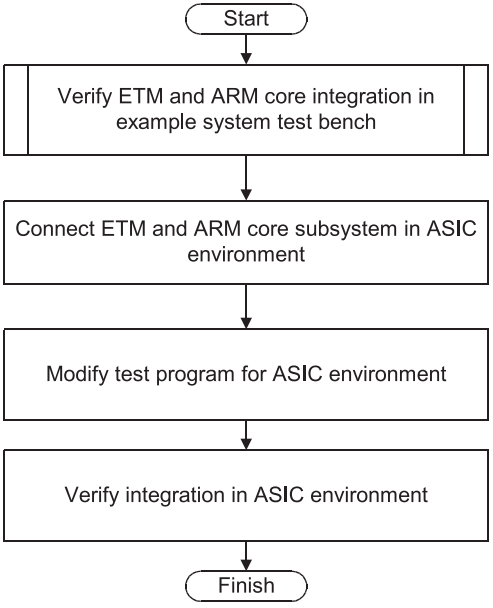


Figure 6-1 ETMIK design flow

6.1.2 Supported configurations

The ETMIK is supplied with HDL files for various ARM cores. Each of these has the option to include an on-chip ETB. The supported configurations are:

- ARM7TDMI rev 3 (AHB wrapped) with ETM7 rev 1A
- ARM7TDMI rev 4 (AHB wrapped) with ETM7 rev 1A
- ARM7TDMI-S rev 4 (AHB wrapped) with ETM7 rev 1A
- ARM720T rev 3 (AHB wrapped) with ETM7 rev 1A
- ARM720T rev 4(AHB wrapped) with ETM7 rev 1A
- ARM920T rev 1 (AHB wrapped) with ETM9 r2p2
- ARM922T rev 0 (AHB wrapped) with ETM9 r2p2
- ARM926EJ-S rev 0 with ETM9 r2p2
- ARM946E-S rev 1 with ETM9 r2p2
- ARM966E-S rev 1 with ETM9 r2p2
- ARM966E-S rev 2 with ETM9 r2p2.

The test bench is suitable for modification to support cores not directly supported by the ETMIK.

Configurations indicated as being AHB wrapped require the ARM AHB wrappers to convert the memory interface to that of an AHB-compliant device because the test bench is based on an AHB system.

6.1.3 Directory structure

Figure 6-2 on page 6-5 shows the ETMIK directory structure. It is based on the ARM SoC directory structure:

Test program

`EtmIntKit/design/logical/tbench/tests/results` contains the results files for the test programs. `EtmIntKit/design/logical/tbench/tests/src` contains the source files for the test programs.

HDL source

`EtmIntKit/design/logical/arm_and_etm/verilog` and its subdirectories contain the HDL files for demonstrating the connection of an ARM core and ETM. Those prefixed with ARM contain files specific to the implementation of the named processor core.

`EtmIntKit/design/logical/ElementsAHB` and its subdirectories contain the HDL files for the AHB components of the test bench.

`EtmIntKit/design/logical/ahb_wrapper/verilog/rtl_source` is a placeholder to insert the HDL files for an AHB wrapper. The AHB wrappers are not supplied with the ETMIK.

`EtmIntKit/design/logical/etb` is placeholder to insert the HDL files for the ETB. The ETB is not supplied with the ETMIK and must be installed here if used.

Simulation

`EtmIntKit/design/logical/tbench` contains scripts to help set up the environment for running the simulation. The subdirectories contain extra scripts to help run a simulation with a specific simulator.

`EtmIntKit/design/logical/tbench/verilog` and its subdirectories contain the behavioral HDL files for use within the simulation.

Verification scripts

`EtmIntKit/bin` contains Perl scripts to verify the ETM traced program execution correctly.

Documents

`EtmIntKit/documentation` contains files related to the ETMIK.

Libraries

`EtmIntKit/libraries/ARM/DSM` is a placeholder to install the *Design Sign-Off Models* (DSM) required for use with the ETMIK. The DSMs are not supplied with the ETMIK.

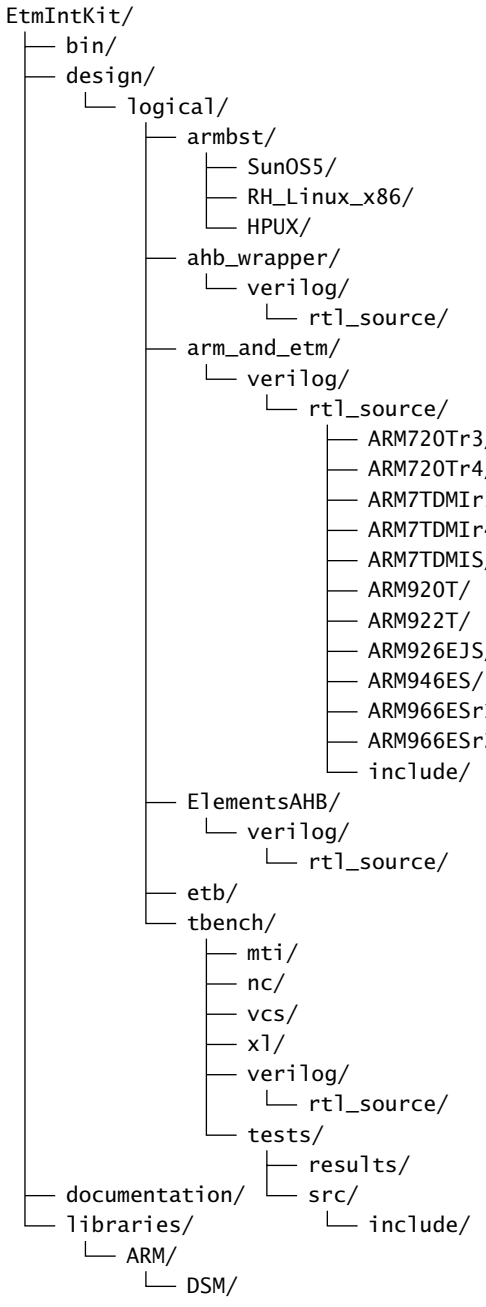


Figure 6-2 ETMIK directory structure

6.1.4 Additional components required

Table 6-1 shows components that are not supplied. In Table 6-1 ***** is a simulator/platform specific number. These additional components must be unpacked as required and put into the appropriate locations as detailed in *Source compilation* on page 6-25. 1

Table 6-1 Additional components

Description	Part number
ARM7TDMI (Rev 3) *Design Simulation Model* (DSM)	AT010-MS-*****
ARM7TDMI (Rev 4) *Design Simulation Model* (DSM)	AT010-MS-*****
ARM7TDMI-S DSM	AT080-MS-*****
ARM720T (Rev 3) DSM	AT070-MS-*****
ARM720T (Rev 4) DSM	AT070-MS-*****
ARM920T (Rev 1) DSM	AT090-MS-*****
ARM922T (Rev 0) DSM	AT070-MS-*****
ARM926EJ-S (Rev 0) DSM	AT230-MS-*****
ARM946E-S (Rev 1) DSM	AT210-MS-*****
ARM966E-S (Rev 1) DSM	AT200-MS-*****
ARM966E-S (Rev 2) DSM	AT200-MS-*****
ARM ETM7 (Rev 1A) DSM	TM03*-MS-*****
ARM ETM9 (r2p2) DSM	TM02*-MS-*****
ARM AHB Wrappers (Rev 1)	BU030-BU-00001
ARM Embedded Trace Buffer	TM060-MN-22100

6.1.5 Design structure

The structure of the ETMIK is shown in Figure 6-3 on page 6-7.

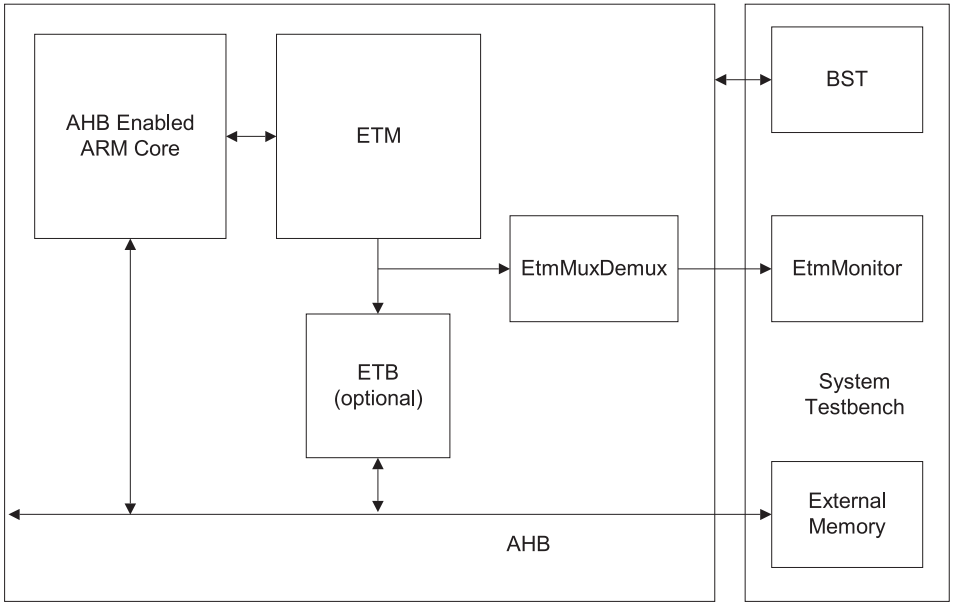


Figure 6-3 Design structure

The system is constructed from an AHB-enabled core, connected to the ETM. The trace outputs from the ETM are connected through a module, `EtmMuxDemux`, that converts the trace depending on the current trace port mode. The trace outputs are also connected to an optional on-chip trace buffer (ETB). This buffer is used to store trace output for transfer over a JTAG interface, and is also accessible across the AHB.

The test bench comprises:

- some external memory for program and data storage
- the ARM BST to control the JTAG interface. The BST is equivalent to a Multi-ICE device in a hardware system
- the ETM Monitor that takes the trace output and stores it to a file. The ETM Monitor is equivalent to a Trace Port Analyzer.

It is intended that this system is an example. It must be replaced by your ASIC design.

6.2 Test system

This section describes the structure of the test programs, test system and test bench supplied with the ETMIK. Read *ETM integration test program* on page 6-14 to determine whether the test program needs adapting for your system.The HDL is split into the hierarchical structure shown in Figure 6-4.

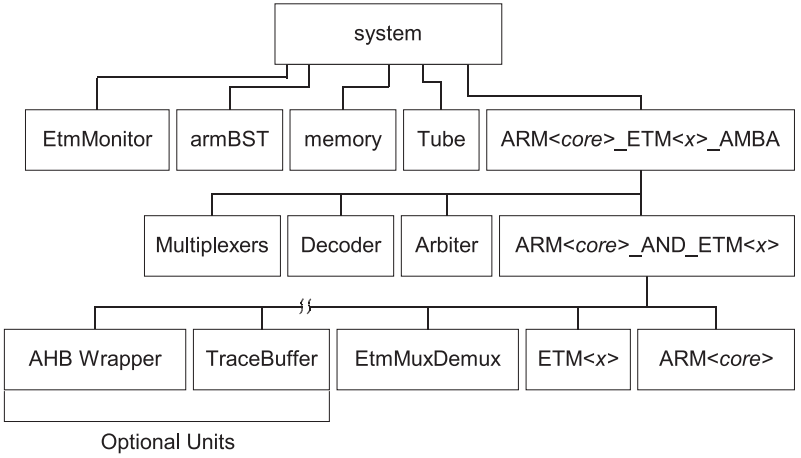


Figure 6-4 HDL hierarchy

6.2.1 The system test bench

The test bench is located in `EtmIntKit/design/logical/tbench/verilog/rtl_source/system.v`. It instantiates some external memory, the BST, EtmMonitor and a *Tube* used for displaying messages to the screen.

The external memory

This is a dynamically allocating memory model, with an AHB-compliant interface. On startup, it loads the file `rom.hex` which contains the program to run in the system. It also implements a bad memory location which returns an ERROR response on the AHB. This enables memory aborts to be tested, and is at address `0x02000000`.

When testing aborts in your ASIC, you must direct the test to a region of memory where aborts can be generated.

The *Boundary Scan Trickbox* (BST)

The BST is used to control the JTAG port, similar to the way a Multi-ICE device does in a hardware system. In this test, it is used to program the ETM and ETB and to control the ARM core during debug. It reads a file called `JTAGbsi` that contains instructions to control the BST.

The commands that you scan into the JTAG port must be included in the source code of the test. The `TOBST` assembler macro indicates the start of the BST instructions in the test.

