ETM10RV™

Revision: r0p0

Technical Reference Manual

ARM DDI 0245B

ETM10RV
Technical Reference Manual

Release Information

Change history

Date	Issue	Change
03 September 2002	A	First release r0p0
30 September 2003	B	Second release r0p0. Updated with correction to Table 4-5.

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
ETM10RV Technical Reference Manual

List of Tables
ETM10RV Technical Reference Manual

List of Figures
ETM10RV Technical Reference Manual

Preface

This preface introduces the *ARM10™ Embedded Trace Macrocell RealView™* (ETM10RV) *Revision r0p0 Technical Reference Manual*. It contains the following sections:

- *About this manual* on page x
- *Feedback* on page xv.

About this manual

This is the technical reference manual for the *ARM10 Embedded Trace Macrocell RealView* (ETM10RV) Revision r0p0. This product is referred to as ETM10RV throughout this manual.

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.

Intended audience

This manual is written for hardware and software engineers who want to incorporate an ARM1026EJ-S core based product with instruction and data trace capability into their hardware and software design.

Using this manual

This manual is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the ETM10RV macrocell.

Chapter 2 *Accessing the ETM10RV Registers*

Read this chapter for information about programming the registers that control the ETM10RV macrocell.

Chapter 3 *Integrating the ETM10RV*

Read this chapter for information about integrating the ETM10RV macrocell with an ARM1026EJ-S processor.

Chapter 4 *Design for Test*

Read this chapter for details of the *Design for Test* (DFT) features of the ETM10RV macrocell, including how to select a test mode.

Chapter 5 *Implementation-defined Behavior*

Read this chapter for additional implementation-specific information relating to ETM10RV macrocell.

Chapter 6 ***Tracing Dynamically-loaded Images***

Read this chapter for a description of the software issues that relate to the ETM10RV macrocell.

Chapter 7 ***Physical Trace Port Signal Guidelines***

Read this chapter for guidance about output pad selection and PCB design.

Appendix A ***Signal Descriptions***

Read this appendix for a description of the ETM10RV macrocell functional and DFT signals.

Conventions

Conventions that this manual can use are described in:

- *Typographical*
- *Timing diagrams* on page xii
- *Signals* on page xii
- *Numbering* on page xiii.

Typographical

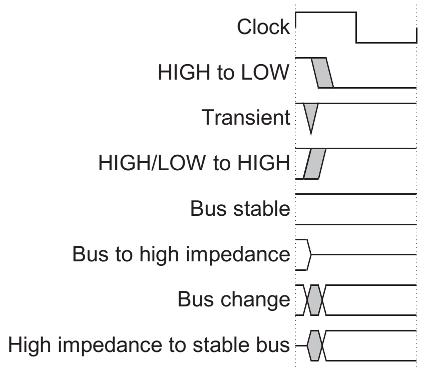
The typographical conventions are:

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

< and >	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example: • `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>` • The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.
Prefix A	Denotes *Advanced eXtensible Interface* (AXI) global and address channel signals.
Prefix B	Denotes AXI write response channel signals.
Prefix C	Denotes AXI low-power interface signals.
Prefix H	Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix n	Denotes active-LOW signals except in the case of AHB or *Advanced Peripheral Bus* (APB) reset signals. These are named **HRESETn** and **PRESETn** respectively.
Prefix P	Denotes APB signals.
Prefix R	Denotes AXI read channel signals.
Prefix W	Denotes AXI write channel signals.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of `0x3F`. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

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

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the ARM Limited Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the ETM10RV. Read the following manuals for additional information:

- *Embedded Trace Macrocell Specification* (ARM IHI 0014)
- *Embedded Trace Buffer Technical Reference Manual* (ARM DDI 0242)
- *ETM9 Technical Reference Manual* (ARM DDI 0157)
- *ARM1026EJ-S Technical Reference Manual* (ARM DDI 0244)
- *ETM10RV Implementation Guide* (ARM DII 0044)
- *Multi-ICE User Guide* (ARM DUI 0048).

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 ETM10RV and its documentation.

Feedback on the product

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

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

Feedback on this manual

If you have any comments on this manual, send email to `errata@arm.com` giving:

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

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the *ARM10 Embedded Trace Macrocell RealView* (ETM10RV) r0p0. It contains the following sections:

1.1 About the ETM10RV

The ETM10RV r0p0 provides instruction and data trace for the ARM1026EJ-S processor. The ETM10RV is an integral part of the ARM real-time debug solution, RealView.

The main feature enhancements of the ETM10RV over previous ETM versions include:

- ETMv3 protocol p-headers that compress information given by pipestats and embed the same information into a single packet stream. Because this feature is not compatible with previous versions of the ETM architecture, it cannot be referred to as an extension of ETMv2. The advantages of this feature are:
  - significant improvement in bandwidth efficiency, especially when data trace is disabled
  - more efficient trace storage when used with the new enhanced trace buffer
  - trace port speed can be decoupled from core clock speed without effecting the port width
  - trace can be stored as a raw bit stream.
- Java tracing.
- Replace FIFOFULL with a data trace suppression mechanism. If overflow is imminent, stop tracing data instead of stopping the core.
- Improvements to *CoProcessor Register Transfer* (CPRT) trace filtering.

The block diagram of the ETM10RV is shown in Figure 1-1 on page 1-3.

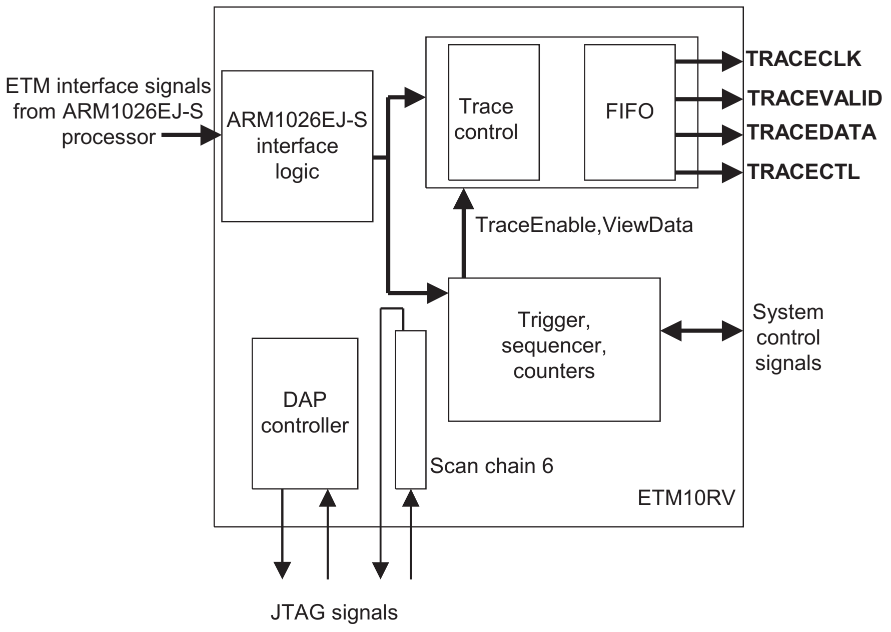


Figure 1-1 Block diagram of the ETM10RV

For information about the trace protocol, and about controlling tracing using triggering and filtering resources, see the *Embedded Trace Macrocell Specification*.

For information about ETM10RV input and output signals, see Appendix A *Signal Descriptions*.

1.2 Standard configuration

Unlike previous ETM versions, the ETM10RV is offered in just one standard configuration. The resources available are described in Table 1-1.

Table 1-1 ETM10RV configurations

Resource description	Configuration
Pairs of address comparators	4
Data comparators	2
Memory map decoders	0
Context ID comparators	1
Counters	2
Sequencer present	Yes
External inputs	4
External outputs	2
FIFOFULL present	No
FIFO depth	64
Port size[a]	4/8/16/24/32

a. Software-selectable using the ETM control register.

Chapter 2
Accessing the ETM10RV Registers

This chapter describes how to program the registers that control the trace and triggering facilities of the ETM10RV. It contains the following sections:

- *Debug interface* on page 2-2
- *ETM10RV registers* on page 2-4.

2.1 Debug interface

All registers in the ETM10RV are programmed through a Debug interface. The interface is an extension of the ARM *Debug Access Port* (DAP) controller, and is assigned scan chain 6.

The ETM10RV DAP is logically part of the ARM1026EJ-S processor to which it is connected. This means that Multi-ICE detects only one DAP in a single ARM10-ETM system.

The general structure of the ETM10RV debug registers is shown in Figure 2-1.

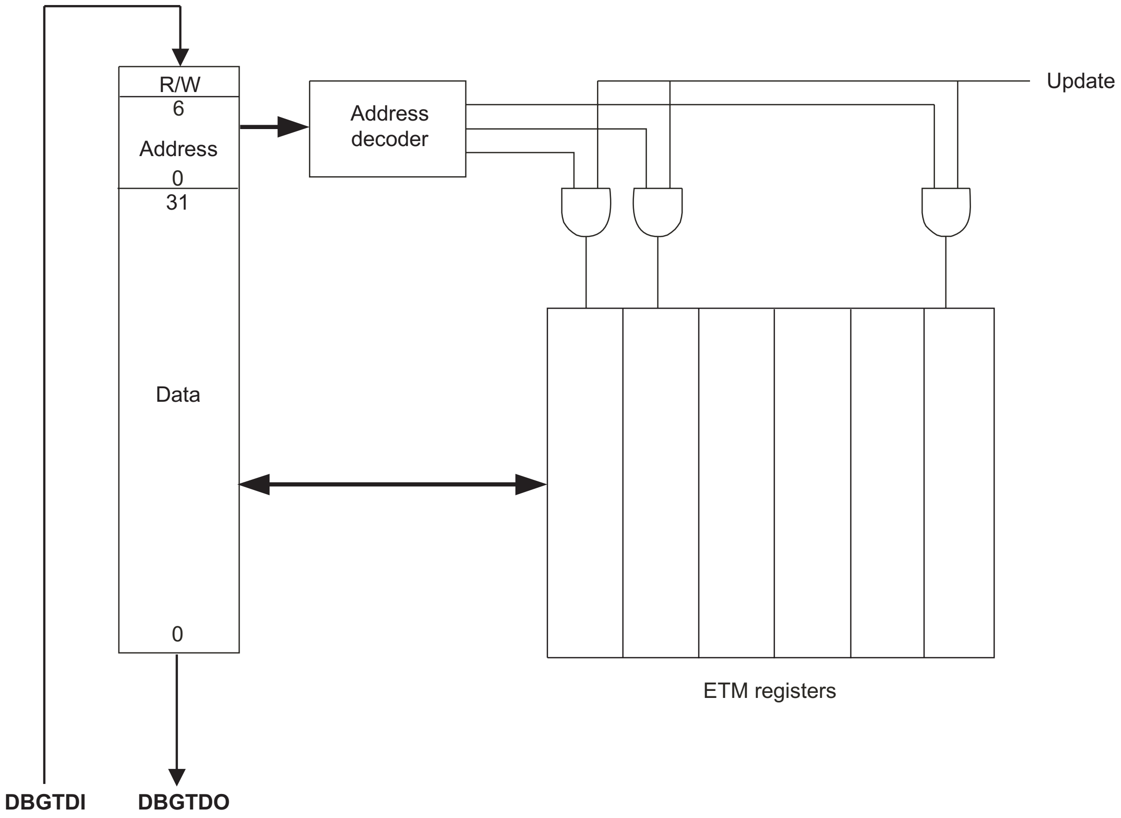


Figure 2-1 ETM10RV DAP structure

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 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 DAP controller enters the UPDATE-DR state.

