CoreSight™ Technology

System Design Guide

ARM®

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
ARM DGI 0012D (ID062610)

CoreSight Technology
System Design Guide

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.

Release Information

The table below shows the release state and change history of this document.

Change history

Date	Confidentiality	Issue	Change
29 September 2004	Non-Confidential	A	First release
20 July 2007	Non-Confidential	B	Updated for r1p0
29 April 2010	Non-Confidential	C	Updated for STM and TMC
25 June 2010	Non-Confidential	D	Update on Clock domain interactions

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

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

Web Address

`http://www.arm.com`

Contents
CoreSight Technology System Design Guide

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

List of Tables
CoreSight Technology System Design Guide

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

List of Figures
CoreSight Technology System Design Guide

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Preface

This preface introduces *CoreSight Technology System Design Guide*. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

About this guide

This is the *System Design Guide* for CoreSight Technology.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this guide, 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 guide is written for system designers, and others who make design decisions about *System-on-Chip* (SoC) designs that can use CoreSight Technology.

Using this guide

This guide is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to CoreSight Technology.

Chapter 2 *CoreSight Components and Systems*

Read this chapter for a description of the CoreSight components available and for information of how to combine components in typical systems for use in your CoreSight SoC designs.

Chapter 3 *Features of CoreSight Technology and ETM Architectures*

Read this chapter for a description of the features of CoreSight components, ETM, and ETM architecture versions that you can use in your CoreSight SoC designs.

Chapter 4 *Debug Access*

Read this chapter for a description of debug access in CoreSight systems.

Chapter 5 *Trace Capture*

Read this chapter for a comparison of the performance of different trace methods and how to make choices about trace capture.

Chapter 6 *Implementation*

Read this chapter for a description of the implementation issues you must consider when designing a CoreSight system.

Appendix A *Revisions*

Read this for a description of the technical changes between released issues of this book.

Glossary Read the Glossary for definitions of terms used in this guide.

Conventions

This section describes the conventions that this guide uses:

- *Typographical* on page x
- *Timing diagrams* on page x

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- *Signals* on page xi

Typographical

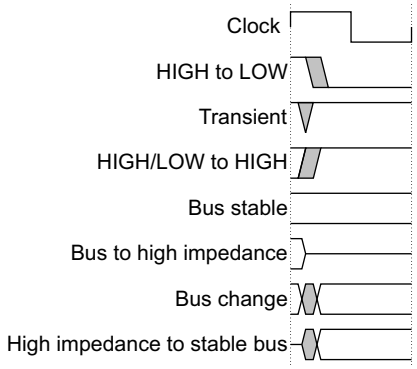
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes 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.
`mono`space	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

This guide contains one or more timing diagrams. The figure named *Key to timing diagram conventions* explains the components used in these diagrams. When variations occur they have clear labels. You must not assume any timing information that is not explicit in the diagrams.

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.



Key to timing diagram conventions

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

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.

Lower-case n Denotes an active-LOW signal.

Prefix A Denotes global *Advanced eXtensible Interface* (AXI) signals:

Prefix AR Denotes AXI read address channel signals.

Prefix AW Denotes AXI write 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 P Denotes Advanced Peripheral Bus (APB) signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This guide contains information that is specific to CoreSight Technology systems. See to the following documents for other relevant information:

- *AMBA™ Specification (Rev 2.0)* (ARM IHI 0011)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *AMBA 3 APB Protocol Specification* (ARM IHI 0024)
- *AMBA 3 ATB Protocol Specification* (ARM IHI 0032A)
- *CoreSight Architecture Specification* (ARM IHI 0029)
- *Embedded Trace Macrocell Architecture Specification* (ARM IHI 0014)
- *AHB Trace Macrocell Technical Reference Manual* (ARM DDI 0328)
- *CoreSight Components Implementation Guide* (ARM DII 0143)
- *CoreSight Components Technical Reference Manual* (ARM DDI 0314)
- *CoreSight DK9 Integration Manual* (ARM DII 0131)
- *CoreSight DK11 Integration Manual* (ARM DII 0092)
- *CoreSight DK-A8 Integration Manual* (ARM DII 0135)

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- *CoreSight ETM11 Technical Reference Manual* (ARM DDI 0318)
- *CoreSight ETM11 Implementation Guide* (ARM DII 0097)
- *CoreSight ETM11 Integration Manual* (ARM DII 0098)
- *CoreSight ETM9 Technical Reference Manual* (ARM DDI 0315)
- *CoreSight ETM9 Implementation Guide* (ARM DII 0093)
- *CoreSight ETM9 Integration Manual* (ARM DII 0094)
- *Intelligent Energy Controller r0p0 Technical Overview* (ARM DTO 0005)
- *ARM Debug Interface v5 Architecture Specification* (ARM IHI 0031A)
- *AMBA Network Interconnect (NIC-301) Technical Reference Manual* (ARM DDI 0397F)
- *ARM Debug Interface v5.1 Supplement* (DSA09-PRDC-008772)
- *System Trace Macrocell Programmers' Model Architecture Specification Version 1.0* (ARM IHI 0054)
- *CoreSight System Trace Macrocell Technical Reference Manual* (ARM DDI 0444A)
- *CoreSight System Trace Macrocell (STM) Integration and Implementation Manual* (PR430-PRDC-011726)
- *CoreSight Trace Memory Controller Technical Reference Manual* (ARM DDI 0461A)
- *CoreSight Trace Memory Controller (TMC) Integration and Implementation Manual* (PR430-PRDC-011743).

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DGI 0012D
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 1
Introduction

This chapter describes CoreSight Technology. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

1.1 About CoreSight systems

CoreSight systems provide all the infrastructure you require to debug, monitor, and optimize the performance of a complete *System on Chip* (SoC) design.

Historically, the following methods of debugging an ARM processor based SoC exist:

- Conventional JTAG debug. This is invasive debug with the core halted using:
  - breakpoints and watchpoints to halt the core on specific activity
  - a debug connection to examine and modify registers and memory and provide single-step execution.
- Conventional monitor debug. This is invasive debug with the core running using a debug monitor that resides in memory.
- Trace. This is non-invasive debug with the core running at full speed using:
  - a collection of information on instruction execution and data transfers
  - delivery off-chip in real-time
  - tools to merge data with source code on a development workstation for future analysis.

CoreSight Technology addresses the requirement for a multi-core debug and trace solution with high bandwidth for whole systems beyond the core, including trace and monitor of the system bus.

CoreSight Technology provides:

- debug and trace visibility of whole systems
- cross triggering support between SoC subsystems
- higher data compression than previous solutions
- multi-source trace in a single stream
- standard Programmer's Models for standard tool support
- open interfaces for third party cores
- low pin count
- low silicon overhead.

CoreSight Technology addresses a number of trends in SoC design that increase the debug challenge:

Frequency increases and trace generation

Systems are tracing more information per second and must transfer this out of the SoC. Pin interface frequencies are not rising as fast as on-chip frequencies.

Design Complexity

The interactions between cores in SoCs are crucial to understanding system behavior. System logic is sufficiently decoupled from core execution to require direct visibility. For example, a system cannot determine from inside a processor with cache, the amount of time a peripheral takes to respond to a memory request.

Clock and power domain implementations are complicated. The clock frequencies can change and any part of the system can enter a low-power mode at any time. Conventional JTAG-based systems must disable all power saving features to provide debug, but in many situations this is not acceptable.

Pin count Pin count is crucial. Chip package restrictions do not permit a separate trace port for each core in a chip and static switching between trace ports prevents debug of complex interactions.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

SoC designs must be flexible to provide the correct number of pins to achieve the required trace capture capabilities. It is not acceptable to double the number of pins because the frequency is 10% too high, or because the data bandwidth is 5% too high.

Performance optimization

Products must reach their performance targets. To make the most of high performance cores in SoC designs, it is essential to profile processor and bus activity to optimize performance.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

1.2 CoreSight features

This section describes some of the fundamental features of CoreSight Technology that enable you to address the issues and challenges of debugging complex SoCs. It contains the following sections:

- *Debug access*
- *Cross Triggering* on page 1-5
- *Trace* on page 1-6.

For more information on CoreSight components, see Chapter 2 *CoreSight Components and Systems* and the appropriate *Technical Reference Manuals* for the components.

1.2.1 Debug access

You gain debug access in CoreSight systems using the *Debug Access Port* (DAP) that provides:

- real-time access to physical memory without halting the core, and without any target resident code
- debug control and access to all status registers.

The same mechanism provides fast access for downloading code at the start of the debug session. This is faster than the traditional JTAG mechanism that uses the ARM core to write data to memory. You can still use the ARM core to write data to virtual memory and to ease migration when the debugger does not support this approach.

Figure 1-1 shows an example system with debug components and a DAP in a SoC design.