A script called `bintobst` is provided. This script first extracts the BST instructions from the assembled binary image of the test. It then uses a program called `parse_bsi.pl` to process the high level commands into a form that the BST can accept.

If the `JTAGbsi` file read by the BST is incorrect, it can cause the following types of problems:

- The ARM does not enter debug state
- The ETM is not programmed correctly
- The ARM does not exit debug state, or restarts at the wrong address.

Many of these problems can be caused by the BST incorrectly identifying the type of ARM processor (ARM7 family or ARM9 family) that it is talking to. The test program therefore requires that the *`PROC`* variable is defined. This variable controls the size of JTAG TAP controller.

Another problem that can arise is that the `parse_bsi.pl` script can have difficulty assembling the ARM instructions to be executed in Debug state. The `ARMINST` command specifies an instruction to execute. The `armasm` program (not provided in the ETMIK) is called to assemble the ARM instruction. The resulting instruction bit pattern is then turned into a BST scan command. This instruction is scanned into the ARM and executed at debug speed. For details of exactly how this works, refer to the appropriate ARM core Technical Reference Manual.

Other problems can arise if `armasm` is not available, or if the version available is too old. You must use the `armasm` program that is provided with the unix version of the *ARM Developer Suite* (ADS) or SDT 2.50 (SDT 2.11a is not supported).

EtmMonitor

The EtmMonitor module simulates the operation of a Trace Port Analyzer. It captures the data coming from the trace port of the ETM and stores it to a log file, `log.etmraw`. Analysis of this file is performed in *Test verification* on page 6-33

Tube

This is an AHB-compliant module to print messages to the screen during simulation. ASCII bytes are written to it, and when a string termination character is written, the message is displayed. If a CTRL+D character is written, the simulation is terminated after a few cycles, to allow the ETM FIFO to drain. The Tube is located at address `0xC0000000` in the memory map.

6.2.2 HDL hierarchy

The lowest level is where the ARM core and ETM are instantiated. This module (in `ARM<core>_AND_ETM<x>.v`) is an example of how to correctly connect the ETM to the respective core. It follows the instantiations described in Chapter 3 *Integrating the ETM9*. It also demonstrates how to correctly interface the `EtmMuxDemux` module and the ETB. If the ETB is not required, the ETB define must not be defined. Refer to *Verilog source compilation* on page 6-26.

The `ARM<core>_AND_ETM<x>` module does not require changing, so all unused inputs and most outputs are brought out in the port declaration.

ETM outputs and inputs

The following ETM-related signals are brought out. The following ETM-related signals are outputs:

- **CLKDIVTWOEN**
- **ETMEN**
- **EXTOUT**
- **MMDA**
- **MMDCTRL**
- **MMDMAS**
- **MMDnMREQ**
- **MMDnOPC**
- **MMDnRW**
- **PIPESTATA**
- **PIPESTATB**
- **PORTMODE**
- **PORTSIZE**
- **PWRDOWN**
- **TRACECLK**
- **TRACEPKTA**
- **TRACEPKTB**

- **TRACESYNCA**
- **TRACESYNCB**.

These ETM-related signals are inputs:

- **ARMTDO**
- **EXTIN**
- **MMDIN**
- **SYSOPT**.

The memory map decode and external logic signals are provided so that any external logic can be connected to the ETM, and the trace port is connected to higher hierarchical levels. The inputs are either tied LOW or configured as shown in the `ARM<core>_ETM<x>_AMBA.v` file, and must be connected as is appropriate to your system.

Note

The ETM7 **CLK** and **CLKEN** signals are not connected as described in the *ETM7 Technical Reference Manual* for the ARM7TDMI. This is because the ETB AHB interface and ARM7TDMI AHB wrapper can cause a deadlock situation. Therefore the signal **MCLKEN** is brought out from the AHB wrapper, and is connected to the ETM **CLKEN** input. The ETM **CLK** input and ETB **CLK** input are now driven by **HCLK** (rather than !**MCLK**). This enables the ETM and ETB to continue operation even when the ARM7TDMI is stalled. This strategy is implemented whether or not an ETB is present. This modification to the AHB wrapper and other changes made are detailed in *AHB wrappers* on page 6-13.

ARM core

The bus interface to the core is treated as an AHB device. If the core is not already AHB enabled, an AHB wrapper is integrated. All signals not connected to the ETM or JTAG port are brought out to the higher level.

Embedded Trace Buffer

All ETB signals not connected to the ETM trace port or JTAG port are brought out to the higher level. This includes the entire AHB and BIST interfaces. If the `ETB` define is not present, the ETB port signals are not included.

Note

The ETB AHB interface must not be used in a system based on the ARM720T (Rev 3) core because the interaction of the ARM720T (Rev 3) AHB wrapper and the ETB can cause a situation where deadlock occurs.

EtmMuxDemux

The `EtmMuxDemux` module takes the standard trace port interface from the ETM and, using the **PORTMODE** and **CLKDIVTWOEN** outputs, converts the trace to the correct port mode. This is transmitted over the trace port A and B connections, as described in Section *ETM outputs and inputs* on page 6-10.

JTAG interface

The JTAG interface is connected as described in *TAP interface wiring* on page 3-16. A common **TCK** controls all the JTAG ports on the core, ETM and ETB. If an ETB is connected, the JTAG port is connected as in Figure 6-5.

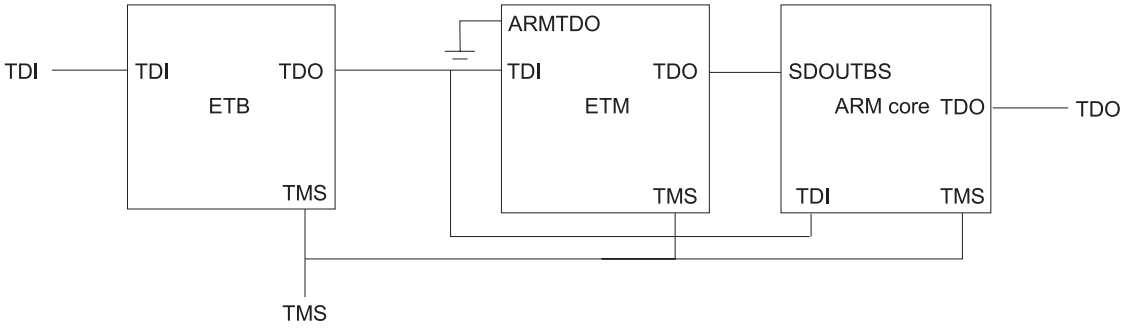


Figure 6-5 JTAG wiring

Note

The test program and test verification scripts assume this wiring structure. If you modify this by adding any custom external scan chains, the test program also needs modification. The test verification configuration also needs changing. This is described in *Trace Comparison Scripts* on page 6-33.

On the ARM926EJ-S, ARM946E-S, and ARM966E-S cores, the **TCKEN** signal must be tested. On all other cores, it must be held HIGH. To test the ETM **TCKEN** pin is correctly connected, the system-wide **TCKEN** signal is generated by a *Linear Shift Feedback Register* (LFSR). This is unlikely to be implemented in your ASIC, so another method for testing this pin must be used when testing your ASIC design.

ARMcore_ETMx_AMBA

This module instantiates the `ARMcore_AND_ETMx` module, along with the other AHB system components. These include the Arbiter, which only deals with one AHB master except in an ARM926EJ-S system. Also included is the decoder which is customized

for this system and the AHB multiplexors. This module also controls the default signal assignments for the ARM core, ETM, and ETB. It is intended this module is an example and must be replaced by your ASIC design.

AHB wrappers

The ARM7TDMI, ARM720T (Rev 3), ARM920T, and ARM922T cores all require AHB wrappers to function within the test bench. The ARM AHB wrappers can be used with the ARM720T (Rev 3) and ARM920T cores. The ARM922T core uses the same wrapper as the ARM920T core.

The ARM7TDMI wrapper is a modified version of the AHB wrapper release. This is to enable the internal **MCLKEN** signal to be used when connecting the ETM. The following signals are also brought out from the wrapper, so default assignments or other devices can be connected to them.

- **xBIGEND**
- **xBREAKPT**
- **xCPA**
- **xCPB**
- **xDBGEN**
- **xEXTERN0**
- **xEXTERN1**
- **xISYNC**
- **xSDOUTBS**
- **xDBGRQ**.

Therefore the supplied files in `EtmIntKit/design/logical/arm_and_etm/verilog/rtl_source/ARM7TDMIr3` or `EtmIntKit/design/logical/arm_and_etm/verilog/rtl_source/ARM7TDMIr4 must be used in place of the standard ARM7TDMI wrapper.` These files must be left in their respective directories.

6.3 ETM integration test program

The test program supplied in `EtmIntKit/design/logical/tbench/tests/src/test.s` tests all the major interconnects between the ETM and the ARM core. It interacts with the test bench to control certain pins on the ETM. You do not have to modify the program unless you are adding any of the following functionality to the ETM:

- Memory Map Decode logic
- External Inputs (EXTIN inputs)
- External outputs (EXTOUT outputs)
- External Scan chain connected to ARMTDO.

If your ASIC design requires any of these, then you must add custom testing to the test program. Regions are indicated within the test program for implementing your own tests.

6.3.1 Test program breakdown

At the start of the test, the core comes out of reset and enters an infinite loop. After a short delay, the core is forced into debug state by asserting the debug request bit in the debug control register using the JTAG interface. During this the ETM is programmed to trace continuously, including PROCID. If present, the ETB is also programmed to capture the trace output. Once the ETM is programmed, the core is enabled to start executing from the end of the vector table.

Note

The test program uses an absolute address to restart the processor. Therefore when you use HIVECS you must edit the test program, near line 817, to point to address `0xFFFF0020`.

The first task of the program is to cause an ETM trigger to be output which causes the external debug request pin on the core to be asserted. The core enters debug and the EmbeddedICE watchpoints are programmed by the BST. When the core returns from debug state, the main part of the test is executed. This involves testing coprocessor instructions and changing the PROCID signals, testing the data interface with multiple loads and stores, switching the core between ARM and Thumb state, testing data and Prefetch Aborts, and on Jazelle enabled cores switching between ARM and Java state.

After this section is complete, the watchpoint signals are tested by activating both EmbeddedICE watchpoint units. While the core is in debug after the second watchpoint, the ETM is programmed to stop tracing and de-assert the **ETMEN** signal, which will stop any more trace output. To test the **ETMEN** is correctly connected, another trigger packet is activated, causing the core to re-enter debug state. If the ETB is present, this section of the test reads out the contents of the traced data over the JTAG port. This is a

time-consuming stage of the test as the data has to be shifted out serially. The data is stored in the `log.dsm_bst` file which is the normal log file produced by the BST. Details of how to extract the trace data from this file are given in *Test verification* on page 6-33. The core then returns from debug state.

Note

The EmbeddedICE watchpoint units are not tested on an ARM9TDMI based core (ARM920T and ARM922T). See the ARM9TDMI errata document for an explanation.

If an ETB is present, the core then reads the data write pointer register, to check if any trace data has been stored. If the pointer is zero, this indicates no tracing has occurred, or an AHB bus failure, and the test fails.

Note

The ETB AHB section of the test does not run on an ARM720T (Rev 3)-based system because the interaction of the ARM720T (Rev 3) AHB wrapper and the ETB can cause a situation where deadlock occurs.

If the test runs to completion, the message `** TEST COMPLETED - Now proceed to Test Verification **` is displayed on the Tube.

The presence of this message does not indicate correct connection of the core, ETM and ETB. The trace output needs to be verified, as described in *Test verification* on page 6-33.

Note

- If **FIFOFULL** stalling is supported by your processor then the test program has to know the threshold at which the ETM stalls the core to enable the internal FIFO to empty. A preprocessor define is included in the Make file, which has to be set to indicate the configuration of the ETM. The threshold is then set to the maximum size of the FIFO to avoid any loss of trace. If **FIFOFULL** stalling is not supported, the test program inserts multiple NOP (No OPeration) instructions to prevent FIFO overflow within the ETM. This is required to ensure the entire program is traced by the ETM and to verify full functionality.
- The number of NOPs used is suited to using a medium configuration ETM, and trace data is likely to be lost if a small configuration is used
- Not all cores support the full functionality of the test program, so sections are included and excluded based on the core in use.

6.3.2 Adapting the test program

The test program does not test all of the pins on the ETM. Some are implementation-dependent and some are not tested because of complexity. Table 6-2 describes how each pin of the ETM7 is tested, and if not, how it can be tested.