2.2 ETM10RV registers

When modifying ETM10RV registers, you must set the programming bit in the ETM control register (`0x00`) before making any further modifications. When all register updates are complete, you must clear the programming bit. While the programming bit is set, all ETM operations are disabled.

You do not have to set the programming bit to read ETM registers. You can use the ETM programming bit in the ETM control register to disable all operations during programming. To do this, you must follow the procedure shown in Figure 2-2 on page 2-5.

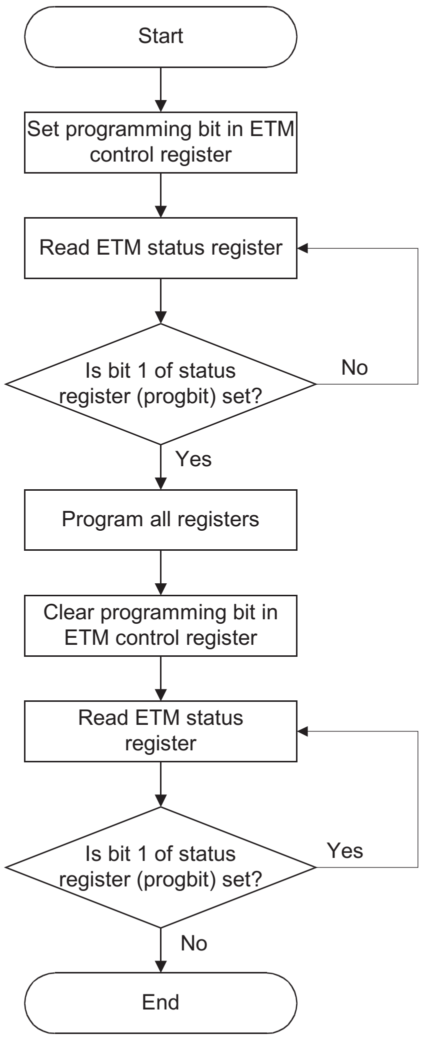

Start
Set programming bit in ETM control register
Read ETM status register
Is bit 1 of status register (progbit) set?
No
Yes
Program all registers
Clear programming bit in ETM control register
Read ETM status register
Is bit 1 of status register (progbit) set?
Yes
No
End

Figure 2-2 Programming ETM registers

Note

It is not necessary for the ETM to be in debug state while the registers are being programmed.

The ETM control register and all the other ETM registers are described in the *Embedded Trace Macrocell Specification*.

Chapter 3
Integrating the ETM10RV

This chapter describes how to integrate the ETM10RV with an ARM1026EJ-S processor. It contains the following sections:

3.1 About integrating the ETM10RV

The interface comprises 301 signals as follows:

253 inputs All input signals are registered immediately inside the ETM10RV.

48 outputs All outputs signals are driven directly from the output of a register, with the single exception of **DBGTDO**.

3.1.1 ETM10RV to ARM1026EJ-S connection guide

Table 3-1 shows how the ETM10RV must be connected to the ARM1026EJ-S processor.

Table 3-1 ETM10RV to ARM1026EJ-S signal connections

ETM10RV signal name	ARM1026EJ-S signal name
DBGRQ	**EDBGRQ**
DBGTDO	**SDOUTBS**
PWRDOWN	**ETMPWRDOWN**
DBGACK	**DBGACK**
ETMCORECTL[30:0]	**ETMCORECTL[30:0]**
ETMDATAVALID[1:0]	**ETMDATAVALID[1:0]**
ETMIA[31:0]	**ETMIA[31:0]**
ETMDA[31:0]	**ETMDA[31:0]**
ETMR15EX[31:0]	**ETMR15EX[31:0]**
ETMR15BP[31:0]	**ETMR15BP[31:0]**
ETMDATA[63:0]	**ETMDATA[63:0]**

The functional diagram of the ETM10RV is shown in Figure 3-1 on page 3-3.

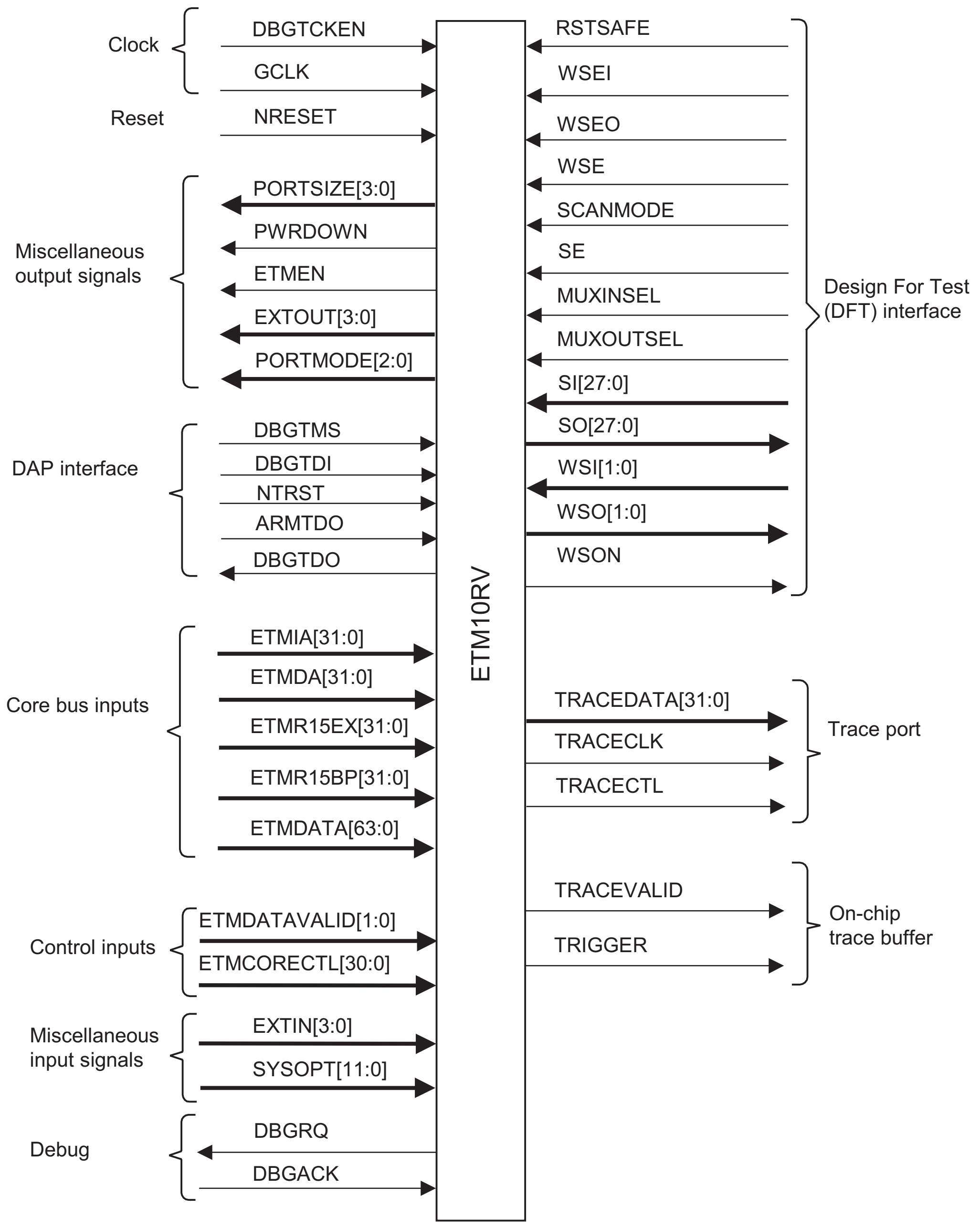


Figure 3-1 ETM10RV signals

3.2 ARM1026EJ-S trace interface

The ARM1026EJ-S trace interface is described in the following sections:

- *ETM10RV data path inputs*
- *ETM10RV control inputs* on page 3-5.

3.2.1 ETM10RV data path inputs

The ETM10RV data input buses are described in Table 3-2.

Table 3-2 ETM10RV data path input signals

Signal name	Description
ETMDATA[63:0]	Contains the data value for a Load, Store, MRC, or MCR instruction.
ETMDA[31:0]	Data address bus. Gives the address for every Load or Store transaction.
ETMIA[31:0]	Instruction address bus. Gives the address for every instruction fetch.
ETMR15BP[31:0]	Address for the branch phantom currently in the Execute stage of the processor pipeline.
ETMR15EX[31:0]	Address for the instruction currently in the Execute stage of the processor pipeline.

The ETMDATA bus

Four data buses are used for ETM data tracing:

- Load data bus
- Store data bus
- MCR data bus
- MRC data bus.

Each data bus is 64 bits wide. Only one data bus can contain valid data in any cycle, so all four buses are multiplexed within the ARM1026EJ-S processor to a single 64-bit data bus, **ETMDATA**. **ETMDATA** is registered within the ARM1026EJ-S processor before it is driven to the ETM. **ETMDATA** is valid in the Write (WR) stage of the ARM1026EJ-S pipeline.

Address buses

Four address buses are driven from the ARM1026EJ-S processor to ETM10RV:

ETMDA[31:0]	Valid in the Memory (ME) stage of the ARM1026EJ-S pipeline.
ETMIA[31:0]	Valid in the Decode (DE) stage of the ARM1026EJ-S pipeline.
ETMR15BP[31:0]	Valid in the Execute (EX) stage of the ARM1026EJ-S pipeline.
ETMR15EX[31:0]	Valid in the Execute (EX) stage of the ARM1026EJ-S pipeline.

As shown in Table 3-2 on page 3-4, **ETMDA[31:0]** is for data addresses and the other address buses are for instruction addresses.

3.2.2 ETM10RV control inputs

The ETM10RV control inputs are described in the following sections:

- *ETMDATAVALID[1:0]*
- *ETMCORECTL[30:0]*.

ETMDATAVALID[1:0]

This signal qualifies the data on the **ETMDATA[63:0]** bus as follows:

- **ETMDATAVALID[1]** qualifies **ETMDATA[63:32]**
- **ETMDATAVALID[0]** qualifies **ETMDATA[31:0]**.

ETMCORECTL[30:0]

The control signals from the ARM1026EJ-S processor comprise **ETMCORECTL[30:0]**. The control signals present on this bus are described in Table 3-3 on page 3-6.

——— **Note** ———

All the signals described in Table 3-3 on page 3-6 are valid in the Write (WR) stage of the ARM1026EJ-S pipeline, unless specified otherwise.