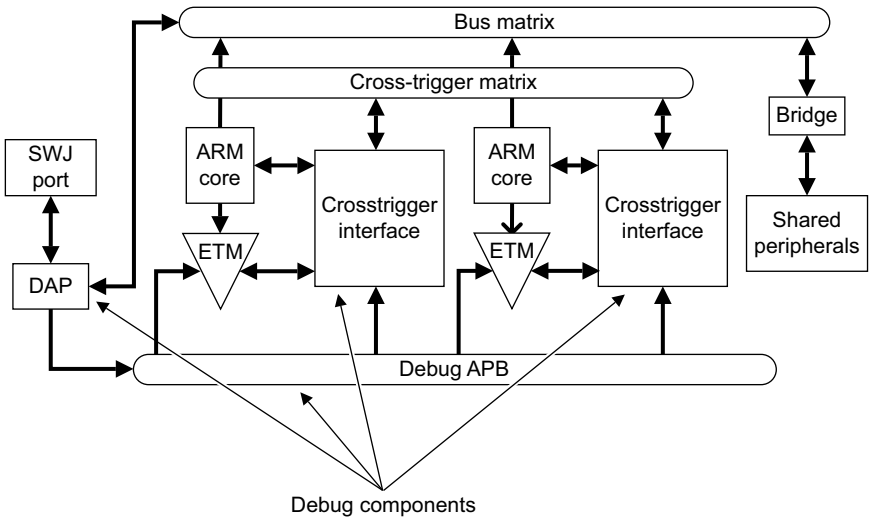


Figure 1-1 DAP connections inside a SoC

The DAP provides the following advantages for multi-core SoC designs:

- There is no requirement to run at the lowest common speed. A slow or powered-down component has no effect on access to other components. This means that power management has minimal impact on debug.
- The speed of access is not affected by the number of devices in the system. You have direct access to individual devices.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- You can add third party debug components using the *Advanced Microcontroller Bus Architecture* (AMBA) debug bus interface, AMBA 3 *Advanced Peripheral Bus* (APB), that provides internal and external access to the component.
- More than one core can control debug functionality, rather than restricting this to the core being debugged. One core can debug another. In particular, this enables a multi-core SoC when used as a single core platform to have complex on-chip debug and analysis features. You could use this, for example, during application development.

The DAP eases the physical implementation:

- You have a choice of physical debug interfaces. JTAG is no longer the single choice for a system, because other lower-cost interfaces are possible.
- The system does not have to support a fully asynchronous clock, **TCK**, because the DAP manages the clock. Debug clock synchronization is a problem for synthesized cores because the you must either keep the frequency of **TCK** well below the processor clock, or use a handshaking clock signal, **RTCK**.
- You do not require a return **TCK**, **RTCK**, off-chip because synchronization is performed inside the DAP.
- You can make significant savings in gate area by not having to implement a TAP controller and associated clock domain synchronization circuitry for each new debug element in the SoC.

The DAP supports increased security to prevent unwanted debug activity:

- you can add extra control over software access to the debug register file between the core and the debug bus
- you have a single access mechanism for hardware and software because the DAP arbitrates between accesses.

For 100% backward compatibility with existing tools, you can serially daisy-chain JTAG scan chains with the DAP to provide access to them.

If you do this, you lose some of the advantages of the CoreSight Technology.

For more information on the DAP, see *Debug Access Port* on page 2-5.

1.2.2 Cross Triggering

The *Embedded Cross Trigger* (ECT), comprising of the *Cross Trigger Interface* (CTI) and *Cross Trigger Matrix* (CTM), provides a standard interconnect mechanism to pass debug or profiling events around the SoC.

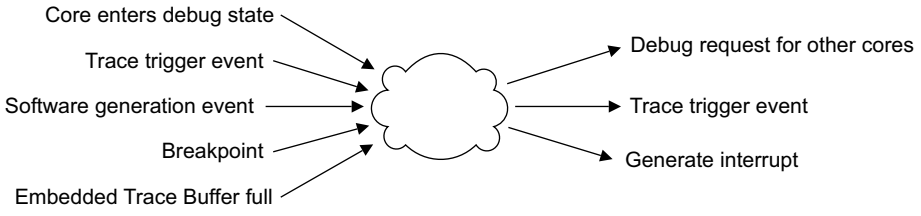


Figure 1-2 Cross triggering

The ECT provides you with a standard mechanism to connect different signal types. A set of standard triggers for cores and *Embedded Trace Macrocells* (ETMs) are predefined, and you can add triggers for third party cores.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

The ECT enables tool developers to supply a standard control dialog so that software programmers can connect trigger events.

1.2.3 Trace

CoreSight Technology provides components that support a standard infrastructure for the capture and transmission of trace data, a combination of multiple data streams by funneling together, and then output of data to a trace port, or storage in an on-chip buffer. Figure 1-3 shows some CoreSight components.

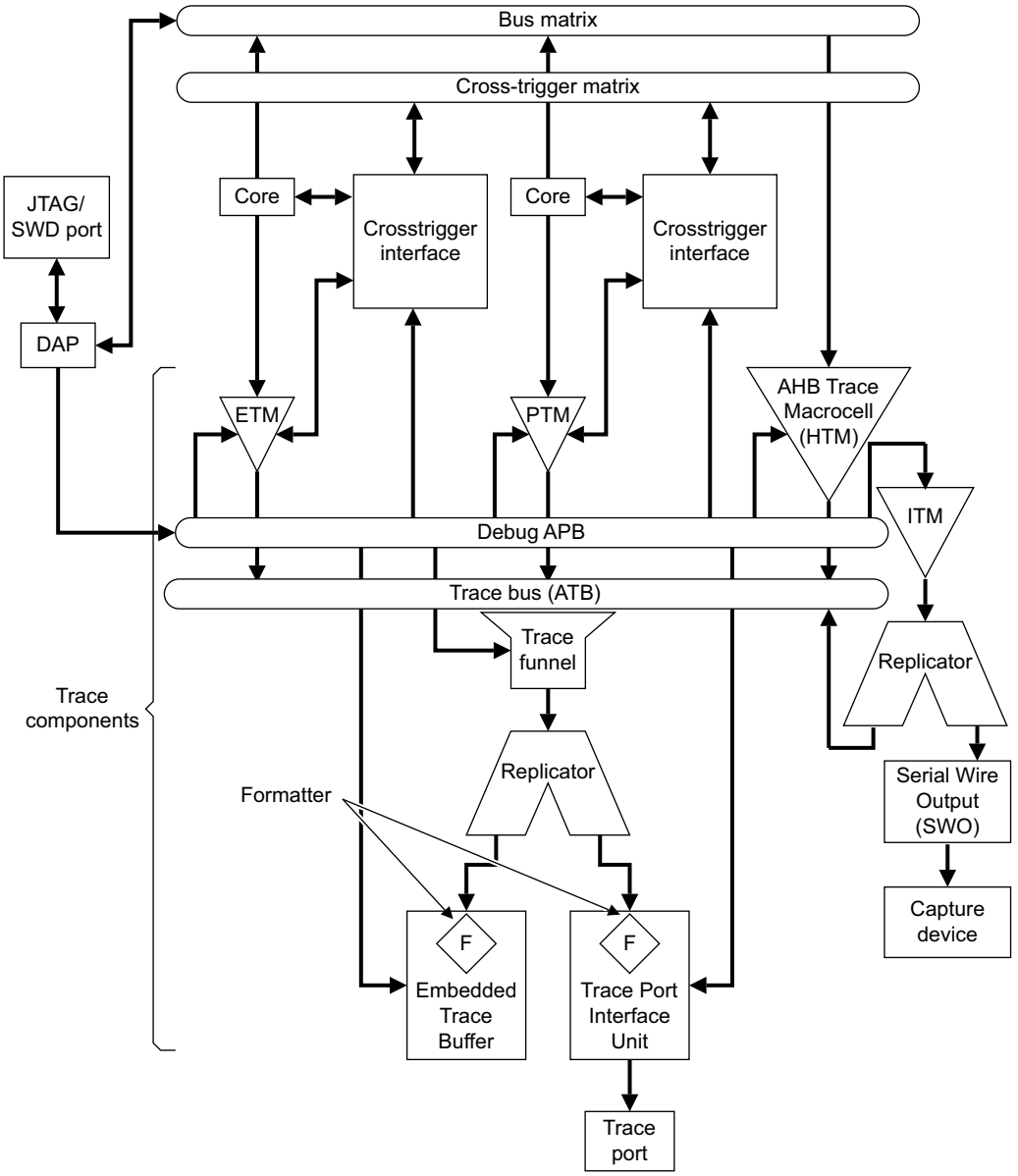


Figure 1-3 Example system with trace components

CoreSight Technology enables:

- simultaneous trace of asynchronous cores, busses and intelligent peripherals
- debug and trace of an AMBA 2 AHB bus
- tracing of instrumented bus masters

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- output of trace data to:
  - the Trace Port that can run at an independent frequency to on-chip busses
  - an embedded trace buffer for on-chip storage of trace data in dedicated RAM or system RAM
  - the DAP (*Debug Access Port*) for low-demand trace solutions in pin count limited targets.
- support for third party cores to enable debug control and trace capture through a standardized Programmer's Model and infrastructure.

For more information on trace components, see *CoreSight components* on page 2-4.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 2
CoreSight Components and Systems

This chapter describes the CoreSight components available and how you can connect them to form complete systems. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.1 About CoreSight systems and components

This chapter describes the individual components that make up CoreSight systems and provides some simple examples of CoreSight systems.

The following sections describe the main components of a CoreSight system:

- *Buses* on page 2-4
- *Control and access components* on page 2-5
- *Trace sources* on page 2-7
- *Trace links* on page 2-9
- *Trace sinks* on page 2-10
- *External debug hardware and software* on page 2-12.