Table 6-2 ETM7 signals

Signals	How tested
CLK, **CLKEN**, **nRESET**	If any of these are not correctly connected, the test can fail at either run time or during test verification.
TCK, **TCKEN**, **TDI**, **TDO**, **TMS**, **nTRST**	If any of these are not correctly connected, the test can fail at either runtime or during test verification.
ARMTDO	This pin is not directly tested. It must be tested if an external scan chain is connected to the ETM. This can be performed by reading some data from the external scan chain using the JTAG interface.
A[31:0], **RDATA[31:0]**, **WDATA[31:0]**, **nMREQ**, **SEQ** **nRW**, **MAS[1:0]**, **nOPC**, **nEXEC**	These are tested by performing multiple memory accesses. If incorrectly connected, the test verification indicates incorrect data.
ABORT	This is tested by causing data and prefetch aborts. If this is not correctly connected, this is indicated during test verification by the absence of abort tracing.
TBIT	This bit is tested by switching between ARM and Thumb state. Test verification indicates incorrect transitions if this is incorrectly connected.
INSTRVALID	This pin is not directly tested. It is used to increase the accuracy of timing at the point that an interrupt is taken, and cannot be easily tested.
nCPI, **CPA**, **CPB**	These coprocessor signals are tested by transferring data to and from a coprocessor. Test verification indicates if these are incorrectly connected by not tracing coprocessor instructions properly. These are not tested on the ARM7TDMI and ARM7TDMI-S cores because this test bench does not include a coprocessor.
DBGRQ, **DBGACK**	These debug signals are tested by asserting **DBGRQ** to send the core into debug, and checking **DBGACK** is asserted in response. If any of these are not correctly connected, the test fails at either run time or during test verification.

Table 6-2 ETM7 signals (continued)

Signals	How tested
RANGEOUT0, **RANGEOUT1**	These are tested by activating the EmbeddedICE watchpoint units within the ARM core. If any of these are not correctly connected, the test fails at runtime. These are not tested on the ARM920T or ARM922T cores. See the ARM9TDMI errata document for an explanation.
BIGEND	If this signal is connected incorrectly, this is indicated during test verification.
PROCID[31:0], **PROCIDWR**	These are tested by toggling all the bits in the Process ID register. If any of these are not correctly connected, this is indicated during test verification. These are not tested on the ARM7TDMI and ARM7TDMI-S cores because they do not exist.
PIPESTAT[2:0], **TRACEPKT[15:0]**, **TRACESYNC**	These trace port signals are tested by analyzing the trace output. If any of these are not correctly connected, this are indicated during test verification.
PWRDOWN	This is used to gate the **CLK** input to the ETM. If this is not correctly connected, the test fails at either run time or during test verification.
ETMEN	This is tested by the test bench and the test program. If this is not correctly connected, the test fails at either runtime or during test verification.
PORTMODE[1:0], **CLKDIVTWOEN**	The `EtmMuxDemux` block incorrectly formats the data if these signals are incorrectly connected. If any of these are not correctly connected, this is indicated during test verification.
PORTSIZE[2:0]	These are only tested when using the ETB. If incorrectly connected, this is indicated during test verification. If not using the ETB, the use of **PORTSIZE** is implementation defined and is not tested by the test program.

Table 6-2 ETM7 signals (continued)

Signals	How tested
FIFOFULL	This is not tested because a complementary signal is not available on the ARM7TDMI, ARM7DMI-S, or ARM720T cores.
MMDA[31:0], **MMDn, MREQ**, **MMDnRW**, **MMDnOPC**, **MMDMAS[1:0]**, **MMDCTRL[7:0]**, **MMDIN[15:0**], **EXTOUT[3:0]**, **EXTIN[3:0]**	These are external signals whose use is implementation-defined. They are not tested by the test program, but must be tested if you intend to use these connections. Appropriate locations for these modifications are indicated in the test program.
SYSOPT[7:0]	The test program reads the ETM System Configuration register through the JTAG port to verify it is connected as expected. If this is not correctly connected, the test fails at run time. The file `log.dsm_bst` indicates the stage at which the test failed.

Table 6-3 lists how each pin of the ETM9 is tested, and if not, how it may be tested.

Table 6-3 ETM9 signals

Signals	How tested
CLK, **CLKEN**, **nRESET**	If any of these are not correctly connected, the test fails at either runtime or during test verification.
TCK, **TCKEN**, **TDI**, **TDO**, **TMS**, **nTRST**	If any of these are not correctly connected, the test fails at either runtime or during test verification. In ARM926EJ-S, ARM946E-S and ARM966E-S-based systems, **TCKEN** is generated by a *Linear Feedback Shift Register* (LFSR) to ensure the connection is correct.
ARMTDO	This pin is not directly tested. It must be tested if an external scan chain is connected to the ETM. This may be performed by reading some data from the external scan chain using the JTAG interface.
DA[31:0], **DD[31:0]**, **DDIN[31:0]**, **DnMREQ**, **DSEQ**, **DnRW**, **DMAS[1:0]**, **IA[31:0]**, **ID31To25[31:25]**, **ID15To11[15:11]**, **InMREQ**, **ISEQ**, **INSTREXEC**	These are tested by performing multiple memory accesses. If incorrectly connected, the test verification indicates incorrect data.
DABORT	This pin is not directly tested. It is used by the ETM to control data comparator behavior, and cannot easily be tested.

Table 6-3 ETM9 signals (continued)

Signals	How tested
ITBIT	This bit is tested by switching between ARM and Thumb state. Test verification indicates incorrect transitions if this is incorrectly connected.
IJBIT, **ZIFIRST**, **ZILAST**	These pins are tested by switching between ARM and Java state. Test verification indicates incorrect transitions if this is incorrectly connected. This is only tested on an ARM926EJ-S core because it supports Java. They are not tested on ARM920T, ARM922T, ARM946E-S, or ARM966E-S cores and must be tied LOW.
INSTRVALID	This pin is not directly tested. It is used to increase the accuracy of timing at the point an interrupt is taken, and cannot be easily tested. The ARM920T or ARM922T cores do not support this signal.
CHSD[1:0], **CHSE[1:0]**	These coprocessor signals are tested by transferring data to and from a coprocessor. Test verification indicates if these are incorrectly connected by not tracing coprocessor instructions properly.
LATECANCEL	This coprocessor signal is tested by causing a Data Abort in the instruction preceding a coprocessor instruction. If it is connected incorrectly, this is indicated during test verification.
PASS	If this is not correctly connected, the test fails during test verification.
DBGRQ, **DBGACK**	These debug signals are tested by asserting **DBGRQ** to send the core into debug, and checking **DBGACK** is asserted in response. If any of these are not correctly connected, the test may fail at either run time or during test verification.
RANGEOUT[1:0]	These are tested by activating the EmbeddedICE watchpoint units within the ARM core. If any of these are not correctly connected, the test fails at runtime.
BIGEND, **HIVECS**	If these pins are connected incorrectly, this is indicated during test verification.
PROCID[31:0], **PROCIDWR**	These are tested by toggling all the bits in the Process ID register. If any of these are not correctly connected, this is indicated during test verification.

Table 6-3 ETM9 signals (continued)

Signals	How tested
PIPESTAT[2:0], **TRACEPKT[15:0]**, **TRACESYNC**	These trace port signals are tested by analyzing the trace output. If any of these are not correctly connected, this is indicated during test verification.
PWRDOWN	This is used to gate the **CLK** input to the ETM. If this is not correctly connected, the test fails at either runtime or during test verification.
ETMEN	This is tested by the test bench and the test program. If this is not correctly connected, the test fails at either runtime or during test verification.
PORTMODE[1:0], **CLKDIVTWOEN**	The `ETMMuxDemux` module incorrectly formats the data if these signals are incorrectly connected. If any of these are not correctly connected, this is indicated during test verification.
PORTSIZE[2:0]	These are only tested when using the ETB. If incorrectly connected, this is indicated during test verification. If not using the ETB, the use of **PORTSIZE** is implementation defined. This pin is not tested by the test program.
FIFOFULL	This is tested by enabling FIFOFULL stalling on the ETM9. This signal is not tested on the ARM920T or ARM922T cores, because they do not support FIFOFULL stalling. If it is incorrectly connected, ETM FIFO overflow occurs. See *Running the Trace Comparison Script* on page 6-40 for details on how to check if FIFO overflow has occurred.

Table 6-3 ETM9 signals (continued)

Signals	How tested
MMDA[31:0], **MMDn, MREQ**, **MMDnRW**, **MMDnOPC**, **MMDMAS[1:0]**, **MMDCTRL[7:0]**, **MMDIN[15:0**], **EXTOUT[3:0]**, **EXTIN[3:0]**	These are external signals whose use is implementation-defined. They are not tested by the test program, but must be tested if you intend to use these connections. Appropriate locations for these modifications are indicated in the test program.
SYSOPT[7:0]	The test program reads the ETM System Configuration register through the JTAG port to verify it is connected as expected. If this is not correctly connected, the test fails at run time. The file `log.dsm_bst` indicates the stage at which the test failed.
WSI, **WSO**, **MUXINSEL**, **MUXOUTSEL**, **WSEI**, **WSEO**, **WEDGE**, **SCANMODE**	These are Design For Test signals whose use is implementation defined. They are not directly tested by the test programs, but must be tested if you intend to use these connections.

6.4 Trace buffer integration test program

The test program supplied in `EtmIntKit/design/logical/tbench/tests/src/tbtest.s` tests the AHB interface of the ETB. Table 6-4 lists the signals tested by the test program.

Table 6-4 ETB signals

Signal name	Test description
HCLK, **HWRITE**, **HTRANS[1:0]**, **HSIZE[2:0]**, **HADDR[31:0]**, **HWDATA[31:0]**, **HREADY**, **HSELREG**, **HSELMEM**, **HRESETn**, **HREADYMEM**, **HRESPMEM[1:0]**, **HRDATAMEM[31:0]**	These are tested by performing multiple accesses to the ETB over the AHB bus during the ETB integration test. If any of these are not correctly connected, the test fails.
TCK, **TCKEN**, **nTRST**, **TDI**, **TMS**, **TDO**	These are tested by programming the ETB in the ETM integration test. If any of these are not correctly connected, the test fails at either run time or during test verification.
nRESET, **CLK**	If any of these are not correctly connected, the test fails.
PORTSIZE, **TRACEOUTPUT[23:0]**, **PROTOCOL[1:0]**, **TRACEVALID**, **TRIGGER**	These are tested during the ETM integration test to ensure the ETB is correctly connected to the ETM. If any of these are not correctly connected, the test fails at either run time or during test verification.
SBYPASS, **SWEN**	These are statically configured for each core configuration for use during the ETM integration test. If any of these are not correctly connected, the test fails at either run time or during test verification.
BISTCS, **BISTDI[23:0]**, **BISTA[N:0]**, **BISTWEBUS[3:0]**, **BISTWE**, **BISTEN**, **BISTDO[23:0]**	These are external BIST signals whose use is implementation defined. These are not directly tested by the test programs, but must be tested if you intend to use these connections.
ACQCOMP, **FULL**	These are external signals whose use is implementation defined. These are not directly tested by the test programs, but must be tested if you intend to use these connections.

The test performs reads and writes to the accessible trace buffer registers, then exercises all the data and address pins available to the trace buffer RAM. It also checks the AHB handshaking signals by performing different types of transfer to and from the trace buffer RAM. The ETM interface is not exercised because this is performed by the ETM integration test.

If the test is successful, the message `** TEST COMPLETED **` is displayed using the Tube. If unsuccessful, the message `** TEST FAILED **` is displayed. Test verification is not required when running this test.

Note

This test must not be run on an ARM720T (Rev3) based system because the interaction of the ARM720T (Rev3) AHB wrapper and the ETB can cause a situation where deadlock occurs.

6.5 Simple demonstration test

A test program is supplied with the ETMIK to demonstrate the structure of a test in which the ETM is programmed to enable tracing. The process is as follows:

1. The program starts in an infinite loop at the reset vector, while the BST programs the core debug control register to enter debug state.
2. The ETM is then programmed to enable tracing, and the core is restarted to execute the code at the end of the vector table.
3. A few instructions are then executed to trace some code.

The source code is contained in `EtmIntKit/design/logical/tbench/tests/simpletest.s`.

If the test completes, the message `** TEST COMPLETED **` is displayed. Otherwise, the message `** TEST FAILED **` is displayed.

No test verification is required for this test because it is provided as an introduction to ETM programming. The ETM integration test must be run to verify ETM integration.

6.6 Source compilation

There are three stages to ensure the system is ready for simulation:

1. Environment set up
2. Verilog source compilation
3. Test program compilation.

6.6.1 Environment set up

A `dotcshrc` file is supplied in `EtmIntKit` to set up the tools used by the ETMIK. This is located in the base directory, and must be modified to point to the local location of your tools.These sections require modification:

- the simulator you intend to use and its location
- the location of your ADS tools.