Table 3-3 Signals on the ETMCORECTL[30:0] bus

Signal name	Bit number	Description	Qualified by
ITBit	[0]	Asserted when ARM1026EJ-S processor is in Thumb state (valid in ME).	**InMREQ/ForcePF**
InMREQ	[1]	Current address on the IA bus is for a valid instruction fetch.	None
ForcePF	[2]	Current address on the IA bus is a target for an indirect branch.	None
InstValid	[5]	Asserted once per executed instruction. Not asserted for mispredicted branches, because these are fetched but not executed.	None
BpValid	[3]	Asserted once per executed branch phantom. When asserted, a branch phantom is present in the Execute stage of the processor pipeline.	None
CCFail	[6]	Current instruction failed its condition codes.	**InstValid**
BpCCFail	[4]	Branch phantom failed its condition codes.	**BpValid**
Tbit	[7]	Asserted when ARM1026EJ-S processor is in Thumb state (valid in ME).	**InstValid**
ETMBranch	[8]	Last instruction executed is an indirect branch.	Asserted before or coincident with **ForcePF**
R15HoldMe	[9]	Stall signal for the address given on R15EX.	None
UpdatesContextID	[10]	Current instruction is updating the CONTEXT ID.	**InstValid**
DnMREQ	[11]	Qualifies the Data Address bus, DA.	None
DMAS[1:0]	[13:12]	Load or store data size.	**DnMREQ**
ETMSwap	[14]	Indicates a 64-bit store to a big-endian memory device.	**DnMREQ**
DnRW	[15]	Data request read/write signal (LOW for read cycle).	**DnRW**
HUMACK	[16]	Valid load miss data is present on the data bus this cycle.	None
LSCMInit	[17]	Current instruction is an LSM instruction.	**InstValid** unless it is a Java instruction
LSCM	[18]	LSM is in progress in the Load/Store Unit.	**DnMREQ**

Table 3-3 Signals on the ETMCORECTL[30:0] bus (continued)

Signal name	Bit number	Description	Qualified by
MISSCNT[1:0]	[20:19]	The number of load misses that are outstanding.	None, transitions indicate new miss.
TrueException	[21]	Current instruction is an exception (interrupt, reset, or abort). Asserted on all exceptions.	**InstValid** for ARM & Thumb instructions. JInstEnd for Java instructions.
DAbort	[22]	Data request aborted.	**DnMREQ**
IgnoreMREQ	[23]	Current instruction is a preload and must not be traced.	**InstValid**
JBit	[24]	Asserted when ARM1026EJ-S processor is in Java mode.	**InstValid**
IJBit	[25]	Asserted when ARM1026EJ-S processor is in Java mode.	ForcePF
JInstEnd	[26]	Qualifies end of a Java instruction.	None
ETMRSVALID	[27]	Indicates that the target address for a stack operation is on the **ETMR15BP** bus.	None
Exception	[28]	Asserted to indicate cancelling exceptions.	InstValid for ARM and Thumb instructions. JInstEnd for Java instructions.
ExType[1:0]	[30:29]	Indicates the type of exception.	InstValid for ARM and Thumb instructions. JInstEnd for Java instructions.

3.3 System control signals

The ETM10RV outputs that are input to the ARM1026EJ-S are described in the following sections:

- *Debug*
- *Using the PWRDOWN signal* on page 3-9
- *The SYSOPT bus* on page 3-9.

3.3.1 Debug

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

It is recommended that you connect the **DBGRQ** output of the ETM10RV to the **EDBGRQ** input of the ARM1026EJ-S processor. If this input is already in use, when for example a **DBGRQ** input is present on the device, you can logically OR the **DBGRQ** signals together as shown in Figure 3-2.

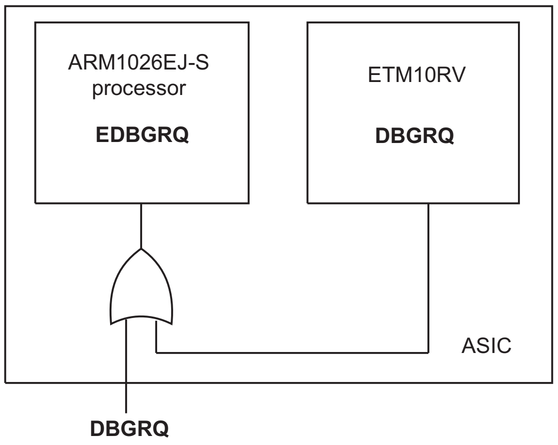


Figure 3-2 Combining DBGRQ inputs

Note

ARM1026EJ-S processors take at least one cycle to respond to **EDBGRQ**. This means that the ARM1026EJ-S 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.

The ARM1026EJ-S processor only recognizes **EDGBRQ** if the processor is currently in hardware debug mode.

3.3.2 Using the PWRDOWN signal

PWRDOWN is asserted when the ETM is not enabled. When a DAP reset (**NTRST**) occurs, **PWRDOWN** is forced HIGH until bit 0 of the ETM10RV control register has been deasserted (the ETM10RV registers are programmed by the ARM debug tools through JTAG). The ARM1026EJ-S processor uses the assertion of **PWRDOWN** to indicate that ETM10RV bus inputs must be held stable. Within the ETM, **PWRDOWN** assertion is also used to gate the clock for more power savings.

Note

For **PWRDOWN** to function correctly, the DAP reset signal must be asserted every time the system is powered on (see *NTRST* on page 3-11).

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

Context ID values

ETM10RV supports the tracing of dynamically loaded memory and overlay systems by maintaining the current context ID value in an internal register. To guarantee that context ID values are updated while the ETM is in powerdown state, the ETM10RV clock gating logic allows a one cycle clock pulse on context ID updates. This logic is part of the ETM10RV internal clock control logic, and is controlled by **ETMCORECTL[10]**, an input from the ARM1026EJ-S processor.

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.

3.3.3 The SYSOPT bus

System control signals are broadcast on the system options bus, **SYSOPT[11:0]**. You can use the **SYSOPT** bus 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 LOW or HIGH, depending on the features supported. The trace debug tools read the state of the **SYSOPT** bus using the Debug interface, and adapt the user options offered accordingly. The signals on the **SYSOPT** bus are described in Table 3-4 on page 3-10.

Table 3-4 SYSOPT bus settings

Bit number	Name	Description
[11]	Port mode supported	If HIGH, the currently selected port mode is supported internally or externally.
[10]	Port size supported	If HIGH, the currently selected port size is supported internally or externally for the currently selected port mode. Allows more complex port sizes to be supported.
[9]	Maximum port size (bit 3)	See bits [2:0].
[8]	FIFOFULL	Unchanged, typically reads as 0.
[7:5]	Old port mode bits	If reads as 0, indicates no trace port in ETMv2. Causes tools not supporting ETM10RV to ignore **SYSOPT**. Use bit 0 instead.
[4]	Full-rate clocking supported	If clear, indicates no support for this mode.
[3]	Half-rate clocking supported	If set, indicates support for this mode.
[2:0]	Maximum port size bits [2:0]	Gives the maximum size supported in any mode. Used in conjunction with bit 9. All sizes from here down to 4 bits must be supported. See bit 4 to determine if other sizes are supported.

3.4 Clocks and resets

The ETM10RV has two sets of clock and reset inputs. These are described in the following sections:

- *GCLK*
- *DBGTCKEN*
- *NRESET*
- *NTRST*.

3.4.1 GCLK

GCLK is the clock used by ETM10RV. **GCLK** is the same clock that is used by the ARM1026EJ-S processor. To save power, **GCLK** is gated internal to ETM10RV whenever the ETM10RV output **PWRDOWN** is asserted. The **PWRDOWN** signal is described in *Using the PWRDOWN signal* on page 3-9.

3.4.2 DBGTCKEN

DBGTCKEN is the test clock enable for the DAP interface. This signal must be GCLK gated with the system DBGTCKEN signal, but this logic must be external to the synthesizable macrocell.

Note

All the registers that were clocked by **TCK** in ETM10 are now enabled by **DBGTCKEN** and clocked by **GCLK** in ETM10RV.

3.4.3 NRESET

NRESET is the system power-on reset. **NRESET** is used only to reset the context ID shadow register in ETM10RV. **NRESET** is synchronized to **GCLK** for deassertion internal to ETM10RV to avoid timing problems. Assertion of **NRESET** is asynchronous.

3.4.4 NTRST

NTRST is the DAP controller reset. It is used within the ETM10RV as the main reset signal for **GCLK** domain logic. ETM10RV uses **NTRST** rather than the processor reset signal so that you can trace the processor through reset. Deassertion of the main **NTRST** input can be asynchronous to **GCLK**. **NTRST** is synchronized internally for **GCLK** domains.

——— **Note** ———

NTRST must be asserted for at least eight cycles when the system is initially powered on, otherwise the behavior of the ETM and of the whole system is Unpredictable.

3.5 ETM10RV ASIC interface signals

Table 3-5 shows the ETM10RV ASIC interface signals.

Table 3-5 ETM10RV ASIC interface signals

Signal name	Direction	Description
ETMEN	Output	Trace port enable. Must not be used to power down functionality required by the ETM10RV unless it only relates to the trace port.
PWRDOWN	Output	Indicates that the ETM is inactive and logic supporting it must be clock-gated to conserve power. This is asserted when the ETM is reset and remains asserted until it is first programmed. This is connected to the core, although it might be used by other functionality.
EXTIN[3:0]	Input	General-purpose external inputs to the ETM, controlling tracing and other resources.
EXTOUT[1:0]	Output	External outputs from the ETM. Can be used to trigger hardware inside the ASIC, or external equipment such as a logic analyzer.
PORTMODE[2:0]	Output	Indicates the frequency at which the **TRACECLK** requires to be run with reference to **GCLK**.
PORTSIZE[3:0]	Output	Indicates the currently selected port size in use on the **TRACEPKT** bus.

3.6 DAP interface wiring

Both the ARM1026EJ-S processor and the ETM10RV provide scan chain expansion inputs. These are:

- **SDOUTBS** on the ARM1026EJ-S processor
- **ARMTDO** on ETM10RV.

In each case this input is routed to **DBGTDO** when a non-ARM1026EJ-S or non-ETM10RV scan chain is selected. This enables the ARM1026EJ-S and ETM10RV DAP controllers to run in parallel with a single **DBGTDO** output.

ETM10RV uses the same DAP interface wiring as previous ETM versions. The **DBGTDO** pin is connected to the ARM1026EJ-S scan expansion input **SDOUTBS**. ETM10RV registers are accessed using scan chain 6 and are programmed in a manner identical to previous ETM implementations.

The recommended connectivity is shown in Figure 3-3. For information about the Debug interface, see *Debug interface* on page 2-2.

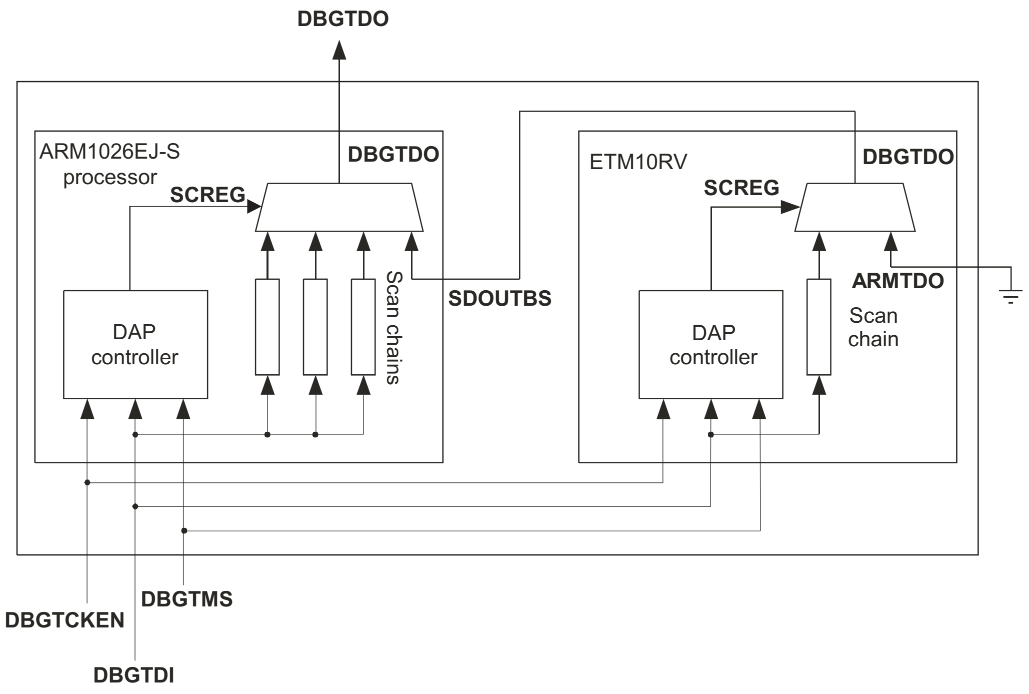


Figure 3-3 Recommended DAP interface structure

Note

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

If your ASIC includes a further scan chain controlled by the ARM1026EJ-S DAP controller, then the **DBGTDO** of this scan chain can be connected into the otherwise unused **ARMTDO** input on the ETM10RV. This is shown in Figure 3-4.