Figure 2-1 on page 2-3 shows a system containing CoreSight components.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

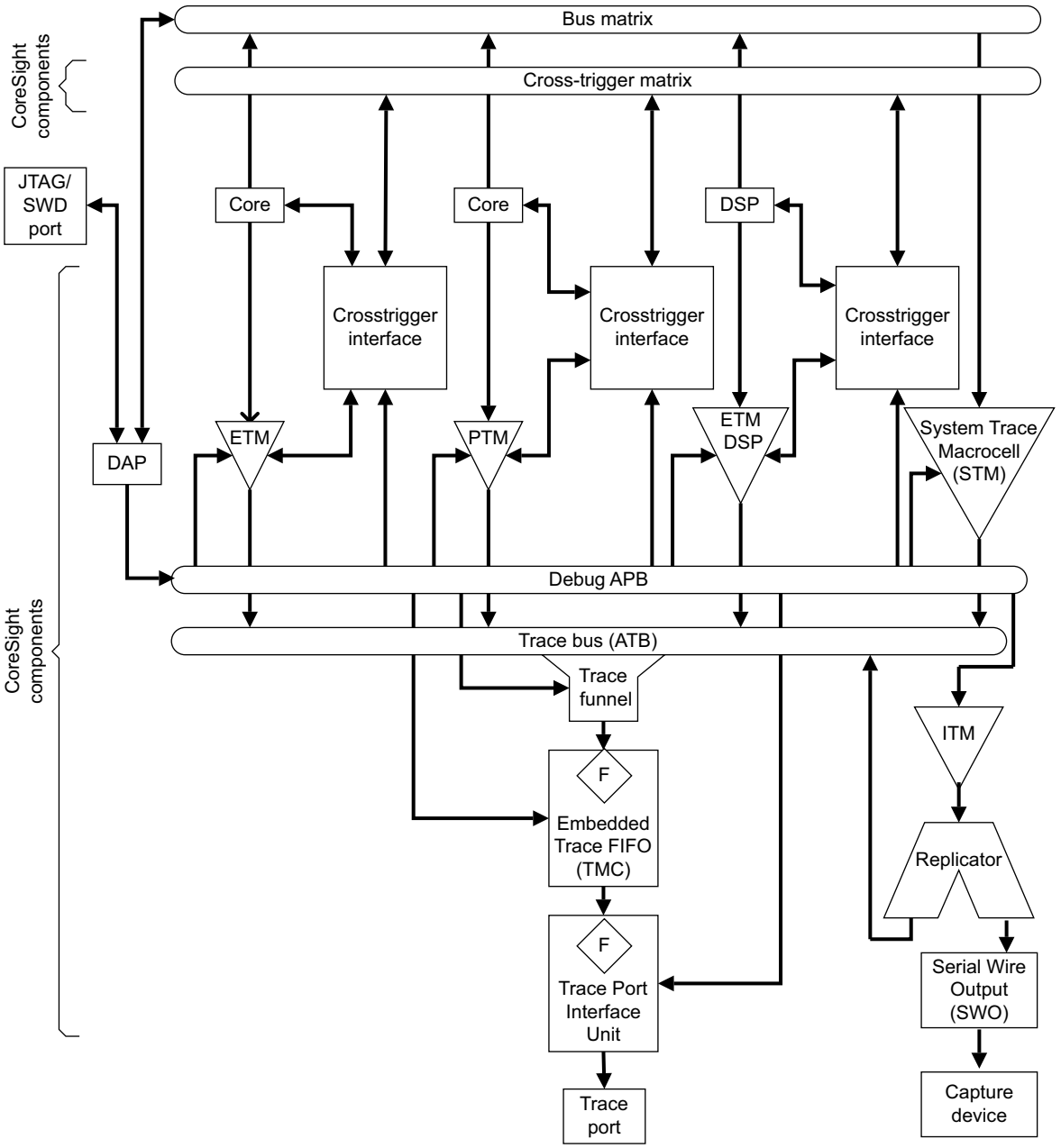


Figure 2-1 CoreSight system components

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.2 CoreSight components

This section describes the CoreSight components. For more information on CoreSight Technology, see the *CoreSight Design Kit Technical Reference Manual*.

2.2.1 Buses

The CoreSight systems use the following bus protocols to connect components together, and to enable integration in a SoC:

- *AMBA Trace Bus* (ATB)
- AMBA 3 *Advanced Peripheral Bus* (AMBA 3 APB)
- *Advanced High-performance Bus* (AHB)
- *AMBA Advanced eXtensible Interface* (AXI).

AMBA Trace Bus (ATB)

The ATB transfers trace data through the CoreSight infrastructure in a SoC. Trace sources are ATB masters, and sinks are ATB slaves. Link components provide both master and slave interfaces.

The ATB protocol supports:

- Stalling of trace sources to enable the CoreSight components to funnel and combine sources into a single trace stream.
- Association of trace data with the generating source using trace source IDs. A CoreSight system can trace up to 111 different items at any one time.
- Capture and transfer of multiple byte bus widths, currently to 32-bits.
- A flushing mechanism to force historic trace to drain from any sources, links, or sinks up to the point that the request was initiated.

For more information about ATB, see the *AMBA ATB Protocol Specification*.

AMBA 3 APB

CoreSight supports the AMBA 3 APB protocol to enable transfer extension using wait states.

The Debug APB bus uses the AMBA 3 APB protocol within a CoreSight system. The Debug APB is a bus dedicated to the connection of debug and trace components in a CoreSight-compliant SoC. All CoreSight components are configured and accessed over this bus through the APB-Mux in the DAP.

For more information about AMBA 3 APB, see the *AMBA 3 APB Protocol Specification*.

Advanced High-performance Bus (AHB)

CoreSight supports access to a system bus infrastructure using the *AHB Access Port* (AHB-AP) in the DAP. The AHB-AP provides an AHB master port for direct access to system memory.

CoreSight also supports AHB bus tracing using an *AHB Trace Macrocell* (HTM) that provides non-invasive debug visibility to any bus transactions on AHB connections.

For more information on AHB, see the *AMBA Specification*.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

AMBA Advanced eXtensible Interface (AXI)

CoreSight supports the use of AXI in the system interconnect. Direct access to the AXI system can be provided through a Cortex core as an AXI bus master, or through the use of an AHB to AXI bridge on the AHB Access Port in the DAP.

CoreSight also supports trace generation from bus masters on the AXI through the use of the STM that converts stimulus writes to the device into a trace data stream.

For more information on AXI, see the *AMBA Specification*.

2.2.2 Control and access components

Control and access components configure, provide access to, and control debug logic and the generation of trace. They do not generate trace, or process the trace data. The CoreSight control and access components are:

- the *Debug Access Port* (DAP)
- the *Embedded Cross Trigger* (ECT) that includes the *Cross Trigger Matrix* (CTM) and the *Cross Trigger Interface* (CTI).

Debug Access Port

The DAP enables debug access to the complete SoC through system master ports. Figure 2-2 on page 2-6 shows the structure of the DAP.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

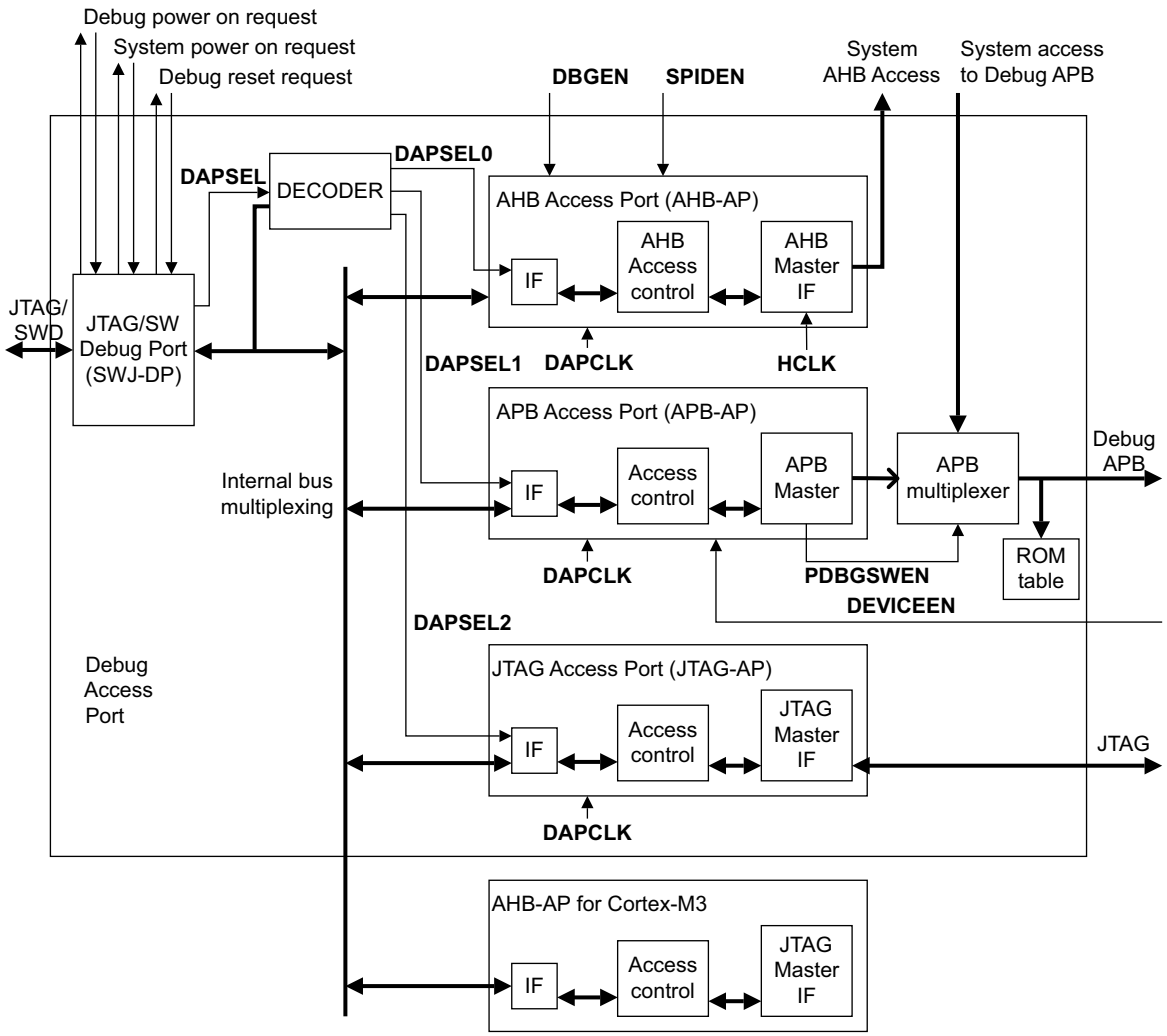


Figure 2-2 Structure of the CoreSight DAP component

Access to the CoreSight Debug APB is enabled through the *APB Access Port* (APB-AP) and *APB Multiplexer* (APB-Mux), and system access is provided through the AHB-AP and *JTAG Access Port* (JTAG-AP). The DAP has the following interface blocks:

- External Serial Wire or JTAG access using the *Serial Wire/JTAG Debug Port* (SWJ-DP).
- Internal system access using:
  - AHB-AP
  - APB-AP
  - JTAG-AP
  - AHB-AP for Cortex-M3, if present.
- An APB-Mux enables system access to CoreSight components connected to the Debug APB.
- The ROM table provides a list of memory locations of CoreSight components connected to the Debug APB. This is visible from both tools and system access and you must configure it during system implementation.

External read/write access to the internal interface is provided by the SWJ-DP. The SWJ-DP provides both a standard interface and an ARM Serial Wire Debug interface for debug access to an SoC through the DAP. It interfaces to the DAP internal bus.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Internal access to on-chip busses and other interfaces is provided by the *Access Ports* (APs). The APs are as follows:

- the AHB-AP that provides an AHB-Lite master for access to a system AHB bus
- the APB-AP that provides an AMBA 3 APB master for access to the Debug APB that configures all CoreSight components
- the JTAG-AP that provides JTAG access to on-chip components and operates as a JTAG master port to drive JTAG chains throughout the SoC.

For more information, see the *CoreSight Design Kit Technical Reference Manual*.

Embedded Cross Trigger

The ECT is a modular component that supports the interaction and synchronization of multiple triggering events within a SoC.

The ECT consists of the following types of module:

- A CTI. The CTI provides the interface between a component or subsystem and the *Cross Trigger Matrix* (CTM). The system requires a CTI for each subsystem that supports cross triggering.
- A CTM. The CTM combines the trigger requests generated from CTIs and broadcasts them to all CTIs as channel triggers. This enables subsystems to interact, cross trigger, with one another. You can connect CTMs together to increase the number of CTIs.

For more information, see the *CoreSight Design Kit Technical Reference Manual*.

2.2.3 Trace sources

Sources generate trace data and provide master ports to the AMBA Trace Bus. Depending on the licensed CoreSight components, the following trace sources can be provided:

- the *AHB Trace Macrocell* (HTM)
- CoreSight ETMs and PTMs for CPU trace:
  - *ETM9 for CoreSight* (ETM9CS)
  - *ETM11 for CoreSight* (ETM11CS)
  - *ETM for Cortex-A8* (ETM-A8)
  - *ETM for Cortex-R4* (ETM-R4)
  - *ETM for Cortex-M3* (ETM-M3)
  - *ETM for Cortex-A5* (ETM-A5)
  - *PTM for Cortex-A9* (PTM-A9).
- the *Instrumentation Trace Macrocell* (ITM)
- the *System Trace Macrocell* (STM).

CoreSight Technology also enables you to add third party devices, for example a DSP trace component.

AHB Trace Macrocell (HTM)

The HTM makes bus information visible that you can not infer from core trace using an ETM:

- An understanding of multi-layer bus utilization.
- Software debug. For example, visibility of access to memory areas and data accesses.
- Bus event detection for trace trigger or filters, and for bus profiling.

Figure 2-3 on page 2-8 shows the HTM connected to an AHB bus. For more information on HTM see the *AMBA Trace Macrocell Technical Reference Manual*.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

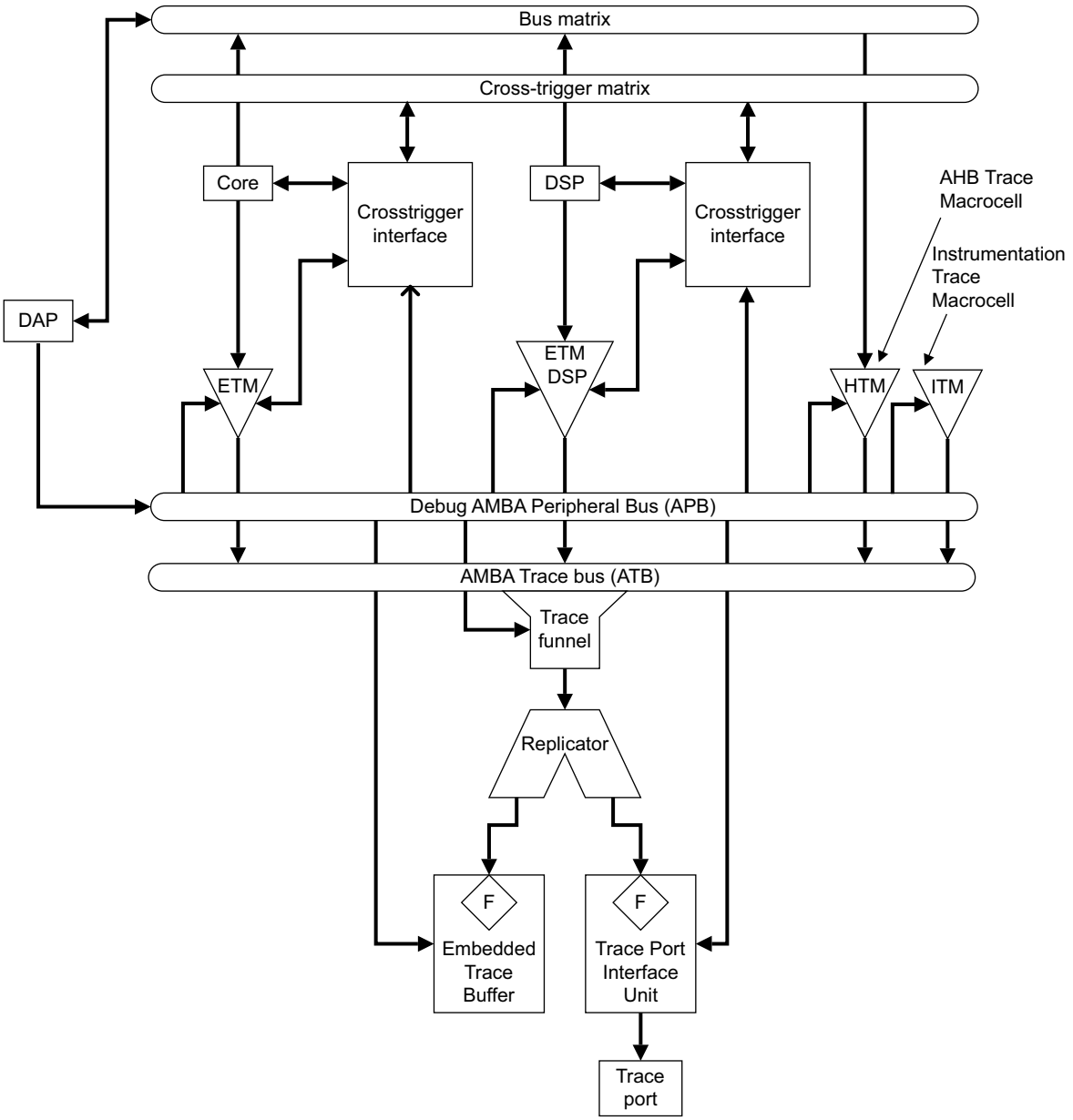


Figure 2-3 HTM connected to a multi-layer AHB system

Instrumentation Trace Macrocell and System Trace Macrocell

The Instrumentation Trace Macrocell (ITM) and *System Trace Macrocell* (STM) are application-driven trace sources that generate trace based on software written to the program interface. The ITM presents 32 APB registers, and the STM provides a set of 64K AXI registers that, on a write transaction, generate corresponding trace that indicates the register and value written.

Embedded Trace Macrocells (ETMs) and Program Trace Macrocells (PTMs)

The ETMs provide processor-driven trace through an ATB-compliant trace port. You can configure the ETM through the CoreSight APB programming interface. ETM9CS and ETM11CS both provide an asynchronous ATB master port that transfers trace data onto the CoreSight infrastructure.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

ETM9CS implements the ETMv3.2 protocol. ETM11CS implements the ETMv3.2 protocol with TrustZone™ and Thumb®-2 trace support.

ETM for Cortex-A8 and ETM for Cortex-R4 implement the ETMv3.3 protocol. However, the ETM for A8 is instruction and data address trace only.

ETM for Cortex-M3 implements the ETMv3.4 protocol. It only implements instruction trace.

PTM for Cortex-A9 implements the PTMv1 protocol to provide instruction flow trace.