After the `dotcshrc` file has been modified, you must source it using:

```
source dotcshrc
```

ModelGen Design Signoff models

If you are using ModelGen *Design Signoff Models* (DSMs),these must also be set up. Follow the instructions supplied with the DSM for the ARM core, and ETM. Install the DSMs in the `EtmIntKit/libraries/ARM/DSM` directory. If your DSMs use SWIFT models, you must ensure they are all installed to the same `LMC_HOME` location.

Embedded Trace Buffer

If you are using the ETB, this must be installed. Unpack the deliverables for the ETB to the `EtmIntKit/design/logical/etb` directory.

AHB Wrappers

If you are using an ARM7TDMI-S, ARM720T(Rev 3), ARM920T, or ARM922T core, you must unpack the AHB wrappers. After unpacking the wrappers, copy the HDL files from the relevant core directory to the directory:
`EtmIntKit/design/logical/ahb_wrapper/verilog/rtl_source`.

The ARM7TDMI wrapper contained in the AHB wrapper deliverable requires modifications as described in *AHB wrappers* on page 6-13. These modifications have already been performed, and the modified files are provided with the ETMIK in the `EtmIntKit/design/logical/arm_and_etm/verilog/rtl_source/ARM7TDMIr3` and `EtmIntKit/design/logical/arm_and_etm/verilog/rtl_source/ARM7TDMIr4` directories. These files must be left in their respective directories.

Test bench environment

In the EtmIntKit/design/logical/tbench directory, a script named setup<core> is supplied to aid in the set up of the DSMs and other environment variables for each core supported. The following modifications are required:

- Set the ARM core, and ETM DSM sections to mimic the settings required in the DSM setup. If your DSMs use SWIFT models, you must ensure they are all installed to the same LMC_HOME location.
- Ensure the rom.hex and JTAGbsi files are linked in the correct directory for the simulator you are using.
- Set the MG_LIB environment variable to point to a version of the ModelManager to support all the DSMs you are using. The models might be built with different versions of ModelGen tools. You must set the MG_LIB variable for the latest version of the ModelManager. The README supplied with the DSM shows the version of ModelGen tools used.
- Update the LD_LIBRARY_PATH environment variable as detailed in the script if you are using the Verilog-XL or Verilog-NC simulators.
- If the ARM core DSM does not have *Executed Instruction Sequence* (EIS) logging enabled by default it must be enabled, and this can be performed by setting an environment variable. If your core does require EIS tracing to be enabled, there is a relevant section in the script
- If you are implementing the ETB, ensure the environment variable DIR_ETB points to the correct location and the EtbDefs.v file is linked to the EtmIntKit/design/logical/arm_and_etm/verilog/rtl_source/include directory.

After you have modified the relevant file, it must be sourced to set up the environment, for example:

```
source setup922T
```

6.6.2 Verilog source compilation

The environment for using the simulators and the DSMs has already been set up. This section describes how to compile the model with the following simulators:

- *Using ModelSim* on page 6-27
- *Using Cadence Verilog-NC* on page 6-27
- *Using Cadence Verilog-XL* on page 6-28
- *Using VCS* on page 6-28.

Using ModelSim

Change to the EtmIntKit/design/logical/tbench/mti directory.

A script called compile_mti is supplied to ease the compilation of the system. It must be passed a parameter indicating the core for which the test bench is being compiled. This must be one of the following:

- 7TDMIr3
- 7TDMIr4
- 7TDMIS
- 720Tr3
- 720Tr4
- 920T
- 922T
- 926EJS
- 946ES
- 966ESr1
- 966ESr2
- remove

The compile script compile_mti automatically removes all compilation and simulation files already present in the directory and then compiles the HDL files for the chosen core.

There is a system<core>.vc file present for each configuration that sets any defines required by the code and sets the locations of the HDL files required for compilation. ModelSim enables .vc files to access unix environment variables so these files do not require editing except to define the ETM configuration and whether the ETB is present. The file reports where to make these changes. The remove option does not perform the compile stage and just removes the compilation and simulation files already present.

To compile the HDL, type a command similar to:

```
compile_mti 922T
```

Using Cadence Verilog-NC

This section describes how to compile the model using Verilog-NC.

Change to the EtmIntKit/design/logical/tbench/nc directory.

The compile script compile_nc automatically removes all compilation and simulation files already present in the directory and then compiles the HDL files for the chosen core.

There is a system.vc file which sets any defines required by the code and sets the locations of the HDL files required for compilation. NC does not permit.vc files to access unix environment variables so this file must be modified as detailed in the file.

To compile the HDL, type:

```
compile_nc
```

Using Cadence Verilog-XL

The Verilog-XL simulator has no compilation phase, but because it does not enable .vc files to access unix environment variables. You must modify EtmIntKit/design/logical/tbench/xl/system.vc as detailed within the file.

Using VCS

This section describes how to compile the model using VCS.

Change to the EtmIntKit/design/logical/tbench/vcs directory.

A script called compile_vcs is supplied to ease the compilation of the system which you must edit to suit your system. In the VCS compile line, the ARM core pli.tab must be directed to the environment variable indicating the location of your ARM core DSM, for example:

$DIR_ARM7TDMI/pli.tab.

This compile script automatically removes all compilation and simulation files already present in the directory and then compiles the HDL files for the chosen core.

There is a system.vc file which sets any defines required by the code and sets the locations of the HDL files required for compilation. VCS does not permit.vc files to access unix environment variables so this file must be modified as detailed in the file.

To compile the HDL, type:

```
compile_vcs
```

6.6.3 Test program compilation

A Make file is provided in EtmIntKit/design/logical/tbench/tests to enable compilation of the test program using the ADS tools. Your configuration must be customized as follows:

- Specify the correct test to the SRCFILE parameter. Use test.s if running the ETM integration test, or tbtest.s for the ETB integration test, or simpletests.s for the simple demonstration test.

- Define the correct CPU type in the ASFLAGS parameter for the target processor. Choose the appropriate value for your system:
  - ARM7TDMI for an ARM7TMDI, ARM7TMDI-S, or ARM720T processor
  - ARM9TDMI for an ARM920T, or ARM922T processor
  - ARM9E for an ARM926EJ-S, ARM946E-S, or ARM966E-S processor.
- Select the ASDEF1 parameter for the target processor. These defines are used to determine which sections of the test program are compiled. Choose the appropriate values for your system:
  - PROC SETS "ARM7TDMI" and ARCH SETS "ARMv4T" for an ARM7TDMI
  - PROC SETS "ARM720Tr3" and ARCH SETS "ARMv4T" for an ARM720T (Rev 3)
  - PROC SETS "ARM720Tr4" and ARCH SETS "ARMv4T" for an ARM720T (Rev 4)
  - PROC SETS "ARM9TDMI" and ARCH SETS "ARMv4T" for an ARM920T or ARM922T
  - PROC SETS "ARM9ES" and ARCH SETS "ARMv5TE" for an ARM946E-S or ARM966E-S
  - PROC SETS "ARM9EJS" and ARCH SETS "ARMv5TEJ" for an ARM926EJ-S.
- The ASDEF2 parameter defines where the Tube resides and which memory location responds with an error to test data and prefetch aborts. If you change the memory map or model this must be modified.
- The ASDEF3 parameter defines the port size and port mode for the ETM.
- The ASDEF4 parameter defines if half rate clocking is used by the ETM.
- The ASDEF5 parameter defines the ETM configuration. This must take a value of either small, medium, mediumplus, or large. This is used within the test program to set the threshold at which FIFOFULL stalling occurs.
- The ASDEF6 to ASDEF9 parameters are used within the test program to verify the ETM SYSOPT bus is correctly connected. Set these options for your system implementation, defining which ETM features are supported.
- The ASDEF10 parameter is required if a ETB is present in the system. Comment out this line if it is not required.

If you are using SDT 2.50, then you must adjust the rule in the Make file for bin/test.bin. You can do this by uncommenting the appropriate rule from the two that appear towards the end of the Make file. You must also comment out the default ADS rule.

To compile the program, in `EtmIntKit/design/logical/tbench/tests` type:

```
make
```

This produces some intermediate files, and most importantly the `results/test.hex` file required by the memory model and the `results/test.bsi` file used by the BST to control the JTAG port. These are linked to the correct location by the `setup<core>` script in *Test bench environment* on page 6-26.

6.7 Simulation

This section describes how to simulate the supplied test bench with the following scripts:

ModelSim A simple script is supplied to run a command-line simulation of the test bench. In the directory `EtmIntKit/design/logical/tbench/mti` type:

```
sim_mti
```

Verilog-NC The compilation stage of a Verilog-NC simulation generates a script that can be used to run a command-line simulation. In the directory `EtmIntKit/design/logical/tbench/nc` type:

```
RUN_NC
```

Verilog-XL A simple script is supplied to run a command-line simulation of the test bench. In the directory `EtmIntKit/design/logical/tbench/xl` type:

```
sim_xl
```

Using VCS A simple script is supplied to run a command-line simulation of the test bench. In the directory `EtmIntKit/design/logical/tbench/vcs` type:

```
sim_vcs
```

6.7.1 Analyzing the ETM integration test simulation results

The simulation displays the message `** TEST COMPLETED - Now proceed to Test Verification **` if it runs to completion. Then proceed to *Test verification* on page 6-33.

If the simulation fails, this is shown in a number of ways:

- The message `** UNEXPECTED EXCEPTION has occurred: **` is displayed.This means the simulation failed because an exception occurred at the wrong time. This can be an IRQ, FIQ, SWI, or an undefined instruction. The test also calls the handler to display this message if a part of the test fails. In this situation, a number can also be displayed after the semicolon in the message. This indicates the stage of the test at which it failed. Numbers 1 to 4 indicate a failure to enter debug at the correct moment, and 5 indicates the ETB failed to capture any data (or there is a ETB malfunction). The failure to enter debug can be caused by a number of reasons:
  - — incorrectly wired ETM
  - — incorrect setup of scan chains, either in wiring or software
  - — incorrectly programmed ETM
  - — incorrectly programmed EmbeddedICE watchpoints.

  The failure of the ETB can be caused by any of the following:
  - — incorrectly wired ETB

  - incorrect set-up of scan chains, either in wiring or in software
  - incorrectly programmed ETB.
- The test terminates because of BST failure.

  This means the BST either ran out of code to execute, or when testing the values of registers an incorrect value was found. If the BST runs out of code, this means the core entered debug too many times, which is caused by incorrect wiring or an access which causes the core to enter debug being effected too many times.

  The files `log.dsm_bst` and `log.bst_tube` can indicate the location and reason for this failure.
- The test never terminates.

  This can be caused by the core being halted, or executing bad instructions which cause failure. The most common cause of this is that the test program is compiled for the wrong core.

When using the ETB, the contents of the Trace RAM are read out using the JTAG port which consumes the majority of the time taken for the test. This must not be misinterpreted as a non-terminating test.

If any of these do occur, the cause of failure must be investigated, then the relevant compilation and simulation stages must be repeated.

6.7.2 Analyzing the ETB Integration test simulation results

If the test is successful, the message `** TEST COMPLETED **` is displayed on the Tube. If unsuccessful, the message `** TEST FAILED **` is displayed. Test verification is not required when running this test.

If the test fails, this is probably due to a wiring fault, or the test is incorrectly set up for the correct location of the ETB.

6.8 Test verification

This section describes the process of verifying the operation of the test program, ensuring the ETM correctly traced the processor. The steps involved are:

1. Change to the directory: `EtmIntKit/design/logical/tbench/<simulator>`.
2. Modify the `EtmCompare.cfg` file as instructed within the file.
3. Run `EtmCompare` from the command line.
4. Check the output produced for errors. If any are produced, consult the rest of this section which describes the operation of the test verification scripts:

6.8.1 Trace Comparison Scripts

The following Perl scripts are provided in the `EtmIntKit/bin` directory:

`Decomp.pl`	Trace decompression script.
`Convert.pl`	Converts an EIS output to an intermediate format.
`EtmBufExtract.pl`	Extracts the ETB output in a BST log into the compressed trace format.
`EtmModeConv.pl`	Converts the ETM trace depending on the port mode used.
`EtmCompare`	Compares the decompressed trace and EIS.

Note

`Decomp.pl`, `Convert.pl`, `EtmBufExtract.pl`, and `EtmModeConv.pl` do not have to be run manually because they are run automatically by `EtmCompare`.