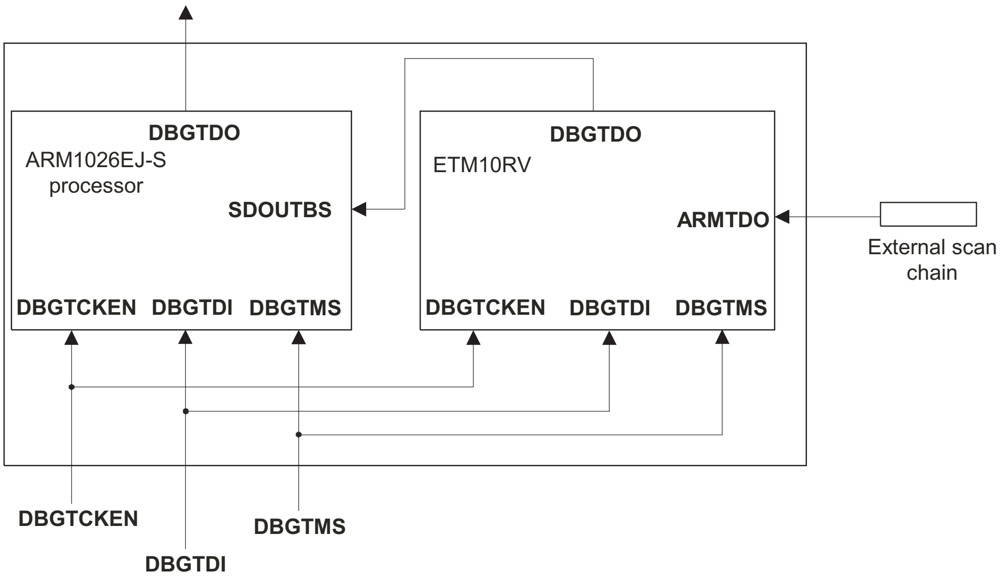


Figure 3-4 Using ETM10RV and ARM1026EJ-S with an external scan chain

3.6.1 Multiprocessor DAP structure

If you want your debug 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 DAP structure. The presence of an ETM10RV on any or all of the ARM processors does not affect this. The DAP structure for a dual-processor system is shown in Figure 3-5 on page 3-16.

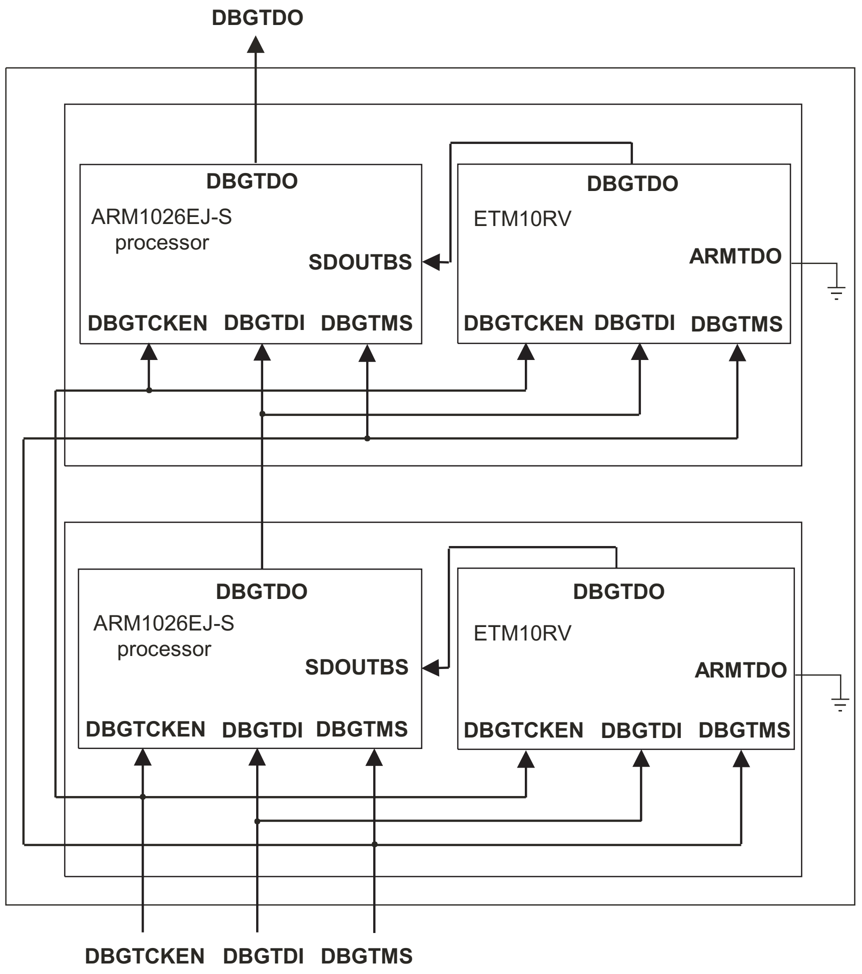


Figure 3-5 Multiprocessor DAP structure

Note

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

3.7 Trace port interfacing

Trace port interfacing is described in the following sections:

- *Trace port logic*
- *Single-processor tracing*
- *Dual-processor tracing* on page 3-18
- *Supported trace clocking modes* on page 3-20
- *PCB design guidelines* on page 3-21.

3.7.1 Trace port logic

Trace information from the ETM10RV is broadcast on the following signals:

- **TRACECLK**
- **TRACECTL**
- **TRACEDATA[31:0]**.

Three configuration signals are also provided:

- **ETMEN**
- **PORTSIZE[3:0]**
- **PORTMODE[2:0]**.

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

3.7.2 Single-processor tracing

Some chips might not dedicate 16 pins to the TRACEDATA bus. Under some circumstances, you might be able to reuse miscellaneous output signals from the chip as trace port pins. To allow for this, the ETM10RV has the following outputs:

- ETMEN
- PORTSIZE[3:0].

Figure 3-6 on page 3-18 shows one way in which the TRACEDATA pins can be shared with the ASIC pins.

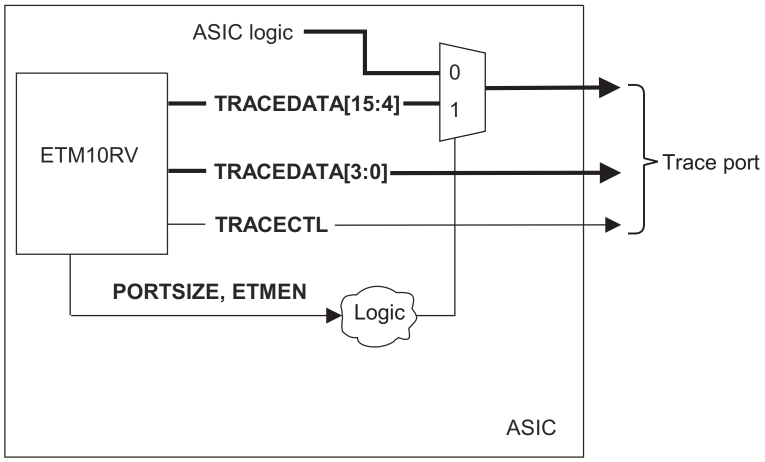


Figure 3-6 Reusing TRACEDATA pins

You can use the PORTSIZE and ETMEN signals to control on-chip logic to reset between the normal ASIC output signals and the ETM10RV 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 ETM10RV is disabled (ETMEN LOW) and a four-bit port is selected (PORTSIZE = 000). This ensures that normal operation of the ASIC is unaffected.

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

3.7.3 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 3-17, 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 20 pins on the ASIC and two clock pins, giving maximum flexibility between port sizes across all ETM architecture versions. See Embedded Trace Macrocell Specification for exact pinout of dual target connectors.

You can use a single 20-pin trace port to allow for configurations of dual-processor systems as shown in Table 3-6.

Table 3-6 Dual-processor configurations for a single 20-pin trace port

Processor 1	Processor 2
20 data pins	No trace
12 data pins	8 data pins
8 data pins	12 data pins
No trace	20 data pins

The following pseudo-HDL describes how an ETM10RV can share a trace port with another ETM:

```
//ETM10RV (ETMv3)TRACE_PINS_A[19:0] = {TRACEDATA_A[15:1]}, 3'b001, TRACECTL_A,
TRACEDATA_A[0]}
```

```
//Choose the appropriate line for the secondary ETM//ETMv3//TRACE_PINS_B[19:0] =
{TRACEDATA_B[15:1]}, 3'b001, TRACECTL_B,
TRACEDATA_B[0]}//ETMv2TRACE_PINS_B[19:0] = {TRACEPKT_B[15:0]},
PIPESTAT_B[3:0]}//ETMv1//TRACE_PINS_B[19:0] = {TRACEPKT_B[15:0]}, TRACESYNC_B,
PIPESTAT_B[2:0]}
```

```
if (PORTSIZE_A = 16)
TRACE_PINS[19:0] = TRACE_PINS_A[19:0]
else if (PORTSIZE_A = 8)
TRACE_PINS[19:0] = {TRACE_PINS_B[7:0], TRACE_PINS_A[11:0]}
else if (PORTSIZE_A = 4)
TRACE_PINS[19:0] = {TRACE_PINS_A[7:0], TRACE_PINS_B[11:0]}
else
        TRACE_PINS[19:0] = {TRACE_PINS_B[19:0]
end if
```

Note

If the secondary core is an ETMv1 or ETMv2 device and you want to use it in demultiplexed mode, you must use two separate target system connectors. See *Supported trace clocking modes* on page 3-20 for more details.

The *Embedded Trace Macrocell Specification* describes 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.

3.7.4 Supported trace clocking modes

The port mode is selected by the trace software. No additional support is required by the ASIC. Because the ETMv3 trace protocol enables the trace port clock to be decoupled from the core clock, no additional support is required by the TPA to use these modes.

No support is provided for additional modes to be supported externally. The currently selected port mode is indicated on **PORTMODE[2:0]**.

ETM10RV supports the following clocking modes as defined in the *Embedded Trace Macrocell Specification*:

- 1:2
- 1:4.

The above clocking modes enable the frequency of the trace port signals to be reduced to an acceptable speed. Data is captured by the TPA on both edges of **TRACECLK**.

Figure 3-7 shows the timing for 1:2 mode.

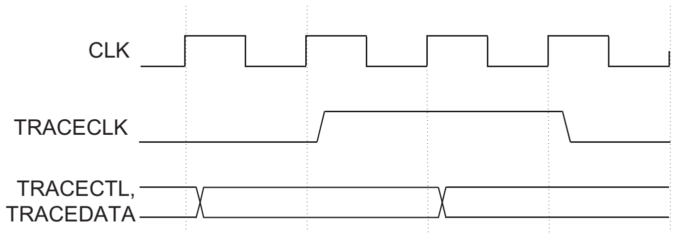


Figure 3-7 1:2 clocking mode timing

Figure 3-8 shows the timing for 1:4 mode.

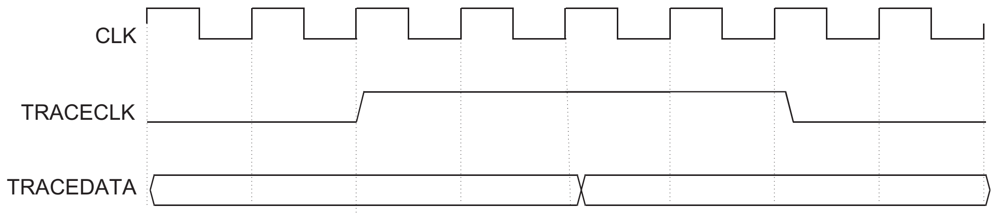


Figure 3-8 1:4 clocking mode timing

Note

To trace data using the ETM10RV in 1:1 mode, you must use the on-chip trace buffer. For more information on using the trace buffer, see the *Embedded Trace Buffer Technical Reference Manual*.