For more information, see the following manuals:
- appropriate *CoreSight ETM Technical Reference Manual*
- *Embedded Trace Macrocell Architecture Specification*
- *CoreSight PTM-A9 Technical Reference Manual*
- *Program Flow Trace Macrocell Architecture Specification.*

2.2.4 Trace links

Links provide connection, triggering, and flow of traced data. The following sections describe the links:
- *Trace funnel*
- *Replicator*
- *Synchronous 1:1 ATB Bridge*
- *Embedded Trace FIFO (ETF)*.

Trace funnel

The Trace funnel combines up to eight trace sources on a single funnel. A static arbitration scheme selects the input trace stream to pass at any instant. The static arbitration permits reorganization of the slave port priorities between trace sessions. You can chain funnels together, with the ATB output from one funnel connected to an ATB input port of another. This enables you to both increase the number of inputs, and to connect independent systems together.

Replicator

The Replicator enables you to wire two trace sinks together and operate them on the same incoming trace stream. The input trace stream is output on two ATB ports that can then operate independently.

Synchronous 1:1 ATB Bridge

The Synchronous ATB Bridge provides a register slice that enables timing closure through the addition of a pipeline stage. It also provides a unidirectional link between two synchronous ATB domains. The bridge is 1:1 because both the input and output interfaces exist in the same clock domain. Because the bridge is a single register slice over the ATB interface, it temporarily holds one cycle of trace data within the register bank.

Embedded Trace FIFO (ETF)

The Embedded Trace FIFO is a trace buffer that uses a dedicated SRAM as either a circular capture buffer, or as a FIFO. The trace stream is captured by an ATB input that can then be output over an ATB output or the Debug APB interface.

The ETF is a configuration option of the *Trace Memory Controller* (TMC).

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.2.5 Trace sinks

Sinks are the endpoints for trace data on the SoC. CoreSight provides sinks that the following sections describe:

- *Trace Port Interface Unit (TPIU)* for output of trace data off-chip
- *Embedded Trace Buffer (ETB)* on page 2-11 for on-chip storage of trace data in RAM
- *Serial Wire Output* on page 2-11 for output of trace data over a single pin
- *Trace Port Interface Unit Lite* on page 2-12
- *Embedded Trace Router (ETR)* on page 2-12 for on-chip storage of trace data across an AXI interconnect
- *Enhanced ETB* on page 2-12 for on-chip storage of trace as a configuration option of the TMC.

Trace Port Interface Unit (TPIU)

The TPIU is an ATB slave that drains trace data off the chip. It acts as a bridge between the on-chip trace data and a data stream that is captured by a *Trace Port Analyzer* (TPA). The formatter within the TPIU combines the source data and IDs into a single data stream, to enable serialization of data, inserting trigger packets on trigger detection. You can bypass formatting if your system only traces a single source, and in this situation, no IDs are embedded. The TPIU supports off-chip port sizes from 2 to 34 pins. The off-chip trace port can operate asynchronously to the incoming trace data. Figure 2-4 shows a block diagram of the TPIU.

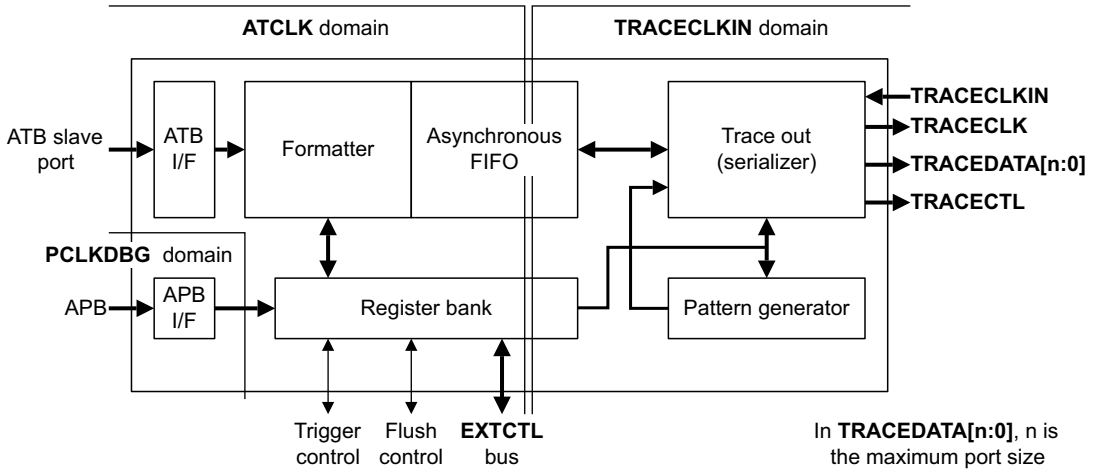


Figure 2-4 TPIU block diagram

The TPIU has the following ports:

- a Debug APB programming interface
- an ATB slave port for receiving trace data from a source or link
- an asynchronous *Trace Port* (TP) at the pins of the device for connection to a TPA
- trigger ports for connection to a CTI.

For more information, see the *CoreSight Technical Reference Manual*.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Embedded Trace Buffer (ETB)

The *Embedded Trace Buffer* (ETB) is an ATB slave and provides on-chip storage of trace data using a configurable sized RAM. The ETB stores data as follows:

- The ATB bus receives Trace Data.
- The Formatter in the ETB combines the source data and IDs into a single data stream. The Formatter operates in an identical manner to the Formatter in the TPIU.
- The ETB stores the data in RAM.

You can bypass formatting if your system only traces a single source, and so reduce the amount of data stored in RAM. The ETB accesses RAM using read and write pointers to permit memory access through the APB interface. Figure 2-5 shows a block diagram of the ETB.

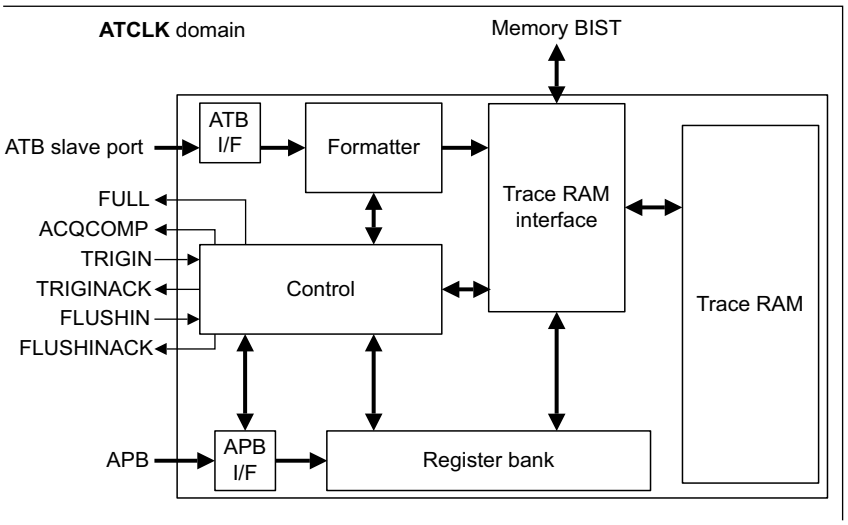


Figure 2-5 ETB block diagram

The ETB has the following ports:

- a Debug APB programming interface
- an ATB slave port for trace data for a source or link
- trigger ports for connection to a CTI
- a memory *Built In Self Test* (BIST) interface.

Serial Wire Output

Serial Wire Output (SWO) is a trace sink similar to the TPIU. It can only trace one source, the ITM. It outputs the data stream off-chip through a single-pin interface. You can select between the following operating modes for the single pin output:

- Manchester encoded stream
- NRZ-based UART byte structure, start bit, data bits, stop bit.

The SWO has an 8-bit ATB slave interface that you can connect to the CoreSight ITM. The SWO and the ATB interface are collectively called the *Serial Wire Viewer* (SWV).

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Trace Port Interface Unit Lite

The Trace Port Interface Unit Lite (TPIU-Lite) is a reduced feature, low gate count version of the TPIU. The following differences apply:

- A synchronous trace port that operates at **ATCLK** speed.
- Single trace source only. There is no formatter.
- 2-bit, 4-bit, 8-bit, 16-bit, and 32-bit trace port widths.
- No pattern generator.

Embedded Trace Router (ETR)

The ETR is a trace sink that redirects the trace stream onto AXI. It can utilize a single contiguous region or a scattered allocation of blocks for a circular buffer. Reading of the AXI based trace buffer can either be done directly over AXI from a normal bus master, or through the ETR as if it were an ETB. You can also program it to stream trace data to a single address location for use with high-speed links.

The ETR is a configuration option of the TMC.

Enhanced ETB

An enhanced ETB is available as a configuration option of the TMC. This configuration is similar to the Classic ETB with extra features such as being able to FIFO trace data to the Debug APB interface and more memory size options.

2.2.6 External debug hardware and software

Under the RealView and Keil brands are provides a range of development tools that support CoreSight debug and trace components. For more information on these tools, see, `www.keil.com` and, `www.arm.com`.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.3 CoreSight system examples

You can design a range of systems using CoreSight Technology. Some representative systems are described here and others are possible. The example systems are for:

- *Single core debug*
- *Single source trace*
- *Multi source trace in an single CPU system* on page 2-14
- *Multi source trace in a multi-core system* on page 2-14.

2.3.1 Single core debug

Figure 2-6 shows CoreSight debug on a single core system. This configuration provides no trace capabilities. You can use either the AHB-AP, APB-AP, or the JTAG AP to access system components. In this configuration, the JTAG-AP accesses the core, and the APB-AP is bridged to configure the CTI. The CTI supports triggering of the core from a designated resource, and enables connection to additional triggering resources if this sample is integrated into a larger system.