6.8.2 Decompressor

The Perl script Decomp.pl takes the output from the `EtmMonitor.v` Verilog block, and an image of the code being executed, and produces a decompressed trace. It uses a number of Perl modules, `Image4b.pm`, `ImageAxf.pm`, `CF.pm`, `Coproc.pm`, `Output.pm`, and `Trigger.pm`.

The command for manually invoking the decompression script is:

```
Decomp.pl [options] compressedFile imageFile
```

The command-line options can appear in any order. The options are:

`-arch <Name>`	When ARMV5T is specified, enables decompression of ARMv5T instructions.

`-base <HexValue>`	Allows the base of a binary image to be specified, if it is not 0x0.
`-a`	Indicates that data address tracing is enabled (can be used with `-d`).
`-d`	Indicates that data tracing is enabled (can be used with `-a`).
`-PortSize <size>`	Specifies the port width setting of the trace. `<size>` must be 4, 8, or 16.
`-c <FileName>`	Specifies an optional coprocessor map file.
`-m`	Specifies whether coprocessor CPRT tracing is enabled.

The hex format of the image file is the same as the format used to be read directly into HDL memories (using `$readmemh`).

Typical decompressor script usage

The following is an example of typical decompressor script usage:

```
Decomp.pl -PortSize 8 -arch ARMV4T -a -d -m log.etm rom.hex
```

The image file format supported is hex (see the `rom.hex` file produced). These files can be generated using the `fromelf` program provided with the ARM tools, followed by the `bin2hex` Perl script provided. For example:

```
fromelf -bin -output test.bin test.elf
bin2hex test.bin test.hex
```

6.8.3 Executed Instruction Stream converter

A Perl script called `Convert.pl` is provided to take an *Executed Instruction Stream* (EIS) output file from an ARM model (ARM7TDMI, ARM9TDMI, ARM9E-S, or ARM9EJ-S) convert it into a format similar to that of the decompressed trace. It uses two Perl modules (`EISBase.pm`, and one of either `ARM7MG.pm`, `ARM9MG.pm`, or `ARM9vhd.pm`). The command for invoking the EIS converter script is:

```
Convert.pl [options]
```

The options can appear in any order. The options are:

`-eis<FileName>`	The name of the EIS file (normally `log.eis`).
`-unc <FileName>`	The name of the file produced (normally `log.eis_unc`).

`-eistype <EisFormat>`

The EIS format must be ARM7TDMI, ARM9TDMI, or ARM9x. ARM9x is the default and must be used for all ARM9 cores other than ARM9TDMI, ARM920T, and ARM922T.

Typical EIS converter script usage

The following is an example of typical decompressor script usage:

```
Convert.pl -eis log.eis -unc log.eis_unc -eistype ARM9x
```

6.8.4 Trace Buffer Extraction Script

This Perl script takes the BST log file and extracts data that is read from the ETB. The command for manually invoking the script is:

```
EtmBufExtract.pl [options]
```

The options can appear in any order. The options are:

`-bst <log_file>`

BST log file. If omitted, the default file `log.dsm_bst` is used.

`-out <log_file>`

File to output trace. If omitted, the default file `log.tracebuffer` is used.

`-size <size>` Specifies the selected port width and must be 4, 8, or 16 (default is 16 if not specified).

To ensure the `log.dsm_bst` file is properly formatted for trace capture, it searches for a line that contains `PRINTF TBCAPT`. After this point every time the command `GET 3` is executed, the script extracts the data from the value of variable 3. It makes no attempt to interpret the data, and produces incorrect data if the wrong port size is selected. When the line `PRINTF TBCAPTEND` is encountered, data extraction stops.

6.8.5 ETM Port Mode Conversion Script

The `EtmMonitor` HDL module outputs the trace data from the ETM on every clock edge and from both trace ports from the `EtmMuxDemux` module. The trace decompression script only takes trace data formatted as for a normal port mode. Therefore this script, `EtmModeConv.pl`, converts the trace output depending on the port mode in use. The command for invoking the script is:

```
EtmModeConv.pl [options]
```


The options can appear in any order. The options are:

`-trace <trace_file>` The name of the compressed trace file (default is `log.etm`)

`-out <output_file>` The name of the file the converted trace is written to (default is `log.etmnew`).

`-half` If specified, this option declares that half-rate clocking was used.

`-mode <port_mode>` This sets the port mode in use during tracing. `<port_mode>` can take the value of 0 for normal mode (default), 1 for multiplexed mode, or 2 for demultiplexed mode.

6.8.6 EtmCompare

The `EtmCompare` Perl script compares the decompressed trace with the converted EIS output, using the ETM programming information in the `JTAGbsi` and `log.dsm_bst` files, to verify that the trace is correct. It uses no modules. The command for invoking the trace comparison script is:

```
EtmCompare [options]
```

The `EtmCompare` script can take a large number of options. The options used in the validation process are:

`-TRACE_DATA_ADDRESSES 1`
Specifies data address tracing is enabled.

`-TRACE_DATA 1` Specifies coprocessor register transfer tracing is enabled.

`-TRACE_CPRT 1` Specifies coprocessor register transfer tracing is enabled.

`-PORT_SIZE <size>` Specifies the selected port width. This option is always required. <*size*> must be 4, 8, or 16.

`-trace <trace_file>` The name of the compressed trace file (default is `log.etm`).

`-image <image_file>` The name of the memory image file (default is `rom.hex`).

`-eis <FileName>` The name of the EIS file (default is `log.eis`).

`-c <coproc_map>` The name of the coprocessor map to pass to `Decomp.pl`. This file is necessary for the decompressor to successfully decompress traces involving coprocessor memory transfers (ARM instructions `LDC`, `LDCL`, `STC`, and `STCL`). It is not necessary if no external coprocessors are installed in the system (CP15 does not use these instructions).

`-ETM_BR_BCAST` Enables branch broadcast checking.

`-base <address>` Base address for image in `rom.hex` (default is 0). A common alternative is `0xFFFF0000`, if **HIVECS** is in use.

`-eistype <EisFormat>`

The EIS format, which must be ARM7TDMI, ARM9TDMI, or ARM9x. ARM9x is the default and must be used for all ARM9 cores other than ARM9TDMI, ARM920T, and ARM922T.

`-arch <arch>` The architecture version (ARMv4T or ARMv5TE).

`-BigEnd` This option is required if the test was run with the processor in big-endian mode and tracing is controlled using data comparators.

`-tbtube <address>` Alternate tube address (default `0x03000000`) `EtmCompare` looks for a write of control-D (`0x04`) to this location to signify the end of the test and terminate the comparison. This option is not necessary if you turn tracing off before the end of the test.

`-ProcIdEarly <n>` Indicates the number of cycles ProcIDs are earlier in the trace than the EIS (default is 2).

`-randomProcID` Instructs `EtmCompare` not to check ProcID values.

`-ForceComp` Force comparison even if decompressor fails.

`-jinfo <info_file>` Specify the Java bytecode information file to use.

`-debug` Output internal debugging information.

`-StartBst` This option forces the script to check the programming of the ETM on startup if it is used to trace a core out of reset.

`-tap` ***<n>*** This sets the TAP number of the ETM along the scan chain. '1' is the first device on the scan chain.

`-MayStopEarly` This enables `EtmCompare` to let the trace finish prematurely, for example if tracing was not stopped.

`-cycleacc` Cycle-accurate information.

`-FifoMustNotOverflow`

This option indicates that the ETM FIFO must not overflow. If overflow occurs when this options is set, an error is displayed.

`-PORT_MODE <port_mode>`

This sets the port mode in use during tracing. `<port_mode>` can take the value of 0 for normal mode (default), 1 for multiplexed mode, or 2 for demultiplexed mode.

`-HALF_RATE` If specified, this option declares that half-rate clocking was used.

`-TraceBuffer` If specified, this option instructs the script to extract the trace data from the BST log file, using the `EtmBufExtract.pl` script. The port mode conversion is therefore not performed if this option is selected.

`-EtmModeConv` This must be specified to run the port mode conversion script. If the Trace Buffer extraction script is being run, this switch is ignored.

The following options are also available, but are not normally used:

`-etmNoComp` Only run `Decomp.pl` and `Convert.pl`. Do not compare their outputs.

`-noDecomp` Do not run `Decomp.pl`.

`-noConvert` Do not run `Convert.pl`.

The `EtmCompare` program invokes the `Decomp` and `Convert` programs automatically. This can be suppressed using `-noDecomp` and/or `-noConvert`.

Because the number of options to be passed to `EtmCompare` can be quite large, `EtmCompare` looks for a plain text file named `EtmCompare.cfg` on startup. You must specify one command-line option per line. They are added to the beginning of the options passed on the command line. If you specify an option twice, in `EtmCompare.cfg` and on the command line, the command-line value is used. This enables you to save your default command line in `EtmCompare.cfg` and to override any selected nonstandard options on the command line. An example `EtmCompare.cfg` is included in the simulation directory `tbench/<simulator>`.

Details on how to run `EtmCompare` with the example system are given in *Running the Trace Comparison Script* on page 6-40.

Problems with EtmCompare

When you access areas of uninitialized memory, unknown values can be traced by the ETM. These unknown values are propagated through the ETM, but are converted to known values in the `EtmMonitor` module. This is because the Perl libraries used by the scripts cannot understand X values.

The side effect of this is that `EtmCompare` can generate errors when comparing the decompressed trace with the converted EIS file. You can avoid these errors by ensuring that memory addresses are initialized before they are read.

6.9 Running the Trace Comparison Script

The options passed to EtmCompare depend on the configuration of your system. You must modify and use the EtmCompare.cfg configuration file supplied in the simulator directory.

Note

When you run simulations with the ETM Integration Kit and use HIVECS, that causes the processor to boot from 0xFFFF0000, you must modify EtmCompare.cfg to use:

```
-base 0xFFFF0000
```

This is necessary to enable EtmCompare to use Decomp.pl with HIVECS.

The sections which require modification are clearly shown within the configuration file, and must be set as appropriate. To run the script, type EtmCompare at the command line in the directory EtmIntKit/design/logical/tbench/<simulator>.

Here is an example of typical EtmCompare output:

```
EtmCompare: Revision $Revision: 1.50
$EtmCompare: Running EtmModeConv.pl  -trace log.etmraw -out log.etm -mode 0
EtmCompare: Mode Conversion return code = 0
EtmCompare: Running Decomp.pl -PortSize 16 -arch ARMV4T -a -d -m log.etm rom.hex
EtmCompare: Decomp return code = 0
EtmCompare: Running Convert.pl -eis log.eis -unc log.eis_unc -eistype ARM7TDMI
EtmCompare: Convert return code = 0
EtmCompare: Comparing log.trc_unc and log.eis_unc …EtmCompare: 0 error(s)
```

If no errors are found, the trace comparison has been successful.

If EtmCompare indicates the ETM FIFO overflowed, this can be because of a number of reasons.

If using a core that does not support FIFOFULL stalling (ARM7TDMI, ARM720T, ARM920T, or ARM922T), the test program inserts NOP instructions to avoid overflow. This is suited for using a medium configuration ETM.

If overflow does occur, increase the number of NOPs before the overflow occurs, which gives the ETM FIFO more time to empty. On cores that do support FIFOFULL stalling (ARM926EJ-S, ARM946E-S, and ARM966E-S), overflow indicates incorrect wiring of the ETM FIFOFULL signal.

Note

If using a small configuration ETM, overflow is likely to occur because LDM and STM instructions are performed with data tracing enabled.

When the test runs to completion and the trace comparison scripts confirm the program has been correctly traced, the ETM is correctly verified.

6.10 Troubleshooting

If errors are found by the trace comparison scripts, you can use the log files produced by the simulation and `EtmCompare` to discover where the fault lies:

- *Log files*
- *EtmCompare Errors*
- *Information messages* on page 6-43.

6.10.1 Log files

This section lists the log files produced by the simulation and `EtmCompare`:

`log.eis` The EIS trace produced by the ARM core.

`log.eis_unc` The EIS trace in an uncompressed format suitable for `EtmCompare`. This file is produced by the `Convert.pl` script.

`log.etmraw` The file produced by the `EtmMonitor` module containing raw trace data.

`log.etm` The properly formatted trace file produced by the `EtmModeConv.pl` script from the `log.etmraw` file.

`log.trc_unc` The ETM trace in an uncompressed format suitable for `EtmCompare`. This file is produced by the `Decomp.pl` script.

`log.tube` A log of messages printed by the BST (This is not the same as the tube used for displaying messages to the screen).

`log.dsm_bst` A log of all the commands executed by the BST. This is useful for checking what the BST executed during a test, to discover whether the JTAG port is correctly connected.