3.7.5 PCB design guidelines

See Chapter 2 *Accessing the ETM10RV Registers* for information about output pad selection and PCB design, including:

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

Chapter 4
Design for Test

This chapter describes the *Design for Test* (DFT) features of the ETM10RV. It contains the following sections:

4.1 About DFT

For the purposes of scan testing, the ETM10RV comprises a functional core surrounded by a test wrapper. The wrapper provides a single serial scan ring around the entire periphery, and enables:

- test control and observation of the functional core from the ports
- control and observation of the external logic surrounding the functional core.

The test wrapper enables a tester to apply vectors, or control stimulus, that can achieve a high quality measurement with a minimal amount of external pin control. This is extremely important if the design unit is to be embedded or buried within other design units or chip logic. See *ETM10RV test wrapper* on page 4-4 for more details.

4.1.1 DFT modes

In addition to the normal ETM10RV functional mode, there are two DFT modes, as follows:

Internal test mode	In this mode, you have control over the inputs to the functional core during test, and you can observe the core outputs (see *Internal test mode* on page 4-11).
External test mode	In this mode, you can observe the logic in the wrapper cell itself, while signals are propagated between the peripheral logic to the functional core (see *External test mode* on page 4-12).

Note

The delay to the clock port of the ARM1026EJ-S processor and the ETM10RV must match otherwise there can be hold time issues during ETM10RV internal test mode.

4.1.2 Clock gating

A clock gate must be added external to the ETM10RV macrocell to enable clock gating of the **CLK** signal. This is for data hold during I_{ddq} testing or to diminish the overall power consumption during testing of logic other than the ETM10RV macrocell.

4.1.3 Asynchronous signals

Asynchronous signals must be directly controlled by the *Automatic Test Pattern Generator* (ATPG) tool. The asynchronous reset signals on the ETM10RV are directly controlled during testing by the **NRESET** and **NTRST** signals.

Figure 4-1 on page 4-3 shows an example of the reset test logic.

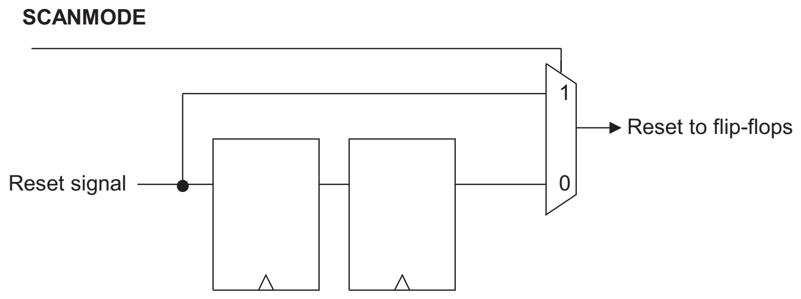


Figure 4-1 Reset test logic example

4.2 ETM10RV test wrapper

You use the test wrapper for control and observation of the ports during testing of the ETM10RV core and the logic external to the core.

The wrapper scan chain comprises both shared and dedicated test wrapper cells. There is a wrapper cell connected to every functional input and output port that is not connected to the ARM1026EJ-S processor, with the exception of the clock ports.

The test wrapper isolates all ports on the ETM10RV, except ports that connect to the ARM1026EJ-S processor. These ports use a portion of the wrapper chain on the ARM1026EJ-S processor. If the ARM methodology is used, then the ETM10RV macrocell test vectors are created using both the ETM10RV and a portion of the ARM1026EJ-S core wrapper. Figure 4-2 shows the ETM10RV wrapper interface.

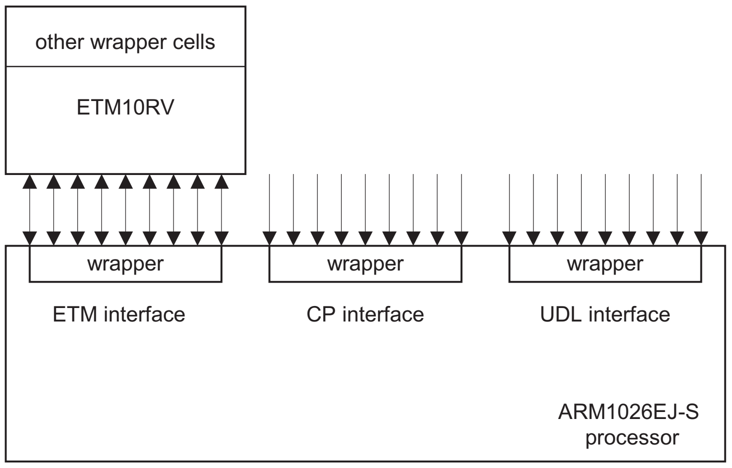


Figure 4-2 ETM10RV wrapper interface

There are four types of wrapper cell, described in the following sections:

- *Dedicated input wrapper cells* on page 4-5
- *Dedicated output wrapper cells* on page 4-5
- *Shared wrapper cells* on page 4-6
- *Scan test reset control* on page 4-7.

4.2.1 Dedicated input wrapper cells

The input wrapper cells are controlled by **MUXINSEL**. When **MUXINSEL** is HIGH, all the input wrapper cells are in inward-facing mode to enable you to control the ETM10RV core inputs during test with the wrapper cell. When **MUXINSEL** is LOW, the wrapper input cells observe signal propagation from the peripheral logic to the functional ports of the ETM10RV. This is the functional mode.

Figure 4-3 shows a dedicated input wrapper cell.

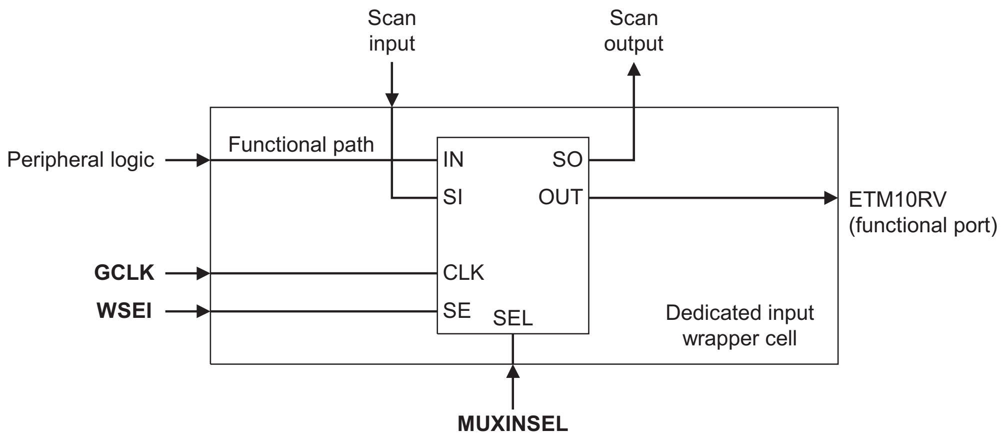


Figure 4-3 Dedicated input wrapper cell

4.2.2 Dedicated output wrapper cells

The dedicated output wrapper cells are controlled by **MUXOUTSEL**. When **MUXOUTSEL** is HIGH, the output wrapper cells control the signals to the peripheral logic attached to the core. When **MUXOUTSEL** is LOW, the cells observe the logic in the functional core. This is the functional mode.

Figure 4-4 on page 4-6 shows a dedicated output wrapper cell.

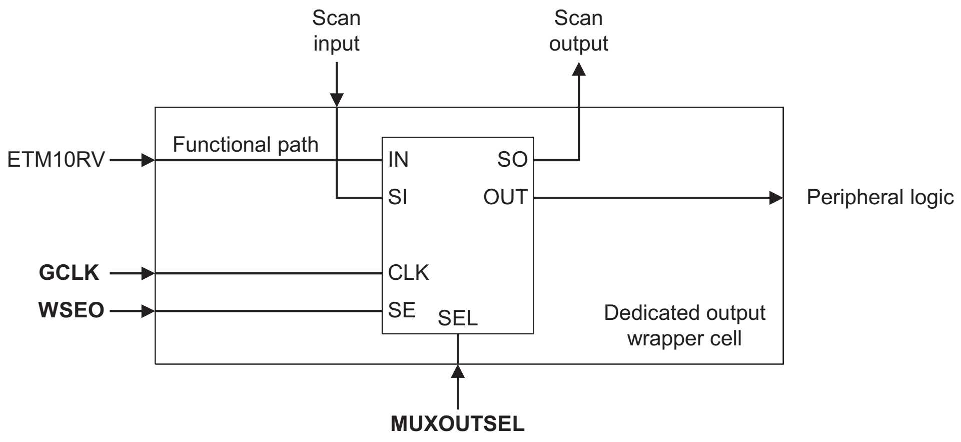


Figure 4-4 Dedicated output wrapper cell

4.2.3 Shared wrapper cells

Shared wrapper cells are functional flip-flops that are also used as wrapper cells. This can only occur on registered inputs and outputs.

Figure 4-5 on page 4-7 shows a shared input wrapper cell and a shared output wrapper cell.

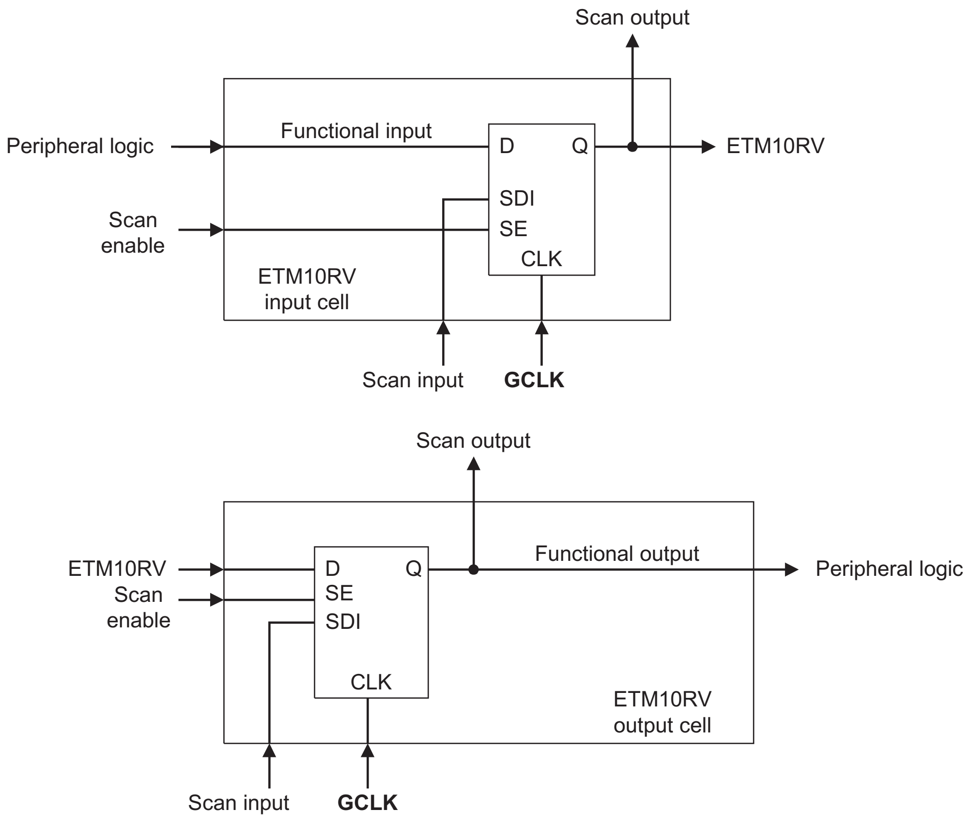


Figure 4-5 Shared input and output wrapper cells

4.2.4 Scan test reset control

Special scan test control is required for all asynchronous reset inputs. During external test mode, the **RSTSAFE** signal can enable the reset of the core to reduce power and to keep the core safe. In addition, both asynchronous resets **NRESET** and **NTRST** must be directly controllable during scan mode.