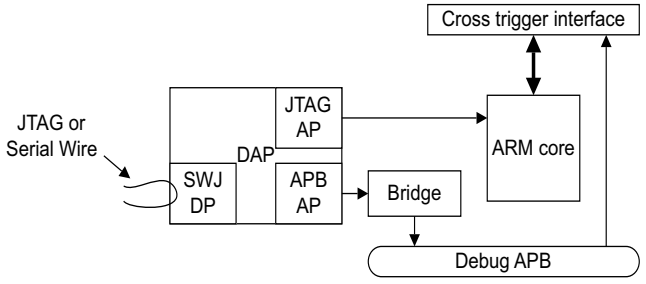


Figure 2-6 Single CPU trace and Debug APB debug access

2.3.2 Single source trace

Figure 2-7 shows single core trace using the CoreSight infrastructure. The CoreSight-compliant ETM outputs directly to a TPIU for direct output of core trace off-chip. The tracing of only a single trace source enables you to configure the TPIU in bypass mode because source IDs do not have to be embedded in the trace data. You can add a CoreSight ETB and replicator to provide on-chip storage of trace data.

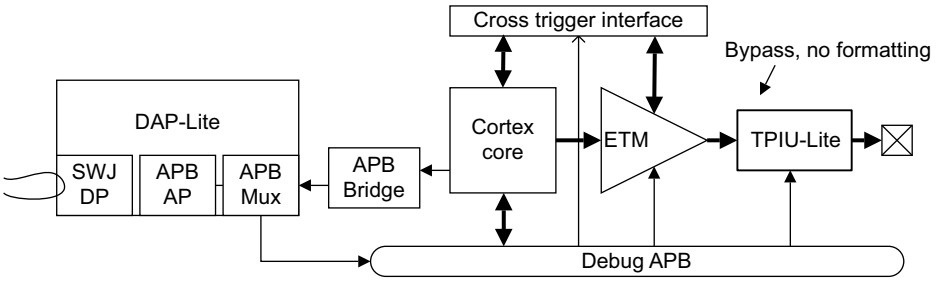


Figure 2-7 Single source trace with the TPIU formatting bypass

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.3.3 Multi source trace in an single CPU system

Figure 2-8 shows full trace capabilities in a single core system. The ETM provides core instruction and data tracing, and the HTM provides bus tracing. The trace funnel combines trace from all sources into a single trace stream, that is then replicated to provide on chip storage using the CoreSight ETB or output off chip using the TPIU. You can configure components using the DAP and operate cross triggering using the CTM and CTIs.

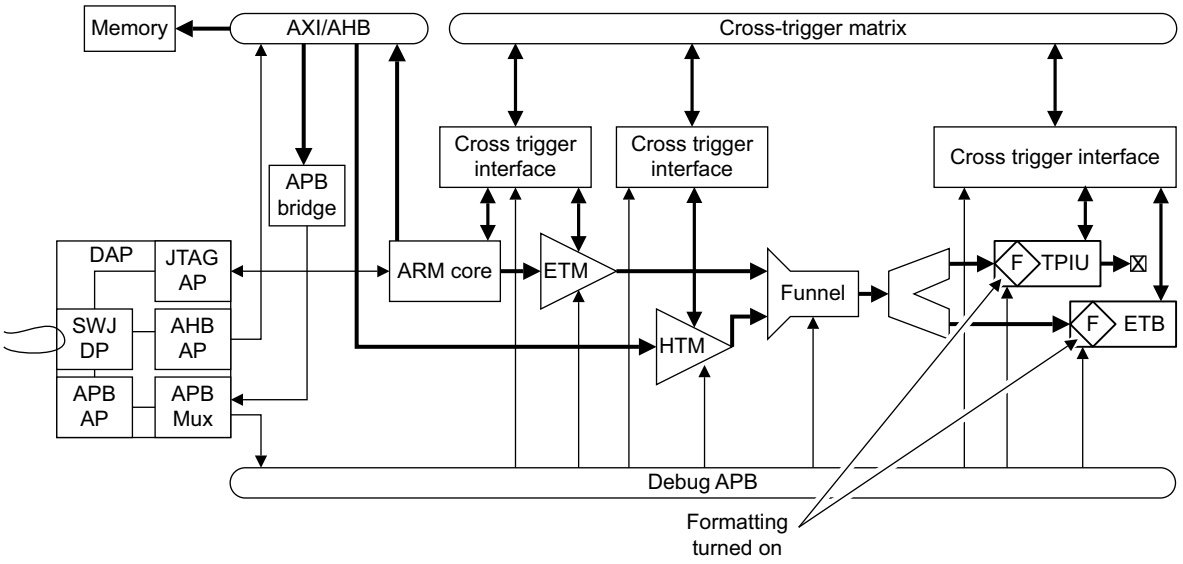


Figure 2-8 Full CoreSight trace with single core

2.3.4 Multi source trace in a multi-core system

Figure 2-9 on page 2-15 shows a system with a core and a third party DSP. A third smaller subsystem supports merging of multiple CoreSight ATB busses into a single trace stream. Figure 2-9 on page 2-15 also shows tracing of instrumented code using the STM.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

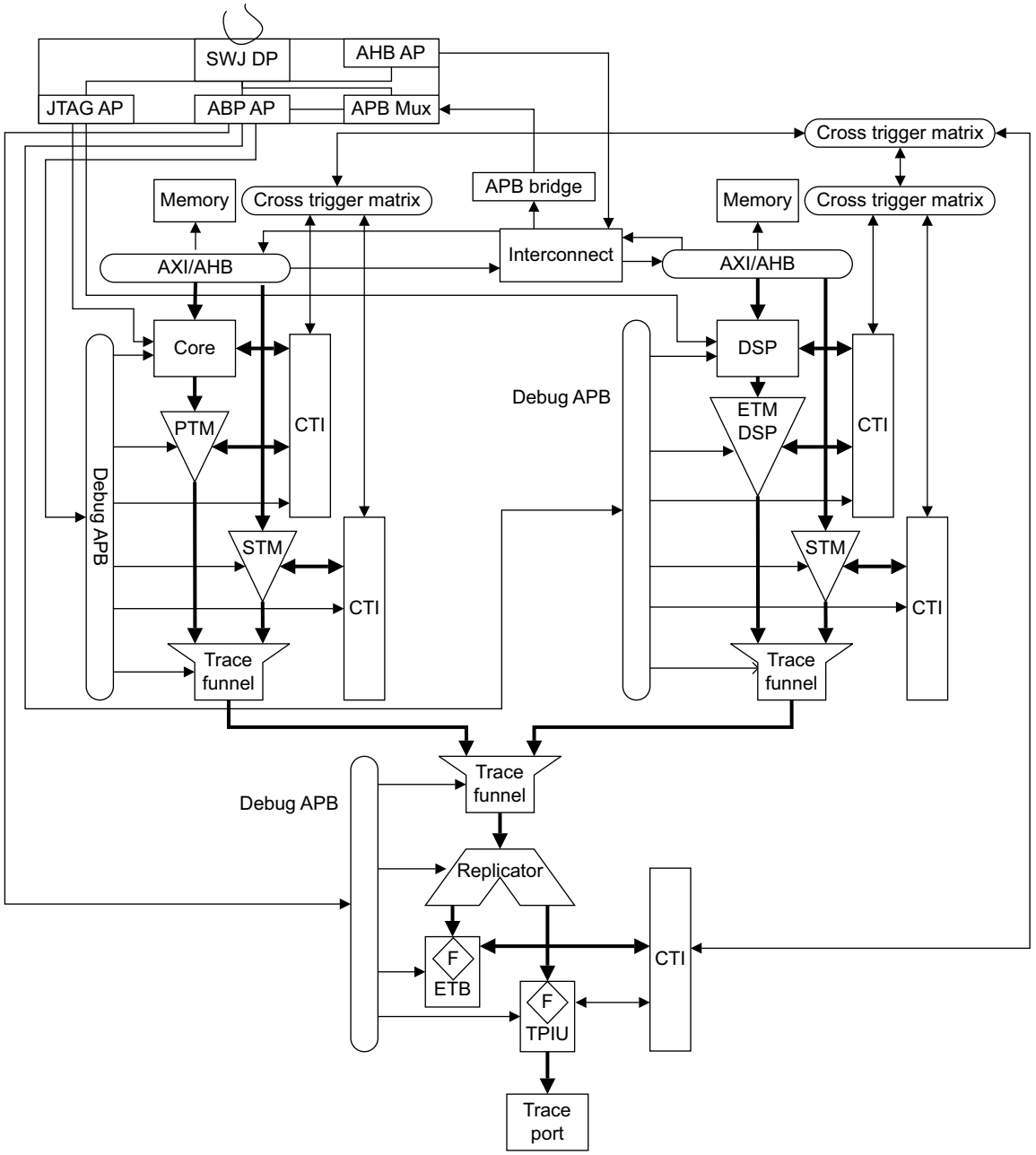


Figure 2-9 Full system trace with ARM core and CoreSight compliant DSP

This system requires bridges to support access through the DAP to the separate core and DSP subsystems.

Note

There are many alternative configurations possible with this set of components. The configuration Figure 2-9 shows does not necessarily correspond with the best configuration to meet your specific requirements.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

2.4 Illegal structures

There are a number of structures that you must avoid even though it is possible to create them with the CoreSight Technology. The following sections describe these illegal structures:

- *Stacked DAPs*
- *Duplicated IDs*
- *Feedback of source ID and data duplication*.