`log.diff` Displays the comparison between the EIS trace and the ETM trace. Errors are noted in the leftmost column, which can be used to determine the cause of failure. A brief description of each error is given in *EtmCompare Errors*.

6.10.2 EtmCompare Errors

There are two types of message produced in the left-most column of the file `log.diff`.

- error messages
- information messages.

Messages that begin with `ERROR` are error messages. They are classified as follows:

`ERROR +` A line appeared in the trace when TraceEn was off.

`ERROR -` A line failed to appear in the trace when TraceEn was on.

`ERROR -VD` A line appeared with no data tracing when ViewData was on.

`ERROR M` A line appeared in the trace on cue, but after adjustment for data tracing and so on, failed to match that from the EIS. This is followed by two lines indicating the actual text that `EtmCompare` compared, after modifications (for example by ViewData).

`ERROR PIDM` A ProcID of the wrong size or value was broadcast.

`ERROR E<` An exception (not an IRQ, FIQ, or Prefetch Abort) occurred, but the trace indicated it an instruction early. `INF3` messages are printed when an IRQ, FIQ, or Prefetch Abort occurs.

`ERROR INT+` An interrupted instruction was incorrectly traced due to the previous instruction being traced (ARM7TDMI-based system).

`ERROR SYNC` Synchronizing encountered in `log.trc_unc`. This means the decompressor could not understand the trace protocol error).

`ERROR ?` Could not understand a line in `log.trc_unc` or `log.eis_unc`.

6.10.3 Information messages

Messages that begin with `INF` are information messages. They are classified as follows:

`INF1` A purely informational message, either to debug what `EtmCompare` is thinking or to mark cases of interest.

`INF2` A special case that is allowed by the specification.

`INF3` A known and documented specification violation.

`EtmIntKit/documentation/EtmCompare.txt` gives a full description of the different types of `INF` messages.

Chapter 7
Software Considerations for Trace

This chapter describes software issues relating to the ETM9. It contains the following sections:

7.1 Tracing dynamically loaded images

Support for dynamically-loaded images with the ETM9 (Rev 0/0a) is only possible with instrumented code, collusion of the OS, and additional support in the debugger.

ETM9 (Rev 1) and above includes specific hardware support for tracing context IDs and overlay numbers. However, software support is still required.

7.1.1 Why dynamically-loaded code requires special hardware and software support

When a debugger is debugging a system it talks to the system mainly in terms of addresses on the system (possibly virtual) memory. To be useful to its user it must translate between these addresses and locations in the images loaded on the system. This enables it to present a symbolic or source level view of the code running on the system to the user.

In a simple statically linked and loaded system the system runs a single image. The user tells the debugger the name of this image, and the image describes the mapping between target addresses as image locations. The debugger requires no further information to debug the image.

Many systems, including *Operating System*s (OS) such as WinCE, Linux, or Epoc32, load part or all of their software dynamically. This can have several effects:

- the address at which an image is loaded might not be known until it is loaded
- at different times different images might be loaded at the same address
- in a complex system the debugger might not even know what images are candidates to be loaded until they are loaded.

To debug such a system the debugger must be able to ask the target system what images are loaded and where. At present ARMs debugger cannot ask such questions.

The problem is more difficult when using trace, because trace contains historical information. When analyzing a trace the debugger has to know what images were loaded when the trace data was collected, rather than what images are loaded now. Additionally, even without a valid image, you can do some very basic, but sometimes useful interactive debugging, such as single-stepping instructions. The compression algorithm used for trace data means that the debugger cannot start to decode trace data unless the images that were loaded when it was generated are available.

In particular, you must remember that, for ALL embedded trace solutions to work, an image of the code being executed must be available to the trace decompression software of the debugger. This is because the instructions being executed are not output, because of the data bandwidth that is required, and so only the minimum of address information

is traced. This means that, given a (compressed) address issued by the trace port, the tools must be able to know what instructions are at and around that point. This enables the target address of direct branches (`B` and `BL` instructions in the case of ARM) to be implied. Virtual memory and software paging, (effectively self-modifying code) for example, make this hard because the debugger probably does not know where the code ends up being executed from.

For a device with a full MMU, and an operating system that loads application and OS code into arbitrary locations in RAM, a way of telling the debugger what the code image for a particular trace sequence is required. The side-effect of this is that the closer to a physical address that can be supplied the better. For the ARM920T core the chosen solution is to output the modified virtual address on the trace interface.

7.1.2 ETM9 (Rev 1 and above) hardware support

ETM9 (Rev 1) and above provides the ability to trace a variable length context identifier field, whenever tracing is enabled, and as part of the periodic address output. This enhancement enables simpler cooperation between the target operating system and the trace debug tools.

7.2 Simple overlay support

A system for supporting simple overlays is possible that does not require specific support within the trace debug tools. This solution is based on the requirement that the memory space into which the overlays are loaded exists in multiple places in the memory map.This system works in the same way that the *Tightly Coupled Memory* (TCM) address space can be used to alias a TCM address range throughout the whole TCM address space. That is, some of the unused address bits are *don't care* when determining the memory to be accessed.

For example, if you have 16KB of SRAM, bits [13:0] of the address determine the 32-bit word to access and bits [31:24] determine when to access that particular block. However, if bits [15:14] are in the address decoder, four copies of the memory block exist in the memory map. In other words the same word can be accessed using four different addresses. That is, when bits [15:14] of the address are 00, 01, 10, or 11 as shown in Figure 7-1 on page 7-5.

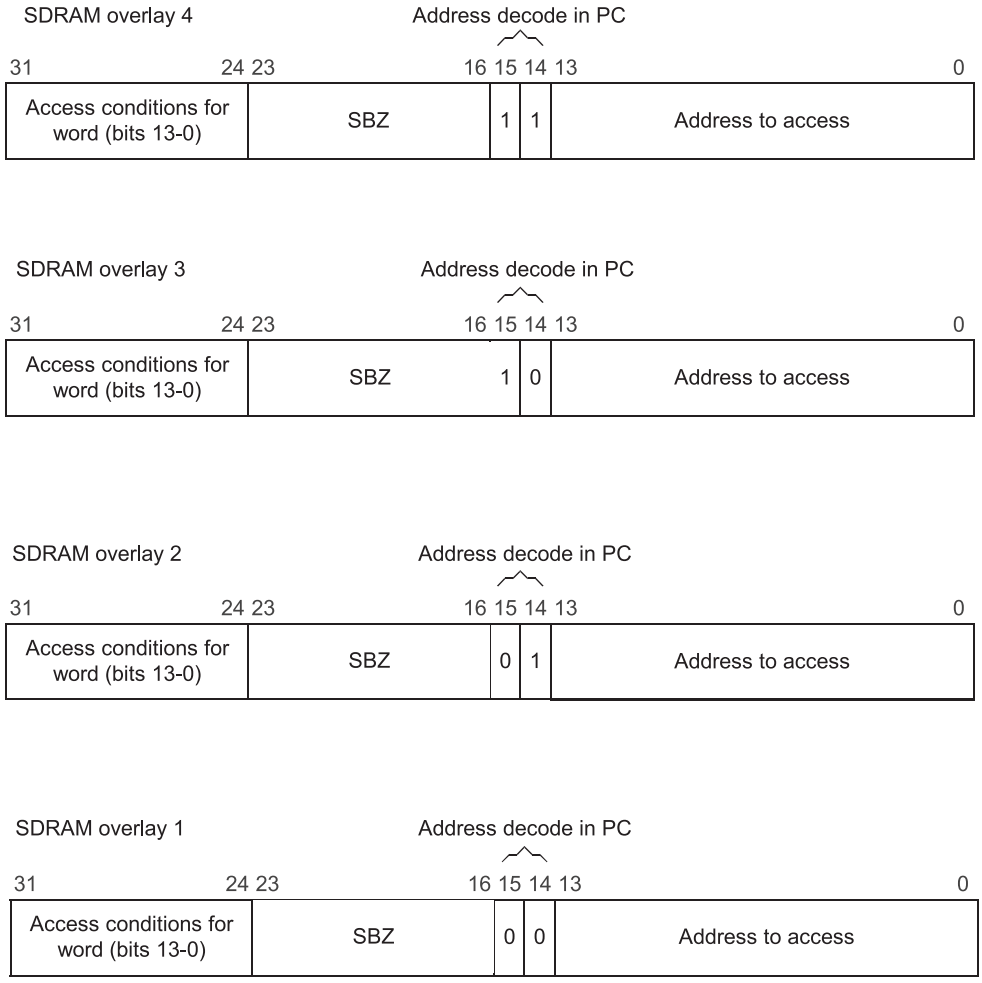


Figure 7-1 SDRAM example overlays

If this 16KB block is to hold overlays, you can use bits [15:14] to indicate which of four possible overlays are loaded into this memory block. The value of bits [15:14] of the program counter is traced by the ETM. In the trace tools a static image of the code being executed, with the four possible overlays statically linked (using scatter loading in the ARM tools) into the appropriate 16KB blocks of memory space, enables the trace decompression tools to successfully decompress the trace.

When the overlay manager loads and calls a new overlay, it copies the code into RAM and then jumps to the overlay. Bits [15:14] of the address, loaded into the PC, are set appropriately. The physical address identifies the location in memory. The virtual address uniquely identifies the data to access and the location within that overlay.

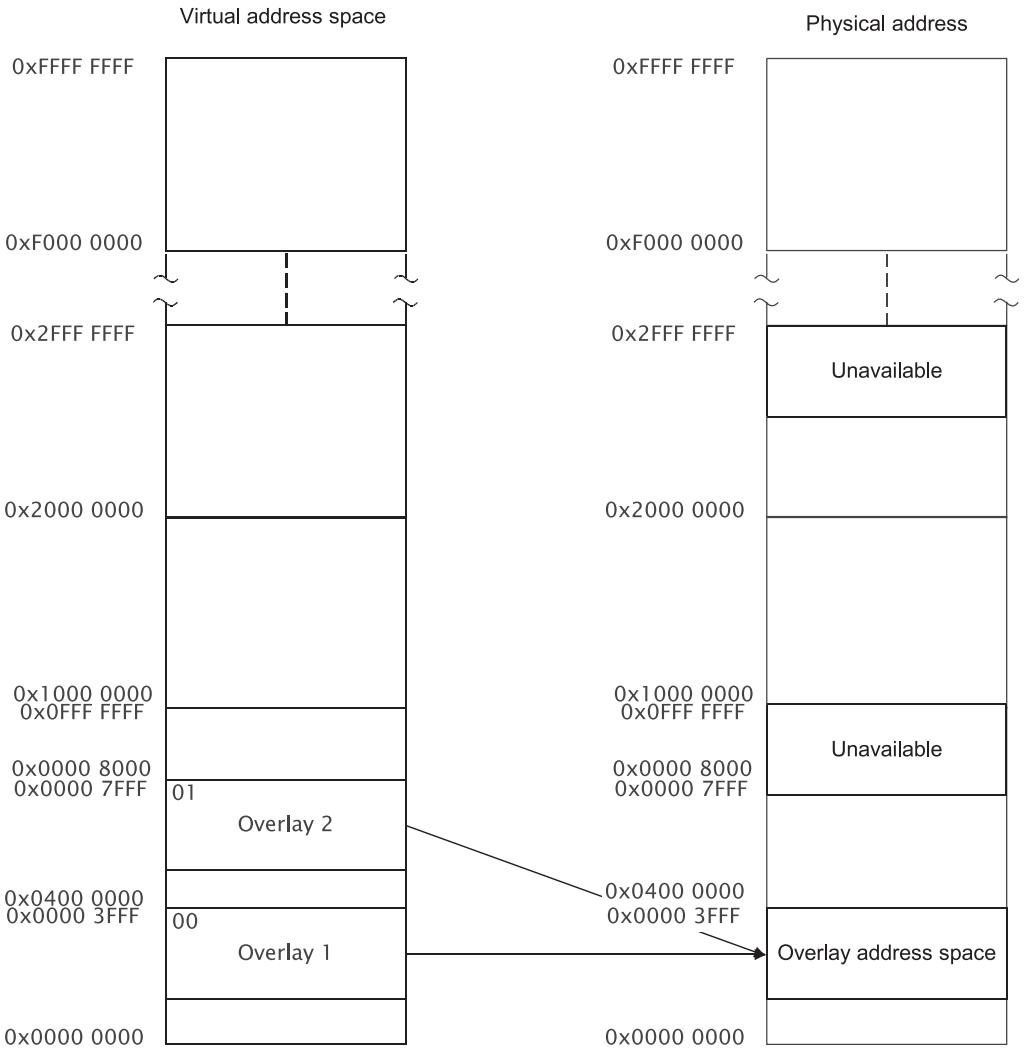


Figure 7-2 Mapping overlays to a physical address

7.2.1 Overlay support on ARM926EJ-S, ARM946E-S, and ARM966E-S macrocells

ARM926EJ-S, ARM946E-S, and ARM966E-S macrocells support tightly-coupled instruction and data RAM on the core. This is aliased to multiple addresses in the processor address map so that overlay support is automatic for code loaded into this space.