Figure 4-6 on page 4-8 shows the logic to control each reset on the ETM10RV macrocell.

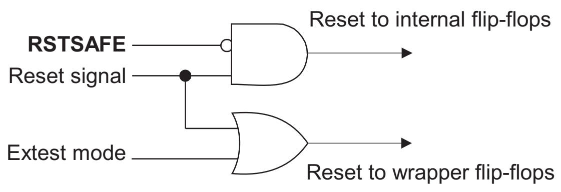


Figure 4-6 Active LOW reset example

4.3 Test modes and ports

Some of the ETM10RV test signals are static and some are dynamic. In the case of the ETM10RV scan patterns, a dynamic signal must propagate from the pin of the chip to the first flip-flop (the head flip-flop) of a scan chain in the core, within a cycle of the test pattern.

The ETM10RV signals during internal test mode are described in Table 4-1.

Table 4-1 ETM10RV test signals in internal mode

Signal name	Direction	Type	Description
SCANMODE	Input	Static	Prevents the asynchronous reset from being controlled from the synchronizer.
SE	Input	Dynamic	Scan enable for all internal clock domains. HIGH = shift mode. LOW = functional mode.
MUXINSEL	Input	Static	Configures the dedicated input wrapper cells for functional or test mode.
MUXOUTSEL	Input	Static	Configures the dedicated output wrapper cells for functional or test mode.
SI	Input	Dynamic	Scan input ports.
RSTSAFE	Input	Static	Enables the reset to the core, except for the wrapper.
WSEI	Input	Dynamic/Static	Scan enable for all input test cells in the wrapper. HIGH = shift mode. LOW = functional mode.
WSEO	Input	Dynamic	Scan enable for all output test cells in the wrapper. HIGH = shift mode. LOW = functional mode.
WSI	Input	Dynamic	Input ports for the wrapper scan chains.
SO	Output	Dynamic	Scan output ports.
WSO	Output	Dynamic	Output ports for the wrapper scan chains.
WSON	Output	Dynamic	Wrapper output port that changes after the falling edge of the clock.

4.3.1 Selecting a test mode

You use the wrapper control signals to select the wrapper mode, as shown in Table 4-2.

Table 4-2 Test mode selection

Mode	ETM10RV MUXINSEL	ETM10RV MUXOUTSEL	ARM1026EJ-S MUXINSEL	ARM1026EJ-S MUXOUTSEL
Internal ETM test mode	HIGH	LOW	LOW	HIGH
External ETM test mode	LOW	HIGH	HIGH	LOW
Functional mode	LOW	LOW	LOW	LOW

4.3.2 Wrapper scan chains

The wrapper scan chains can be concatenated as required by wiring the signal **WSO** of one scan chain to the signal **WSI** of another scan chain. The lengths of the scan chains are shown in Table 4-3.

Table 4-3 Wrapper scan chains

Scan chain	Chain number	Scan in ports	Scan out ports	Number of flip-flops in chain
ETM output	0	**WSI[0]**	**WSO[0]**	47
ETM input	1	**WSI[1]**	**WSO[1]** or **WSON**	22
ARM1026EJ-S output	5	**WSI[5]**	**WSO[5]**	225
ARM1026EJ-S input	2	**WSI[2]**	**WSO[2]**	1

For a single wrapper scan chain, the recommended concatenation is 5, 2, 0, 1.

Scan enable inputs

The wrapper contains two scan enable signals:

- the wrapper scan enable input, **WSEI**
- the wrapper scan enable output, **WSEO**.

The signal **WSEI** connects solely to the wrapper cells adjacent to the functional inputs and the signal **WSEO** connects to the wrapper cells adjacent to the functional outputs. You can, however, tie these two signals as one wrapper scan enable signal.

Wrapper outputs

The ETM10RV input wrapper scan chain has two wrapper outputs as shown in Figure 4-7. The last cell in the serial chain is a lock-up latch, **WSON**, that enables the wrapper chain to be more easily connected to scan chains in other clock domains, if required. To use this in a single wrapper chain mode, the ETM10RV wrapper chain must be last in the concatenation of wrapper chains.

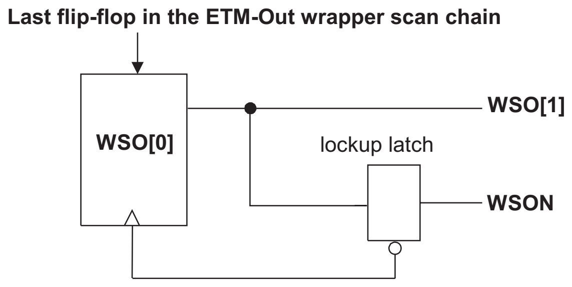


Figure 4-7 Wrapper falling edge logic

4.3.3 Internal test mode

The ETM10RV test signals connections for internal test mode are shown in Table 4-4. You can create a test control module to control the states of these signals.

Table 4-4 Test signals in internal test mode

Signal	Connection
SCANMODE	1
SE	Connect to an external pin
WSEO	Connect to an external pin
WSEI	Connect to an external pin or hold high
MUXINSEL	1
MUXOUTSEL	0

Table 4-4 Test signals in internal test mode (continued)

Signal	Connection
RSTSAFE	0
SI	Connect to external pins
SO	Connect to external pins
WSO	Connect to an external pin or another scan chain
WSON	Connect to an external pin, another scan chain, or does not have to be used
WSI	Connect to an external pin or a scanout pin

4.3.4 External test mode

The ETM10RV test signals connections for external test mode are shown in Table 4-5.

Table 4-5 Test signals in external test mode

Signal	Connection
SCANMODE	1
SE	1 (recommended)
WSEI	Connect to a pin
WSEO	Connect to a pin or hold high
MUXINSEL	0
MUXOUTSEL	1
RSTSAFE	1 (recommended except for I_{ddq} setup)
SI	0 (recommended)
SO	Gated 0 (recommended)

Table 4-5 Test signals in external test mode (continued)

Signal	Connection
WSO	Connect to a pin or another scan chain
WSON	Connect to a pin, another scan chain, or does not have to be used
WSI	Connect to a pin or a scanout pin

4.3.5 Functional mode

The ETM10RV test signals connections for functional mode are shown in Table 4-6.

Table 4-6 Test signals in functional mode

Signal	Connection
SCANMODE	0
SE	0
WSEI	0
WSEO	0
MUXINSEL	0
MUXOUTSEL	0
WSI	0 (recommended)
WSO	Gated 0 (recommended)
WSON	Gated 0 (recommended)
RSTSAFE	0
SI	0 (recommended)
SO	Gated 0 (recommended)

4.4 IDDQ

The ETM10RV macrocell is I_{ddq} capable. There are three basic methods to set up the measurement on a *System on Chip* (SoC):

- *Set up all logic on one or more Vdd planes simultaneously*
- *Partition the logic and set up each logic section individually*
- *Put each core on a separate Vdd plane and set up the logic for one Vdd* on page 4-15.

4.4.1 Set up all logic on one or more V_{dd} planes simultaneously

This method sets up all of the logic on one or more V_{dd} planes simultaneously. This method also requires a test pattern that inputs data into all of the scan chains during a single shift in pattern. This enables the test to run the pattern, then take the measurement. This is the recommended method if there is no instantaneous power drop-off issue when all of the flip-flops toggle together.

4.4.2 Partition the logic and set up each logic section individually

This method partitions the logic and set up each logic section individually. This method requires clock gating capability to each logic section so that when it is set up, the clock action during other testing does not change the state of the logic in the section that has already been set up. On the ETM10RV macrocell, the wrapper uses the same clock as the rest of the design. After the ETM10RV is set up for I_{ddq}, the wrapper cannot be used for external test set up. The recommended flow for this method is as follows:

1. Set up external logic using the ETM10RV macrocell wrapper.
2. Gate the clock to the external logic off.
3. Set up the ETM10RV macrocell.
4. Gate the clock to the ETM10RV macrocell off.
5. Set up any other logic partitions gating the clock off after the I_{ddq} vector has been input.

All logic belonging to a single voltage supply must be set up before taking a measurement of that supply. The ETM10RV macrocell must be set up for I_{ddq} before the ARM1026EJ-S processor because it uses the ARM1026EJ-S wrapper chain during its internal test mode. If the set up is not done in the correct order, the ARM1026EJ-S state can be corrupted while sensitizing the ETM10RV macrocell.

4.4.3 Put each core on a separate V_{dd} plane and set up the logic for one V_{dd}

This method puts each core on a separate V_{dd} plane and sets the logic for one V_{dd} to make a measurement. Repeat this step for other V_{dd}s on the chip. This can be costly in test time because it requires more I_{ddq} measurements. This might not be practical for V_{dd} pin utilization.

Chapter 5
Implementation-defined Behavior

This chapter contains implementation-specific information relating to the ETM10RV. It contains the following sections:

5.1 ETM architecture version

ETM10RV r0p0 implements version 3.0 of the ETM architecture (ETMv3.0).

5.1.1 ETM10RV ID register

Table 5-1 shows the value of the fields when reading the ETM10RV ID register (`0x79`).

Table 5-1 ETM10RV ID register fields

Bit numbers	Value	Meaning
[31:24]	`0x41`	Implementor = A (ARM Limited)
[23:16]	0000	Reserved
[15:12]	0010	ARM core = ARM10 family
[11:8]	0010	Major ETM architecture version number = 3
[7:4]	0000	Minor ETM architecture version number = 0
[3:1]	0000	Implementation revision = 0

5.2 Precise TraceEnable events

The *Embedded Trace Macrocell Specification* states that **TraceEnable** is imprecise under certain conditions, with some implementation-defined exceptions. Selection of the following resources by the enabling event does not cause **TraceEnable** to be imprecise, provided that those resources are themselves precise:

- single address comparators
- address range comparators
- context ID comparators.

5.3 Parallel instruction execution

The ARM1026EJ-S processor supports branch folding, where correctly predicted branches are executed in parallel with the following instruction. The ETM is therefore capable of tracing two instructions per cycle, although only the second instruction can have data associated with it.

While the trace start/stop block is calculated for each instruction as required, the ETM is not capable of tracing one instruction without the other. In particular, if a folded branch is traced, the instruction it is paired with is also traced, along with any data associated with it if **ViewData** is active.

5.4 Comparator behavior

All address comparators have a sticky bit that is observed when the comparator is used as shown in Table 5-2.

Table 5-2 Conditions for observing address comparator sticky bits

Comparator usage	Single address comparator	Address range comparator
Selected as an event	Sticky bit ignored	Sticky bit observed
Selected by **TraceEnable**	Sticky bit ignored	Sticky bit ignored
Selected by start/stop block	Sticky bit ignored	Sticky bit ignored
Selected by **ViewData**	Sticky bit observed	Sticky bit observed

Where it is observed, the sticky bit behaves as follows:

Sticky bit set	When the comparator matches the instruction address of a data instruction.
Sticky bit cleared	When the data instruction completes.

The sticky bit is never set for address comparators configured for data addresses.

Where observed, the comparator continues to match while the sticky bit is set.