2.4.1 Stacked DAPs

You must not connect a JTAG-AP to a JTAG-DP or SWJ-DP on another DAP. Figure 2-10 shows this illegal structure.

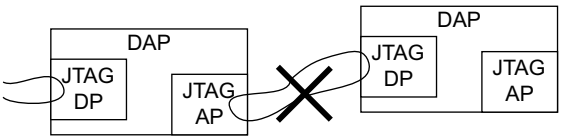


Figure 2-10 Unsupported DAP connection

2.4.2 Duplicated IDs

Two sources must never have the same source ID at any one time, and to the same component. You must not connect replicated source IDs at any point within a system. Figure 2-11shows this illegal structure.

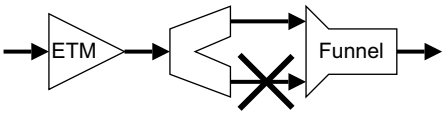


Figure 2-11 Unsupported replicator and funnel connection

2.4.3 Feedback of source ID and data duplication

You must not create feedback loops in the system that cause source ID duplication. Figure 2-12 shows this illegal structure.

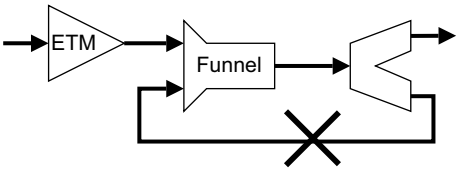


Figure 2-12 Unsupported feedback loop

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 3
Features of CoreSight Technology and ETM Architectures

This chapter lists the features of CoreSight Technology components and ETM architectures. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.1 About CoreSight Technology and ETM architectures features

CoreSight Technology provides a range of components with varying features that enable you to build trace and debug solutions to suit your SoC design. To aid comparison, this chapter lists the features of CoreSight Technology components:

- AHB-AP
- AHB-AP for Cortex-M3
- APB-AP
- APB-Mux
- ATB Bridge 1:1
- ATB Replicator
- CTI
- CTM
- DAP ROM
- ETB
- ETM9CS
- ETM11CS
- ETM-A8
- ETM-R4
- ETM-M3
- HTM
- ITM
- ITM-M3
- JTAG-AP
- JTAG-DP
- STM
- SW-DP
- SWJ-DP
- SWO
- TPIU
- TPIU-Lite
- Trace funnel.

It also describes the ETM, and PTM architecture versions, and lists the features of the trace macrocells:

- ETM9, medium plus
- ETM9CS
- ETM11RV
- ETM11CS
- ETM-A5
- ETM-A8
- ETM-R4
- ETM-M3
- PTM-A9.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2 CoreSight component data

The following sections list the component features:

- *DAP features* on page 3-4
- *Source features* on page 3-6
- *Link features* on page 3-8
- *Sink features* on page 3-10
- *Debug features* on page 3-13.

Table 3-1 on page 3-4 lists the DAP features:

- description
- type
- identification register
- input interface
- output interface
- authentication support
- power control support
- abort mechanism.

Table 3-3 on page 3-6 to Table 3-9 on page 3-13 list these features of the other components:

- description
- features
- primary non-programming inputs
- primary non-programming outputs
- programmer's model
- memory space
- part number
- device type identifier
- device ID.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2.1 DAP features

The DAP is the bridge for access to the Debug APB and system busses. Table 3-1 shows the DAP component features for Debug Ports.

Table 3-1 DAP component features for Debug Ports

Component name	JTAG-DP	SW-DP	SWJ-DP
ADIv5 Architecture Compliance	JTAG-DP	SW-DP	JTAG-DP and SW-DP
Description	Externally visible TAP that links to various on chip master interfaces	Low pin count, clock plus bidirectional data, alternative to a conventional JTAG interface	Combined TAP and Serial Wire interface, effectively JTAG-DP and SW-DP, with a switching sequence to translate between pin modes
Identification Register, excludes revision value	`0xBA00477`	`0xBA01477`	`0xBA02477` Serial Wire `0xBA00477` JTAG
Manufacturer/designer	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)
Part Number/AP Identification	`0xBA00`	`0xBA01`	`0xBA01` Serial Wire `0xBA02` Serial Wire with Multidrop support `0xBA00` JTAG
Input interface	JTAG	Serial Wire	Shared JTAG and Serial Wire
Output interface	DAP internal bus	DAP internal bus	DAP internal bus
Authentication support	No	No	No
Power control support	Yes	Yes	Yes
Abort mechanism	Initiator	Initiator	Initiator

Table 3-2 shows the CoreSight components for Access Ports.

Table 3-2 DAP component features for Access Ports

Component name	AHB-AP	APB-AP	JTAG-AP	AHB-AP for CM3
ADIv5 Architecture Compliance	Mem-AP	Mem-AP	JTAG-AP	Mem-AP
Description	DAP interface to an AHB system	DAP interface to the Debug APB	DAP interface to on-chip TAP controllers	DAP Interface to Cortex-M3 system including debug components
Identification Register, excludes revision value	`0x4770001`	`0x4770002`	`0x4770010`	`0x4770011`
Manufacturer/designer	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)
Part Number/AP Identification	`0x01`, AHB Bus	0x02, APB Bus	0x10, JTAG connection	0x11, AHB Bus
Input interface	DAP internal bus	DAP internal bus	DAP internal bus	DAP internal bus

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-2 DAP component features for Access Ports (continued)

Component name	AHB-AP	APB-AP	JTAG-AP	AHB-AP for CM3
Output interface	AHB-Lite	APB	Multiple JTAG	Cortex-M3 internal AHB
Authentication support	Yes	Yes	Yes	Yes
Power control support	Yes	No	Yes	Asynchronous check only
Abort mechanism	System transfer maintained. DAP internal bus released.	System transfer maintained. DAP internal bus released.	Aborts any JTAG transactions in progress. DAP internal bus released. JTAG-AP is returned to its reset condition. FIFOs are cleared and all registers are returned to their reset values.	System transfer cancelled through functional reset of the bus matrix, permitting access to NVIC and debug components.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2.2 Source features

Table 3-3 shows the trace source ETM component features.

Table 3-3 Trace source component features

Component name	ETM9CS	ETM11CS	ETM-A8	ETM-R4	ETM-M3
CoreSight compliant	Yes	Yes	Yes	Yes	Yes
Description	Trace source for instruction and data trace of the synthesizable ARM9 family of processors	Trace source for instruction and data trace of the ARM11 family of processors	Trace source for instruction and data address trace of the Cortex-A8 processor	Trace source for instruction and data trace of the Cortex-R4 and R4F processors	Trace source for instruction only trace of the Cortex-M3 processor
Features	Instruction only trace, data only trace, instruction and data trace, data suppression	Instruction only trace, data only trace, instruction and data trace, data suppression	Instruction only trace, data address only trace, instruction and data address trace, data suppression	Instruction only trace, data only trace, instruction and data trace, data suppression	Instruction only trace
ATB output	32 bit	32 bit	32 bit	32 bit	8 bit
Architecture reference	ETMv3.2	ETMv3.2	ETMv3.3	ETMv3.3	ETMv3.4
Stimulus input	ARM9	ARM11	Cortex-A8	Cortex-R4	Cortex-M3
Memory footprint	4KB	4KB	4KB	4KB	4KB
Designer ID	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)
Part number	0x910	0x920	0x921	0x930	0x924
Device ID	0x0	0x0	0x0	0x0	0x0
Device type	0x13	0x13	0x13	0x13	0x13
Lock Access Register	Bypassable	Bypassable	Bypassable	Bypassable	Yes
Claim tags	eight bits	eight bits	eight bits	eight bits	eight bits
Topology detection	Yes	Yes	Yes	Yes	Yes
Integration registers	Yes	Yes	Yes	Yes	No

Table 3-4 on page 3-7 shows the trace source HTM and ITM features.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-4 Trace source HTM and ITM features

Component name	HTM	ITM	CM3-ITM	STM
CoreSight compliant	Yes	Yes	No	Yes
Description	Trace source for AHB based activity. Behaves as a passive bus watcher.	Software generated trace source designed for printf style debugging requiring memory accesses to generate stimulus.	Software generated trace source for within the Cortex-M3 platform. Also includes the ability to trace data values through the Debug Watch Trace unit.	Software generated trace source designed for printf style debugging requiring memory accesses to generate stimulus. It can also generate trace based on activity of a set of hardware stimulus ports.
Features	Address trace, data trace, address and data trace, data suppression.	32 virtual stimulus registers, trigger generation on stimulus writes, maskable trace generation based on authentication level.	32 virtual stimulus registers, maskable trace generation based on transaction type.	64K virtual stimulus registers, trigger generation on stimulus writes, global time-stamping on stimulus writes, both lossy and guaranteed channels on stimulus generation, maskable trace generation based on Authentication level, 32 hardware based stimulus events.
ATB output	32-bit	8-bit	8-bit	32-bit
Architecture reference	None	None	None	STPv2, protocol
Stimulus input	32-bit or 64-bit AHB	APB	AHB	AXI
Memory footprint	4KB	4KB	4KB excluding DWT control	4KB excluding AXI stimulus ports
Designer ID	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)
Part number	`0x917`	`0x913`	`0x001`	`0x962`
Device ID	`0x0`	`0x020`	Non-applicable	`0x10000`
Device type	`0x43`	`0x43`	Non-applicable	`0x43`
Lock Access Register	Bypassable	Bypassable	Yes	Bypassable
Claim tags	4-bit	8-bit	None	4-bit
Topology detection	Yes	Yes	Yes	Yes
Integration registers	Yes	Yes	No	Yes

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2.3 Link features

Table 3-5 and Table 3-6 on page 3-9 shows the link component features.