Chapter 8
Physical Trace Port Signal Guidelines

This chapter contains some signal guidelines that can ensure correct operation of the ETM and trace tools. It contains the following sections:

8.1 About trace port signal quality

Guaranteed operation of the TPA or logic analyzer depends on correct design of the ASIC and of the target PCB.

When integrating an ETM into an ASIC, the quality and timing of the trace port signals to the TPA are critical for reliable operation. Some of the issues to consider are:

- output pad selection
- PCB track lengths
- PCB track termination
- setup and hold times for the trace data signals with respect to **TRACECLK**.

The importance of these issues is directly proportional to the operating frequency. At frequencies greater than 100MHz, careful SPICE analysis of the system including the characteristics of the package and the chosen TPA is recommended.

8.2 ASIC pad selection, placement, and package type

The position and type of ASIC pad that you select depends on the following factors:

- the ability to minimize the noise and coupling between trace and other signals
- the ability to drive the external load.

The quality of the **TRACECLK** signal, as observed by the TPA, has the greatest effect on the reliability of the system. It is vital that **TRACECLK** transitions move cleanly through the threshold region of the input circuitry of the TPA, without glitches or ringing.

With certain types of package and pin placement (for example, pads on the corner or the edge of a package), the signal coupling between the trace data signals and the trace clock can be significant. If this problem is encountered during simulations, place **GND** or static I/O signals on both sides of the **TRACECLK** signal.

The quality of the package, and specifically the presence or absence of a ground plane in the package, can significantly affect the quality of the output signal. In general, ASIC pads are specified in terms of current drive and signal slew rate. For calculating the PCB signal quality you are likely to also have to determine:

- the signal rise and fall times
- the pad output impedance.

Note

Matched impedance output pads give a significantly improved performance.

You must also consider the pad placement, to ensure that the PCB tracking to the trace port connector is possible. You are recommended to place the pads so that they are:

- on the outside of the package
- grouped together
- in the same order as the connector.

8.3 PCB design guidelines

Two implementations are possible:

A dedicated trace port

The TPA or logic analyzer is the only load on the trace port signals. See *Dedicated trace port*.

A shared trace port

The trace pins are shared with other functions, and therefore there are stubs on the PCB tracks of the development board and an increased load on the output driver. See *Shared trace port* on page 8-6.

8.3.1 Dedicated trace port

This is the preferred implementation for connecting a TPA to a trace port. The TPA is the only load on the nodes connected to target ASIC pins, so the only factor affecting operation is signal integrity at the TPA connector.

If you know the characteristics of your PCB tracks, use the actual trace impedance and propagation delay. If you do not have access to this information, use the following guidelines for microstrip (track on outer layer over a ground plane) on FR4 PCB:

- Propagation speed is typically 160ps/inch (approximately 63ps/cm).
- The impedance of a 0.005-inch wide track as a microstrip is between 70Ω to 75Ω on a typical six-layer foil construction board. The impedance of a track reduces as the width of the track increases.

To design the target system effectively, you must know:

- the characteristic impedance and signal edge rates of the ETM output drivers
- the actual setup and hold provided by the ASIC ETM outputs with reference to the ETM **TRACECLK**.

If you do not know the characteristics of the signals from your ASIC, consult your ASIC vendor. It is difficult to provide any general rule because ETM output drivers and timings vary between ASIC vendors.

PCB track length

You must match all **TRACECLK**, **PIPESTAT[2:0]**, **TRACESYNC**, and **TRACEPKT[15:0]** track lengths between the ASIC and the trace port connector within 100ps. Overall differences in track lengths directly impact setup and hold requirements as follows:

- if the clock is delayed compared to the data, you must increase the setup specification by the additional clock delay
- if any data is delayed compared to the clock, you must add the delay to the setup requirement
- if data paths are such that data has both greater than and less than delays compared with the clock, you must add the difference to both the setup and hold specification.

Signal quality

The primary variable that characterizes signal quality is the rise time of a signal compared to its propagation time. It is this relationship that affects the track length, and this is where the minimum signal rise and fall time becomes important.

To ensure accurate data acquisition, you must minimize all reflections, overshoot, and undershoot. Aim to keep the one-way propagation time for all tracks at less than one third of the signal rise time.

As the fabrication process for your ASIC improves, your output driver is likely to improve and your rise and fall times are likely to decrease. If you cannot keep the propagation time for all tracks below one third of the signal rise time, some form of signal termination is required. This can be either of the following:

Series termination (Recommended method.) The series resistor must be placed as close as possible to the ASIC pin (half an inch or closer). The value of this series resistor plus the output impedance of the signal driver must closely match the impedance of the PCB track.

Parallel or matched AC termination

If you cannot use series termination, add parallel or matched AC termination on each signal track at the TPA target header. This requires significantly more power from the ASIC, and the AC termination must closely match the frequency and rise time of the terminated signal. In practice therefore, parallel termination is rarely possible.

If the total track length is one rise time propagation delay or greater in length, follow standard high-speed design practices to minimize cross talk between the clock and the data signals. (The total track length is the target PCB track length plus any PCB track on the TPA buffer board.)

Note

ASIC output pads with an output impedance that is matched to the PCB track might be available from your ASIC vendor. If these are used, the signal quality of the trace port signals is significantly improved.

8.3.2 Shared trace port

Some applications might not have enough pins available for trace, so you might have to multiplex trace signals with other functions. This has the effect of increasing the load on the trace signals, unless a specific trace-only development board is built.

When an ETM output pin is multiplexed with other functions, the addition of the TPA target header can add a stub to the PCB track on the target system. When this happens, the following additional constraints apply. (The goal is to minimize the effect of the TPA target header on non TPA-based signal usage and maintain the integrity of the trace measurements.):

Signal does not require termination in normal operation or is parallel-terminated

This means that a full voltage swing signal travels down the track. Ensure that the propagation delay of the stub added for the TPA target header is 20 per cent or less of the overall rise and fall time of the signal.

Signal is series terminated for normal operation

This means that a one-half voltage swing signal begins each transition on the track and propagates down the track until it is terminated at the target node. This case is potentially very problematic. The one-half voltage swing signal can maintain the TPA input at its threshold voltage for longer than the required rise and fall time. To prevent this, you must move the TPA target header to within one fifth of the rise time of the target end of the track. If this is not possible, you must slow down the rise and fall times until this requirement can be met.

Sometimes, board layout constraints make it impossible to keep the stubs to the trace target header to less than 20 per cent of the rise and fall time. If length and speed requirements do not enable the rise and fall times to be increased to meet the design

requirements in this case, you can adjust the thresholds used by the TPA or LA, if supported. The target system must be able to provide:

- sufficient noise margin around an altered threshold
- sufficient setup and hold times because these are now reduced.

Note

It is unlikely that any TPA supplier can guarantee support for this mode of operation.

8.4 EMI compliance

If you follow the guidelines in *Trace signal output timing* on page 3-29, the trace port pins, including the **TRACECLK**, are inactive. This means that the trace port does not affect your EMI compliance testing. The trace port pins are active only when the trace debug tools are connected to the target.

It might be useful to carry out some testing with the trace port enabled, to determine the effect of the trace port switching on overall system noise.

8.5 Further references

Many TPA vendors provide models for download from the internet. These models enable you to use SPICE-like tools to analyze the signal integrity at the point that it is sampled by the TPA or logic analyzer:

Agilent The Agilent web site enables you to download data on their TPA and LA products. You can use the search engine on the web site to look for pages and documents that refer to ETM. For example, the document *Trace Port Analysis for ARM ETM* (Agilent document number E5903-97002) contains equivalent models for Agilent TPA and Logic Analyzer products.

Tektronix The Textronix web site has a number of documents relating to the use of their Logic Analyzers for acquiring trace. For example, the document *P6434 Mass Termination Probe* (Tektronix document number 070-9793-02) provides models for the equivalent load of the Logic Analyzer probe.

Other vendors

Details about TPA vendors are added to this document as they become known to ARM Limited. You can also contact your chosen vendor directly for the latest information.

Appendix A
Signal Descriptions

This chapter describes the signals used in the ARM9 *Embedded Trace Macrocell* (ETM9). It contains the following section:

- Signal descriptions on page A-2.

A.1 Signal descriptions

The signal descriptions for ETM9 (Rev 2 and above) are shown in Table A-1.

Table A-1 ETM9 signals

Type	Signal name	Clock domain	Description
Input	**ARMTDO**	TCK	The **TDO** output signal from the ARM macrocell, or from an external scan chain.
Input	**BIGEND**	-	The signal driving the ARM **BIGEND/CFGBIGEND** input. When HIGH the processor treats bytes in memory as big-endian format. When LOW memory is treated as little-endian. For processors that permit the system control processor to change the endian behavior this signal toggles with the change. Otherwise this is a static configuration signal.
Input	**CHSD[1:0]**	CLK	The coprocessor handshake Decode bus driven into the ARM macrocell.
Input	**CHSE[1:0]**	CLK	The coprocessor handshake Execute bus driven into the ARM macrocell.
Input	**CLK**	-	This clock times most operations in the ETM9. All outputs change from the rising edge and all inputs are sampled on the rising edge. You can stretch the clock in either phase. You can add synchronous wait states using the **CLKEN** signal. **Note** For processors where **TCK** and **CLK** are asynchronous the signals in the **TCK** domain are completely unrelated to **CLK**.
Output	**CLKDIVTWOEN**	TCK	If HIGH, indicates that the ETM9 is in half-rate clocking mode.
Input	**CLKEN**	CLK	The ETM9 can be stalled by driving **CLKEN** LOW. This signal must be held HIGH at all other times. The **CLKEN** signal drives the ARM macrocell **nWAIT/CLKEN** input.
Input	**DA[31:0]**	CLK	The processor data address bus driven by the ARM macrocell.
Input	**DABORT**	CLK	The Data Abort signal driven into the ARM macrocell. The **DABORT** signal tells the processor that the requested data memory access is not allowed.
Input	**DBGACK**	CLK	The debug acknowledge signal driven by the ARM macrocell. When HIGH this signal indicates that the ARM macrocells in debug state.
Output	**DBGRQ**	CLK	Debug request. You can use this signal to stop the ARM processor.

Table A-1 ETM9 signals (continued)

Type	Signal name	Clock domain	Description
Input	**DD[31:0]**	CLK	The **DD/WDATA** bus driven by the ARM macrocell.
Input	**DDIN[31:0]**	CLK	The **DDIN/RDATA** bus driven into the ARM macrocell.
Input	**DMAS[1:0]**	CLK	The data memory access size bus driven by the ARM macrocell. These encode the size of a data memory access in the following cycle.
Input	**DnMREQ**	CLK	The data memory request signal driven by the ARM macrocell. If LOW at the end of a cycle then the processor requires a data memory access in the following cycle.
Input	**DnRW**	CLK	The data read write signal driven by the ARM macrocell. If LOW at the end of a cycle then any data memory access in the following cycle is a read. If HIGH then it is a write.
Input	**DSEQ**	CLK	The data sequential address signal driven by the ARM macrocell. If HIGH at the end of the cycle then any data memory access in the following cycle is sequential from the last data memory access.
Output	**ETMEN**	TCK	This output is HIGH when the debugger has enabled the ETM.
Input	**EXTIN[3:0]**	CLK	External inputs to the ETM. For example, from signals of interest within the ASIC. Different sizes are described in *Changes to the programmer's model in Rev 2* on page 1-10 **Note** The External Inputs **EXTIN[3:0]** must be synchronous to the ETM9 **CLK**
Output	**EXTOUT[3:0]**	CLK	External outputs from the ETM. Can be used to trigger hardware inside the ASIC, or external equipment such as a logic analyzer.
Output	**FIFOFULL**	CLK	When enabled, this indicates that there is less than a user-programmed number of bytes in the ETM9 FIFO.
Input	**HIVECS**	-	The signal driving the ARM **HIVECS/CFGHIVECS** input. When LOW the ARM exception vectors start at address `0x0000 0000`. When HIGH the ARM exception vectors start at address `0xFFFF 0000`. For processors that permit the system control processor to change the HIVECS setting this signal toggles with the change. Otherwise this is a static configuration signal.
Input	**IA[31:0]**	CLK	The instruction address bus driven by the ARM macrocell.