Single address comparators cannot observe the sticky bit when selected as an event, because they are defined to be active for only one cycle for the benefit of the counters and sequencer. If an event of the kind `instruction address = X AND data address = Y` is required, the instruction address comparator must be an address range comparator to guarantee that it still matches when the data address comparator matches.

ViewData always observes the sticky bits so that if an instruction address comparator is selected, all the data corresponding to that instruction matches. In normal circumstances, you must not use single instruction address comparators as the enabling event, because they do not observe the sticky bit.

There is no need for **TraceEnable** or the start/stop block to observe the sticky bit, because when a data instruction is traced, tracing cannot be disabled until all data transfers corresponding to that instruction have occurred.

Note

Using data values to create an event, such as a sequencer transition, might result in out-of-order events occurring because the load data might be returned out of order. If you are concerned that the ARM1026EJ-S nonblocking cache might affect programmed events, you can disable it in the core by writing to bit 21 of the CP15 Configuration Register (R1). See the *ARM1026EJ-S Technical Reference Manual* for more information.

5.5 Supported port modes and sizes on ETM10RV

The ETM10RV supports the following port modes:

- Dynamic (for on-chip capture)
- 1:2
- 1:4.

If any other port mode is selected, Dynamic mode is used.

The ETM10RV supports the following port sizes:

- 4-bit
- 8-bit
- 16-bit
- 24-bit
- 32-bit.

Selection of any other port size results in Unpredictable behavior.

5.6 Exact match bit

The exact match bit is bit 7 of the address access type register. You can use the exact match bit to control comparator behavior. The functionality of the exact match bit varies depending on the following:

- Access type
- Requirements for the handling of interrupts and Prefetch Aborts
- Requirements for the handling of Data Aborts.

5.6.1 Trace filtering signals and aborts

In an environment that causes memory aborts, such as a demand-paged virtual memory system, you must carefully manage the occurrence of aborts. For example, when counting accesses or generating external outputs, usually programmers do not want to transfer that abort. The ETM10RV must therefore take into account the number of aborts when generating the outputs of address comparators.

5.6.2 Instruction address comparisons

For Execute-stage instruction address comparisons, the behavior of the exact match bit depends on whether the instruction is canceled due to a *Fast Interrupt Request* (FIQ), *Interrupt Request* (IRQ), Prefetch Abort, or reset exception. This is shown in Table 5-3.

Table 5-3 Exact match bit settings for instruction address comparisons

Exact match bit setting	Instruction canceled	Instruction not canceled
Exact match bit = 0	Comparator matches	Comparator matches
Exact match bit = 1	Comparator does not match	Comparator matches

For Fetch-stage instruction address comparisons, the exact match bit is ignored. Canceled instructions can cause the comparator to match.

Both Execute and Fetch-stage instruction address comparisons ignore the occurrence of Data Aborts. An instruction that causes a Data Abort always causes the comparator to match.

Note

When using address comparators to form a range, if Java addresses are used and the exact match bit is set, the enabling of the range comparator at the include/exclude boundary is Unpredictable. Within the boudary, the behavior is Predictable.

5.6.3 Data access comparisons

The behavior of the exact match bit during data access comparisons is summarized in Table 5-4.

Table 5-4 Exact match bit settings for data accesses

Data comparator present?	Exact match bit setting	Data returned immediately	Load miss	Data Abort
Yes	0	Comparator matches if data matches	Comparator matches	Comparator matches
Yes	1	Comparator matches if data matches	Comparator waits	Comparator does not match
No	0	Comparator matches	Comparator matches	Comparator matches
No	1	Comparator matches	Comparator matches	Comparator does not match

Exact match bit set to 0 (default setting)

When a load miss or Data Abort occurs, the comparator matches immediately. This means that tracing of load misses or Data Aborts is based on the data address only because the data value is assumed to be invalid. This is useful when the comparator is used for trace filtering provided that extraneous tracing is not considered to be a problem.

Exact match bit set to 1

When a load miss occurs, the comparator waits for data to be returned, then matches if the data matches.

Waiting for the data compare to occur is useful when data values are used by derived resources to create triggers and other events. Waiting for the data compare to occur causes the load miss to be missed if the comparator is used directly as an include region by **TraceEnable** or **ViewData**. Because the load miss data is traced only if the occurrence of the load miss is traced, this results in data not being traced even if it later matches.

When a Data Abort occurs, the comparator does not output a match regardless of whether or not a data comparison is requested. This is often the preferred behavior when a comparator is meant to match just once because aborted accesses are usually attempted again when the aborting condition is resolved.

When counting the execution of load or store instructions, the occurrence of Data Aborts and subsequent retrying of instructions cause the instruction count to be larger than expected.

Note

Using data values to create an event such as a sequencer transition, might result in out-of-order events occurring because the load data can be returned out of order. If you are concerned that the non-blocking cache might affect programmed events, you can disable it in the core.

5.7 Synchronization behavior during data suppression

During data suppression, all instruction synchronization is delayed:

- I-sync is not delayed during data suppression. If I-sync is not output during data suppression, this causes I-sync to be delayed until the next I-sync is scheduled. This causes the gap between two I-syncs to be twice as long as normal, as a result, an overflow is forced.
- A-sync is delayed until data suppression ends. If this causes A-sync to be delayed until after the next A-sync is scheduled, then an overflow is forced.
- D-sync is suppressed with other data trace. D-sync is always output with the resumption of data tracing following data suppression.

5.8 Java data instructions

The following Java instructions must be treated as data instructions for the purpose of decompression:

Table 5-5 Java data instructions

Instruction name	Instruction value	Maximum data size	Data traced
`iload`	`0x15`	Word	Transferred data
`lload`	`0x16`	Doubleword	Transferred data
`fload`	`0x17`	Word	Transferred data
`dload`	`0x18`	Doubleword	Transferred data
`aload`	`0x19`	Word	Transferred data
`iload_1`	`0x1B`	Word	Transferred data
`iload_2`	`0x1C`	Word	Transferred data
`iload_3`	`0x1D`	Word	Transferred data
`lload_0`	`0x1E`	Doubleword	Transferred data
`lload_1`	`0x1F`	Doubleword	Transferred data
`lload_2`	`0x20`	Doubleword	Transferred data
`lload_3`	`0x21`	Doubleword	Transferred data
`fload_0`	`0x22`	Word	Transferred data
`fload_1`	`0x23`	Word	Transferred data
`fload_2`	`0x24`	Word	Transferred data
`fload_3`	`0x25`	Word	Transferred data
`dload_0`	`0x26`	Doubleword	Transferred data
`dload_1`	`0x27`	Doubleword	Transferred data
`dload_2`	`0x28`	Doubleword	Transferred data
`dload_3`	`0x29`	Doubleword	Transferred data
`aload_1`	`0x2B`	Word	Transferred data
`aload_2`	`0x2C`	Word	Transferred data

Table 5-5 Java data instructions (continued)

Instruction name	Instruction value	Maximum data size	Data traced
`aload_3`	`0x2D`	Word	Transferred data
`iaload`	`0x2E`	Word	Transferred data
`laload`	`0x2F`	Doubleword	Transferred data
`faload`	`0x30`	Word	Transferred data
`daload`	`0x31`	Doubleword	Transferred data
`aaload`	`0x32`	Word	Transferred data
`baload`	`0x33`	Byte	Transferred data
`caload`	`0x34`	Halfword	Transferred data
`saload`	`0x35`	Halfword	Transferred data
`istore`	`0x36`	Word	Transferred data
`lstore`	`0x37`	Doubleword	Transferred data
`fstore`	`0x38`	Word	Transferred data
`dstore`	`0x39`	Doubleword	Transferred data
`astore`	`0x3A`	Word	Transferred data
`istore_0`	`0x3B`	Word	Transferred data
`istore_1`	`0x3C`	Word	Transferred data
`istore_2`	`0x3D`	Word	Transferred data
`istore_3`	`0x3E`	Word	Transferred data
`lstore_0`	`0x3F`	Doubleword	Transferred data
`fstore_0`	`0x40`	Word	Transferred data
`fstore_1`	`0x41`	Word	Transferred data
`fstore_2`	`0x42`	Word	Transferred data
`fstore_3`	`0x43`	Word	Transferred data
`dstore_0`	`0x44`	Doubleword	Transferred data

Table 5-5 Java data instructions (continued)

Instruction name	Instruction value	Maximum data size	Data traced
`dstore_1`	`0x45`	Doubleword	Transferred data
`dstore_2`	`0x46`	Doubleword	Transferred data
`dstore_3`	`0x47`	Doubleword	Transferred data
`astore_0`	`0x48`	Word	Transferred data
`astore_1`	`0x49`	Word	Transferred data
`astore_2`	`0x4A`	Word	Transferred data
`astore_3`	`0x4B`	Word	Transferred data
`iastore`	`0x4C`	Word	Transferred data
`lastore`	`0x4D`	Doubleword	Transferred data
`fastore`	`0x4E`	Word	Transferred data
`dastore`	`0x4F`	Doubleword	Transferred data
`aastore`	`0x50`	Word	Transferred data [a]
`bastore`	`0x51`	Byte	Transferred data
`castore`	`0x52`	Halfword	Transferred data
`sastore`	`0x53`	Halfword	Transferred data
`iinc`	`0x84`	Word	Result of increment
`ret`	`0xA9`	Word	PC load
`getfield_a`	-	Word	Transferred data
`getfield2_a`	-	Doubleword	Transferred data
`putfield_a`	-	Word	Transferred data
`putfield2_a`	-	Doubleword	Transferred data
`ldc_a`	-	Word	Transferred data
`ldc_w_a`	-	Word	Transferred data
`ldc2_w_a`	-	Doubleword	Transferred data

Table 5-5 Java data instructions (continued)

Instruction name	Instruction value	Maximum data size	Data traced
`getstatic_a`	-	Word	Transferred data [b]
`getstatic_a`	-	Doubleword	Base address load, transferred data [c]
`putstatic_a`	-	Word	Transferred data [d]
`putstatic_a`	-	Doubleword	Base address load, transferred data [e]

a. This instruction is implemented in software. It is recommended that this instruction is treated as a data instruction by the decompressor.
b. Static Pointer (SP) = 1
c. Static Pointer (SP) = 0
d. Static Pointer (SP) = 1
e. Static Pointer (SP) = 0

Note

`iload_0` and `aload_0` are not data instructions.

5.9 Configuration code register

The configuration code register (`0x01`) has the value `0x85294024`. Table 5-6 shows the fields in this register.

Table 5-6 Configuration code register

Bit numbers	Value	Meaning
[31]	1	Architecture version in ID register
[30:28]	000	Reserved
[27]	0	Reserved
[26]	1	Start/Stop block present
[25:24]	01	1 Context ID comparator present
[23]	0	FIFOFULL logic not present
[22:20]	010	2 external outputs present
[19:17]	100	4 external inputs present
[16]	1	Sequencer present
[15:13]	010	2 counters present
[12:8]	00000	0 *Memory Map Decode* (MMD) inputs present
[7:4]	0010	2 data comparators present
[3:0]	0100	4 pairs of address comparators present

5.10 Context ID tracing

The ETM detects the `MCR` instruction that changes the context ID and traces the appropriate number of bytes as a context ID packet instead of a normal data packet. As a result, if context ID tracing is enabled, an `MCR` instruction that changes the context ID does not have its data traced separately.

Chapter 6
Tracing Dynamically-loaded Images

This chapter describes software issues relating to the ETM10RV. It contains the following sections:

6.1 About tracing dynamically-loaded code

When a debugger is debugging a system, it communicates mainly in terms of accesses to addresses in memory or virtual memory. It must translate between these addresses and the locations in the code images loaded on the system. This enables the debugger to present to the user a symbolic or source-level view of the code running on the system.