Table A-1 ETM9 signals (continued)

Type	Signal name	Clock domain	Description
Input	**ID15To11[15:11]**	CLK	A section from the **ID/INSTR** input bus driven into the ARM macrocell.
Input	**ID31To25[31:25]**	CLK	A section from the **ID/INSTR** input bus driven into the ARM macrocell.
Input	**IJBIT**	CLK	The **IJBIT** signal driven by the ARM macrocell. When HIGH, denotes that the ARM processor is in Java state. When LOW the processor is in ARM or Thumb state. This signal is valid with the address.
Input	**InMREQ**	CLK	The **InMREQ** signal driven by the ARM macrocell. If LOW at the end of the cycle then the processor requires an instruction memory access during the following cycle.
Input	**INSTREXEC**	CLK	The **INSTREXEC/DBGINSTREXEC** pipeline status signal driven by the ARM macrocell. The instruction executed signal indicates that the instruction in the Execute stage of the pipeline follower of the ETM9 has been executed.
Input	**INSTRVALID**	CLK	The **DBGINSTRVALID** pipeline status signal driven by the ARM macrocell. The instruction valid signal indicates that the instruction in the Execute stage is valid, and has not been flushed.
Input	**ISEQ**	CLK	The **ISEQ** signal driven by the ARM macrocell. If HIGH at the end of the cycle then any instruction memory access during the following cycle is sequential from the last instruction memory access.
Input	**ITBIT**	CLK	The **ITBIT** signal driven by the ARM macrocell. When HIGH, denotes that the ARM processor is in Thumb state. When LOW the processor is in ARM state. This signal is valid with the address.
Input	**LATECANCEL**	CLK	The coprocessor late cancel signal driven by the ARM macrocell. If HIGH during the first memory cycle of a coprocessor instruction, then the coprocessor must cancel the instruction without changing any internal state. This signal is only asserted in cycles where the previous instruction accessed memory and a Data Abort occurred.
Output	**MMDCTRL[7:0]**	CLK	A control bus, used to reconfigure the memory map decode logic.
Output	**MMDDA[31:0]**	CLK	The ARM **DA[31:0]** signal, pipelined for the memory map decode interface.
Output	**MMDInMREQ**	CLK	The ARM **InMREQ** signal, pipelined for the memory map decode interface.

Table A-1 ETM9 signals (continued)

Type	Signal name	Clock domain	Description
Output	**MMDDnMREQ**	CLK	The ARM **DnMREQ** signal, pipelined for the memory map decode interface.
Output	**MMDDnRW**	CLK	The ARM **DnRW** signal, pipelined for the memory map decode interface.
Output	**MMDIA[31:1]**	CLK	The ARM **IA[31:1]** signal, pipelined for the memory map decode interface.
Input	**MMDIN**	CLK	Decoded MMD resources. Different sizes are described in *Pin names* on page 1-5.
Output	**MMDITBIT**	CLK	The ARM **ITBIT** signal, pipelined for the memory map decode interface.
Input	**MUXINSEL**	**TCK**	1 = Test wrapper in INTEST mode Selects the test wrapper scan chain as the source for ETM9 inputs. If MUXINSEL is set to 1 MUXOUTSEL must be set to 0.
Input	**MUXOUTSEL**	**TCK**	1 = Test wrapper in EXTEST mode. Selects the test wrapper scan chain as the source for ETM9 outputs. If **MUXOUTSEL** is set to 1 **MUXINSEL** must be set to 0.
Input	**nRESET**	CLK	Active LOW ETM9 reset.
Input	**nTRST**	TCK	Active LOW JTAG test reset.
Input	**PASS**	CLK	The **PASS** coprocessor signal driven by the ARM macrocell. This signal indicates that the instruction in the Execute stage of the pipeline follower of the ETM9 is executed.
Output	**PIPESTAT[2:0]**	CLK	Indicates the pipeline status of the ARM macrocell.
Output	**PORTMODE[1:0]**	CLK	This output bus enables the on-chip trace port output logic to be configured for normal, multiplexed, or demultiplexed modes of operation.
Output	**PORTSIZE[2:0]**	CLK	Indicates the currently selected port size in use on the **TRACEPKT[15:0]** bus. 000 = 4-bit port 001 = 8-bit port 010 = 16-bit port 011 to 111 = reserved.

Table A-1 ETM9 signals (continued)

Type	Signal name	Clock domain	Description
Input	**PROCID[31:0]**	CLK	This bus provides a copy of the current context ID or overlay number from the ARM system control coprocessor or peripheral.
Input	**PROCIDWR**In	CLK	This signal must be asserted whenever the **PROCID** bus changes. This causes the ETM to output the new context ID at the next available opportunity.
Output	**PWRDOWN**	TCK	When HIGH, indicates that the ETM9 can be powered down.
Input Input	**RANGEOUT[0],** **RANGEOUT[1]**	CLK	The **RANGEOUT[0]**, **RANGEOUT[1]**, and **DBGRNG[1:0]** EmbeddedICE signals driven by the ARM macrocell. The EmbeddedICE **RANGEOUT** signals indicate that the corresponding watchpoint unit has matched the conditions currently present on the address, control and data buses. These signals are independent of the state of the enable control bit of the watchpoint unit.
Input	**SCANMODE**	CLK or TCK	Selects between functional mode and test mode: Functional mode **TCK** Scan mode **CLK**
Input	**SYSOPT[8:0]**	-	Indicates to the debug tools the system options that have been implemented. Bits are tied HIGH or LOW, as appropriate, as part of the integration process.
Input	**TCK**	TCK	Test clock.
Input	**TCKEN**	TCK	Synchronous enable for test clock.
Input	**TDI**	TCK	Test data input.
Output	**TDO**	TCK	Test data output.
Input	**TMS**	TCK	Test mode select.
Output	**TRACEPKT[15:0]**	CLK	The trace packet port.
Output	**TRACESYNC**	CLK	A synchronization signal, indicating the start of a branch sequence on the trace packet port.
Input	**WEDGE**	TCK	Controls which edge the wrapper chain output activates on: 1 = activate on rising edge 0 = activate on falling edge.
Input	WSEI	TCK	Enables scanning of data through the test wrapper scan chain inputs.

Table A-1 ETM9 signals (continued)

Type	Signal name	Clock domain	Description
Input	WSEO	TCK	Enables scanning of data through the test wrapper scan chain outputs.
Input	WSI	TCK	Serial input data for the test wrapper scan chain.
Output	WSO	TCK	Serial output data from the test wrapper scan chain.
Input	**ZIFIRST**	CLK	Asserted when in Java state on the first ARM instruction to be traced. (No more than two instructions are ever traced for a bytecode.)[a]
Input	**ZILAST**	CLK	Asserted when in Java state on the last ARM instruction to be traced. (No more than two instructions are ever traced for a bytecode.)[a]

a. If only one ARM instruction is traced in Java state, both **ZIFIRST** and **ZILAST** are asserted.

Appendix B
Integrating the EtmMuxDemux Block into ETM9

This appendix describes how to integrate the `EtmMuxDemux` block into an ETM9. It contains the following section:

- *Using the EtmMuxDemux block* on page B-2.

B.1 Using the EtmMuxDemux block

The `EtmMuxDemux` module takes the standard trace port interface from the ETM9 and, using the **CLK**, **PWRDOWN**, **ETMnRESET**, PORTMODE, and **CLKDIVTWOEN** outputs, converts the trace to the correct port mode. The example HDL for configuring the trace port is not part of the ETM9 macrocell HDL. It is provided in an additional file `ETM9/EtmMuxDemux.v` in the ETM9 HDL directory.

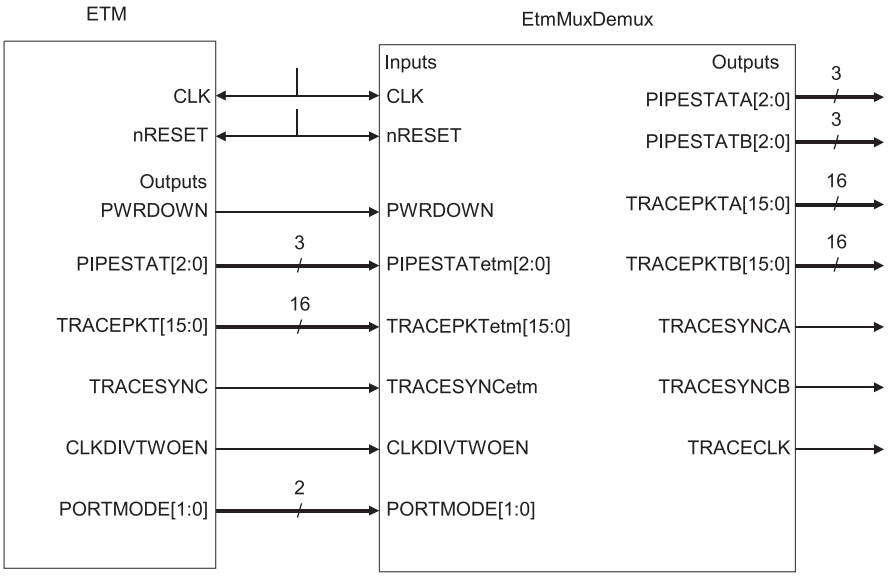


Figure B-1 ETM to EtmMuxDemux connections

Refer to *Clocks and resets* on page 3-12 for more information on **CLK** and **nRESET**.

Note

- When instantiating your `EtmMuxDemux` module, ensure that you wire all 16 **TRACEPKT** signals to all 16 **TRACEPKTetm[15:0]** signals.
- Ensure that the `EtmMuxDemux` **CLK** input is wired to the ETM **CLK** and not the core clock.
- Ensure that `EtmMuxDemux` **nRESET** is wired to the ETM **nRESET** and not the core reset.

Glossary

This glossary describes some of the terms used in technical documents from ARM Limited.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave IP developments and ARM Limited recommends only a subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

Advanced Peripheral Bus (APB)
A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

AHB *See* Advanced High-performance Bus.

AMBA *See* Advanced Microcontroller Bus Architecture.

Application Specific Integrated Circuit (ASIC)
An integrated circuit that has been designed to perform a specific application function. It can be custom-built or mass-produced.

Application Specific Standard Part/Product (ASSP)
An integrated circuit that has been designed to perform a specific application function. Usually consists of two or more separate circuit functions combined as a building block suitable for use in a range of products for one or more specific application markets.

ASIC *See* Application Specific Integrated Circuit.

ASSP *See* Application Specific Standard Part/Product.

ATPG *See* Automatic Test Pattern Generation.

Automatic Test Pattern Generation (ATPG)
The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

Clock gating Gating a clock signal for a macrocell with a control signal and using the modified clock that results to control the operating state of the macrocell.

Debugger A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.

Design Simulation Model (DSM)
A functional simulation model of the device that is derived from the *Register Transfer Level* (RTL) but that does not reveal its internal structure. The DSM does not model any features added during synthesis such as internal scan chains. The DSM provides higher speed for functional simulation than that of the Sign-Off Model (SOM).

DSM *See* Design Simulation Model.

Embedded Trace Macrocell (ETM)
A hardware macrocell that, when connected to a processor core, outputs instruction and data trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol.

ETM See *Embedded Trace Macrocell*.

Half-rate clocking (ETM)
: Dividing the trace clock by two so that the TPA can sample trace data signals on both the rising and falling edges of the trace clock. The primary purpose of half-rate clocking is to reduce the signal transition rate on the trace clock of an ASIC for very high-speed systems.

Joint Test Action Group (JTAG)
: The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.

JTAG
: *See* Joint Test Action Group.

Macrocell
: A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.

Modified Virtual Address (MVA)
: A Virtual Address produced by the ARM processor can be changed by the current Process ID to provide a *Modified Virtual Address* (MVA) for the MMUs and caches.

  See also Fast Context Switch Extension.

MVA
: *See* Modified Virtual Address.

SCREG
: The currently selected scan chain number in an ARM TAP controller.

SPICE
: Simulation Program with Integrated Circuit Emphasis. An accurate transistor-level electronic circuit simulation tool that can predict how an equivalent real circuit behaves for given circuit conditions.

TAP
: *See* Test access port.

Test Access Port (TAP)
: The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.

Trace driver
: A Remote Debug Interface target that controls a piece of trace hardware. That is, the trigger macrocell, trace macrocell, and trace capture tool.

Trace hardware
: A term for a device that contains an Embedded Trace Macrocell.

Trace port
: A port on a device, such as a processor or ASIC, used to output trace information.

Trace Port Analyzer (TPA)
: A hardware device that captures trace information output on a trace port. This can be a low-cost product designed specifically for trace acquisition, or a logic analyzer.