In a simple statically-linked and loaded system, a single image is run to describe the mapping of target addresses as image locations. To perform debugging, the debugger requires only the name of the code image. However, many systems, including operating systems such as Windows CE, 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 know what images are candidates to be loaded until they are loaded.

To debug systems like these, the debugger must be able to examine the target, to determine what images are loaded and from where they are loaded.

The problem is more complex when using trace, because trace data is historical information. Any embedded trace solution requires an image of the code that was executed to be available to the trace decompression software of the debugger, otherwise the debugger cannot decode the trace.

The compression algorithm used for trace conserves data bandwidth by broadcasting only the minimum of address information. 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 code in ARM state) to be inferred. This is difficult with, for example, virtual memory and software paging, because the debugger is unlikely to know where the code is executed from.

To resolve this problem, ETM10RV uses context IDs. These require both software and hardware support, as described in:

- *Software support for context ID* on page 6-3
- *Hardware support for context ID* on page 6-4.

6.2 Software support for context ID

When the operating system switches between binary images, or virtual memory spaces, it must update the value in the context ID register, which is part of coprocessor 15. The debugger must have access to a mapping file specifying the context ID that correlates to each binary image. With this information, the debugger can then associate each binary image with the correct part of the trace.

6.3 Hardware support for context ID

ETM10RV outputs the variable-length context ID value whenever trace is enabled, and as part of the periodic synchronization packet. This enables the current context ID value to be passed to the debugger. You can also filter out unwanted trace based on the current context ID using programmable trigger resources (see the *Embedded Trace Macrocell Specification)*.

To support tracing when only a partial binary image is available, the ETM10RV compression protocol maintains synchronization even as the ETM branches into unknown code regions. Trace is decompressable immediately again after jumping back into a region for which the code image is available.

Chapter 7
Physical Trace Port Signal Guidelines

This chapter contains some signal guidelines that can ensure correct operation of the ETM10RV and trace tools. It contains the following sections:

7.1 About trace port signal quality

When you use a *Trace Port Analyzer* (TPA), guaranteed operation of the TPA depends on the correct design of the ASIC and of the target *Printed Circuit Board* (PCB).

When integrating an ETM10RV 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 trace port frequencies greater than 100MHz, careful SPICE analysis of the system, including the characteristics of the package and the chosen Trace Port Analyzer is recommended.

7.2 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 important 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 you encounter this problem 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.

————————

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

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

7.3 PCB design guidelines

Two implementations are possible:

A dedicated trace port

The TPA 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 7-6.

7.3.1 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 a 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 (0.013cm) 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 ETM10RV output drivers
- the actual setup and hold provided by the ASIC ETM10RV outputs with reference to the ETM10RV **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 the ETM10RV output drivers and timings vary between ASIC vendors.

PCB track length

You must match all **TRACECLK** and **TRACEDATA[31:0]** 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 become 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 Place the series resistor as close as possible to the ASIC pin, ideally within half an inch (1.27cm). The value of this series resistor plus the output impedance of the signal driver must closely match the impedance of the PCB track. This is the recommended method.

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 equivalent to one rise time propagation delay or greater, 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.

7.3.2 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. It is important to minimize the effect of the TPA target header on non TPA-based signal usage and maintain the integrity of the trace measurements. The following constraints apply:

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 percent 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 percent of the rise and fall time. If length and speed requirements do not allow 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 *Logic Analyzer* (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 will now be reduced.

Note

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

7.4 EMI compliance

The trace port does not affect your EMI compliance testing because 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 trace port switching on overall system noise.

7.5 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 LA:

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 Users Guide* (Agilent document number E5903-97002) contains equivalent models for Agilent TPA and LA products.

Tektronix The Tektronix 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 appendix describes the signals used in ETM10RV. It contains the following sections:

- *Functional signals* on page A-2
- *DFT signals* on page A-4.

A.1 Functional signals

Table A-1 describes the functional ETM10RV signals.

Table A-1 ETM10RV functional signals

Signal name	Direction	Description
ARMTDO	Input	The **TDO** output signal from an external scan chain.
DBGACK	Input	The debug acknowledge signal driven by the ETM10RV macrocell. When HIGH, this signal indicates that the ARM1026EJ-S processor is in debug state.
DBGRQ	Output	Debug request. You can use this signal to stop the ARM1026EJ-S processor.
ETMCORECTL[30:0]	Input	Control signal inputs from the ARM1026EJ-S processor (see Table 3-3 on page 3-6).
ETMDA[31:0]	Input	The ARM1026EJ-S data address bus.
ETMDATA[63:0]	Input	The load, store, and coprocessor data from the ARM1026EJ-S processor.
ETMDATAVALID[1:0]]	Input	Valid signal for **ETMDATA** bus (one bit for each HIGH and LOW word).
ETMEN	Output	Trace port enable. Must not be used to power down functionality required by the ETM10RV unless it only relates to the trace port.
ETMIA[31:0]	Input	The ARM1026EJ-S instruction fetch address bus.
ETMR15BP[31:0]	Input	The instruction address for branch phantom instructions.
ETMR15EX[31:0]	Input	The instruction address for all non-branch phantom instructions.
EXTIN[3:0]	Input	General-purpose external inputs to the ETM, controlling tracing and other resources.
EXTOUT[3:0]	Output	External outputs from the ETM. Can be used to trigger hardware inside the ASIC, or external equipment such as a logic analyzer.
GCLK	Input	This is the clock used by the ETM10RV. It is the same clock that is used by the ARM1026EJ-S processor. **GCLK** is gated internal to ETM10RV to be disabled whenever **PWRDOWN** is asserted.
NRESET	Input	Power-on reset signal. Resets the ETM10RV context ID shadow register.
NTRST	Input	Active LOW JTAG reset. Used to reset everything in ETM10RV except the context ID shadow register.
PORTMODE[2:0]	Output	Indicates the frequency at which the **TRACECLK** requires to be run with reference to **GCLK**.

Table A-1 ETM10RV functional signals (continued)

Signal name	Direction	Description
PORTSIZE[2:0]	Output	Indicates the currently selected port size in use on the **TRACEPKT** bus.
PWRDOWN	Output	Indicates that the ETM is inactive and logic supporting it must be clock-gated to conserve power. This is asserted when the ETM is reset and remains asserted until it is first programmed This is connected to the core, although it might be used by other functionality.
SYSOPT[11:0]	Input	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.
DBGTCKEN	Input	Test clock enable for the DAP interface. This signal must be GCLK gated with the system DBGTCKEN signal, but this logic must be external to the synthesizable macrocell.
DBGTDI	Input	Test data input.
DBGTDO	Output	Test data output.
DBGTMS	Input	Test mode select.
TRACECLK	Output	Trace port clock used by TPA device to sample trace port outputs.
TRACEDATA[31:0]	Output	32-bit trace port. Only data on this bus has to be captured. This signal must hold the value from the previous cycle if no new data is output.
TRACECTL	Output	Indicates whether trace must be stored this cycle, in conjunction with **TRACEDATA[0]**.
TRACEVALID	Output	Trace is valid on this **CLK** cycle. Used by the On-chip trace buffer. Equivalent to **!TRACECTL | TRACEDATA[0]**, but only asserted for one **CLK** cycle. This can therefore be used as a clock enable for **TRACECTL** and **TRACEDATA**.
TRIGGER	Output	Trigger output this cycle. Used by the On-chip trace buffer. Equivalent to **TRACECTL & !TRACEDATA[0]**, but only asserted for one **CLK** cycle.

A.2 DFT signals

Table A-2 describes the ETM10RV test signals.

Table A-2 ETM10RV test signals

Signal name	Direction	Type	Description
SCANMODE	Input	Static	Prevents the asynchronous reset from being controlled from the synchronizer.
SE	Input	Dynamic	Scan enable for all internal clock domains. HIGH = shift mode. LOW = functional mode.
MUXINSEL	Input	Static	Configures the dedicated input wrapper cells for functional or test mode.
MUXOUTSEL	Input	Static	Configures the dedicated output wrapper cells for functional or test mode.
SI	Input	Dynamic	Scan input ports.
RSTSAFE	Input	Static	Enables the reset to the core.
WSEI	Input	Dynamic / Static	Scan enable for all input test cells in the wrapper. HIGH = shift mode. LOW = functional mode.
WSEO	Output	Dynamic	Scan enable for all output test cells in the wrapper. HIGH = shift mode. LOW = functional mode.
WSI	Input	Dynamic	Input ports for the wrapper scan chains.
SO	Output	Dynamic	Scan output ports.
WSO	Output	Dynamic	Output ports for the wrapper scan chains.
WSON	Output	Dynamic	Wrapper output port that changes after the falling edge of the clock.

Glossary

This glossary describes some of the terms used in this manual. Where terms can have several meanings, the meaning presented here is intended.

Abort	A mechanism that indicates to a core that it must halt execution of an attempted illegal memory access. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch Abort, a Data Abort, or an External Abort.
Application Specific Integrated Circuit (ASIC)	An integrated circuit that is designed to perform a specific application function. It can be custom-built or mass-produced.
ASIC	*See* Application Specific Integrated Circuit.
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.
Branch folding	Branch folding is a technique where, on the prediction of most branches, the branch instruction is completely removed from the instruction stream presented to the execution pipeline. Branch folding can significantly improve the performance of branches, taking the CPI for branches below one.

Branch phantom	The condition codes of a predicted taken branch.
Clock gating	Gating a clock signal for a macrocell with a control signal such as **PWRDOWN** and using the modified clock that results to control the operating state of the macrocell.
Context	The environment that each process operates in for a multitasking operating system. In ARM processors, this is limited to mean the Physical Address range that it can access in memory and the associated memory access permissions. *See also* Fast context switch.
DAP	*See* Debug access port.
Data instruction	An instruction that passed its condition code test and might have caused a data transfer, for example `LDM` or `MRC`.
Debug Access Port (DAP)	The collection of three mandatory and one optional terminals that form the input/output and control interface to a debug boundary-scan architecture. The mandatory terminals are **DBGTDI**, **DBGTDO**, **and DBGTMS**. The optional terminal is **NTRST**.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
Embedded Trace Macrocell (ETM)	A hardware macrocell which, 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.
Event	1 (Simple) An observable condition that can be used by an ETM to control aspects of a trace. 2 (Complex) A boolean combination of simple events that is used by an ETM to control aspects of a trace.
Fast context switch	In a multitasking system, the point at which the time-slice allocated to one process stops and the one for the next process starts. If processes are switched often enough, they can appear to a user to be running in parallel, as well as being able to respond quicker to external events that might affect them. In ARM processors, a fast context switch is caused by the selection of a non-zero PID value to switch the context to that of the next process. A fast context switch causes each Virtual Address for a memory access, generated by the ARM processor, to produce a Modified Virtual Address which is sent to the rest of the memory system to be used in

	place of a normal Virtual Address. For some cache control operations Virtual Addresses are passed to the memory system as data. In these cases no address modification takes place. *See also* Fast Context Switch Extension.
Fast Context Switch Extension (FCSE)	An extension to the ARM architecture that enables cached processors with an MMU to present different addresses to the rest of the memory system for different software processes, even when those processes are using identical addresses. *See also* Fast context switch.
FCSE	*See* Fast Context Switch Extension.
Half-rate clocking	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.
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.
Multi-ICE	A JTAG-based tool for debugging embedded 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.
SCREG	The currently selected scan chain number in an ARM DAP controller.
SPICE	Simulation Program with Integrated Circuit Emphasis. An accurate transistor-level electronic circuit simulation tool that can be used to predict how an equivalent real circuit will behave for given circuit conditions.
TPA	*See* Trace Port Analyzer.
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 packet	A discreet quantity of trace information comprising one or more bytes.
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.

Index