Table 3-5 Link component features, part 1

Component name	ATB 1:1 Synchronous Bridge	Trace Funnel	Replicator
CoreSight Compliant	Yes	Yes	Yes
Description	ATB link to enable timing closure when intra-chip propagation delays might affect timing	Trace link that enable multiplexing of up to eight trace streams into a single stream	Component to enable dual sampling of trace data on two independent ATB slave components
Features	Fully registered interfaces	Static priority arbitration, minimum hold time	-
ATB input	1 x 32-bit	8 x 32-bit	1 x 32-bit
ATB Output	1 x 32-bit	1 x 32-bit	2 x 32-bit
ATB Clocking requirements	Input and output the same	Input and output the same	Input and output the same
Memory Footprint	None	4KB	None
Designer ID	-	ARM (`0x43B`)	-
Part number	-	`0x908`	-
Device ID	-	`0x0A0`	-
Device type		`0x11`	
Lock Access Register		Bypassable	
Claim tags	-	4 bits	-
Topology detection	Compatible	Yes	Compatible
Integration registers	Compatible	Yes	Compatible

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-6 Link component features, part 2

Component name	Asynchronous bridge	Upsizer	Embedded Trace FIFO (ETF)[a]
CoreSight Compliant	Yes	Yes	Yes
Description	ATB link to enable crossing of independent clock and/or power domains.	Component to adapt 8-bit ATB outputs of narrow trace sources to 32-bit ATB as used in the majority of CoreSight components.	Large ATB FIFO using a private SDRAM for buffering of trace data.
Features	Buffer to cross clock domains efficiently.	-	Buffer large quantities of trace data to allow averaging of bandwidth over larger windows of time. Can also operate as a traditional ETB, circular buffer based trace sink.
ATB input	1x [8 to 128-bit]	1x 8 bit	1x [32-bit to 128-bit]
ATB Output	1x (same width as input)	1 x 32 bit	1x (same width as input)
ATB Clocking requirements	Input and output fully asynchronous	Input and output the same	Input and output the same
Memory Footprint	None	None	4KB
Designer ID	-	-	ARM (`0x43B`)
Part number	-	-	`0x961`
Device ID	-	-	Configurable
Device type	-	-	`0x32`
Lock Access Register	-	-	Bypassable
Claim tags	-	-	4-bits
Topology detection	Compatible	Compatible	Yes
Integration registers	Compatible	No	Yes

a. The ETF is a configuration option of the *Trace Memory Controller* (TMC)

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2.4 Sink features

Table 3-7 and Table 3-8 on page 3-11 shows the sink component features.

Table 3-7 Sink component features, part 1

Component name	ETB11	TPIU	CS ETB	ETB[a]
CoreSight compliant	No	Yes	Yes	Yes
Description	Onchip trace capture device for capture of non-CoreSight legacy ETMs.	Parallel trace port for interfacing on-chip trace to off-chip capture devices.	Provides on-chip storage of trace data	Provides on-chip storage of trace data in a local SRAM.
Features	-	Includes pattern generation for thorough analysis of pin timing.	-	Includes programmable mode to use the SRAM as a FIFO when taking data over the Debug APB interface.
ATB input	Non-applicable	32 bit	32 bit	32-bits to 128-bits
Parallel Trace Port	No	1-32 data pins 1 clock pin 1 optional control pin	No	No
Serial Wire Output	No	No	No	No
Data Storage	1KB to 1MB	None	1 KB to 1 MB	1KB to 4GB
Formatter	Non-applicable	Yes	Yes	Yes
Memory Footprint	4KB + Memory size	4KB	4KB	4KB
Designer ID	Non-applicable	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)
Part number	Non-applicable	`0x912`	`0x907`	`0x961`
Device ID	Non-applicable	`0x0A0`	`0x000`	Configurable
Dev Type	Non-applicable	`0x11`	`0x21`	`0x21`
Lock Access Register	No	Bypassable	Bypassable	Bypassable
Claim Tags	None	4 bits	4 bits	4-bits
Topology detection	No	Yes	Yes	Yes
Integration registers	No	Yes	Yes	Yes

a. The ETB is a configuration option of the *Trace Memory Controller* (TMC).

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-8 Sink component features, part 2

Component name	TPIU-Lite	SWO	CM3 TPIU	Embedded Trace Router (ETR)[a]
CoreSight compliant	Yes	Yes	Yes	Yes
Description	Low gate count parallel trace interface between on-chip trace and off-chip capture devices.	Serial Wire output for Instrumentation Trace Macrocell trace to off-chip capture devices.	Cortex-M3 specific, parallel and serial trace interface to off-chip trace capture devices.	Provides on-chip storage of trace data in the system through an AXI interface.
Features	-	-	Includes trace funnel	Contains the following modes of using the system memory: • basic circular buffer, single contiguous block of memory • circular buffer using scatter-gather, list of arbitrary blocks of memory • single location for streaming trace data to a peripheral device location.
ATB input	32-bits	8-bits	2 x 8 bit	32-bits, 128 bits
Parallel Trace Port	2, 4, 8, 16, or 32 data pins 1 clock pin 1 control pin	No	1, 2, or 4 data pins 1 clock pin	No. Data is output over AXI at the same width as ATB input.
Serial Wire Output	No	Manchester or NRZ encoded	Manchester or NRZ encoded	No
Data Storage	None	None	None	None
Formatter	No	No	Yes	Yes
Memory Footprint	4KB	4KB	4KB	4KB
Designer ID	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)	ARM (`0x43B`)
Part number	`0x941`	`0x914`	`0x923`	`0x961`
Device ID	`0x000`	`0xEA0`	`0xCA0` or `0xCA1`	Configurable
Dev Type	`0x11`	`0x11`	`0x11`	`0x21`
Lock Access Register	Bypassable	Bypassable	No	Bypassable
Claim Tags	4-bits	4-bits	4-bits	4-bits
Topology detection	Yes	Yes	Yes	Yes
Integration registers	Yes	Yes	No	Yes

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

a. The ETR is a configuration option of the *Trace Memory Controller* (TMC).

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.2.5 Debug features

Table 3-9 and Table 3-10 on page 3-14 shows the debug component features.

Table 3-9 Debug component features, part 1

Component name	DAP-ROM[a]	CTI	CTM
CoreSight compliant	Yes	Yes	Yes
Description	Table for pointers to various debug and trace components present on the same memory structure.	Interface to enable the cross connection of any attached debuggers to and from other trigger stimulus ports.	Channel link to enable more than two CTIs to link together and share trigger information.
Features	Begins in a blank format and must be updated by the system creator to reflect the system implementation with allocation for 32 components.	Eight trigger inputs and eight outputs with extra levels of multiplexing possible, selectable handshaking and synchronizers.	Four-way interconnect, connects to CTIs or other CTMs to link up larger numbers of CTIs. Selectable handshaking and synchronizers on channel interfaces.
Non-programming interfaces	None	8x Trigger inputs 8x Trigger Outputs 1x Channel Interface	4x Channel Interface
Memory Footprint	4KB	4KB	None
Designer ID	Implementation defined	ARM (`0x43B`)	-
Part number	Implementation defined	0x906	-
Device ID	Non-applicable	`0x40800`	-
Dev Type	Non-applicable	`0x14`	-
Lock Access Register	No	Bypassable	-
Claim Tags	None	8-bits	-
Topology detection	Non-applicable	Yes	Compatible
Integration registers	Non-applicable	Yes	Compatible

a. The DAP-ROM is supplied as part of the DAP and DAP-Lite.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-10 Debug component features, part 2

Component name	CTI-A8	Cortex-A9	Cortex-A8	Cortex-A5	Cortex-R4
CoreSight compliant	Yes	Yes	Yes	Yes	Yes
Description	Interface to enable the debug logic, ETM and PMU to interact with each other and with other CoreSight components.	Debug Interface to the Cortex-A9 processor.	Debug Interface to the Cortex-A8 processor.	Debug Interface to the Cortex-A5 processor.	Debug Interface to the Cortex-R4 processor.
Features	Implements a configured set of seven trigger inputs and nine trigger outputs of which some are externally defined.	The processor debug unit provides a set of control registers to enable stopping of program execution, examining and altering processor state and restarting the processor core.	The processor debug unit provides a set of control registers to enable stopping of program execution, examining and altering processor state and restarting the processor core.	The processor debug unit provides a set of control registers to enable stopping of program execution, examining and altering processor state and restarting the processor core.	The processor debug unit provides a set of control registers to enable stopping of program execution, examining and altering processor state and restarting the processor core.
Non-programming interfaces	9x Trigger inputs 9x Trigger Outputs 1x Channel Interface	-	-	-	-
Memory Footprint	4KB	4KB	4KB	4KB	4KB
Designer ID	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)	ARM (0x43B)
Part number	0x922	0xC09	0xC08	0xC05	0xC14
Device ID	0x40906	0x0	0x0	0x0	0x0
Dev Type	0x14	0x15	0x15	0x15	0x15
Lock Access Register	Bypassable	Bypassable	Bypassable	Bypassable	Bypassable
Claim Tags	None	8-bits	8-bits	8-bits	8-bits
Topology detection	Yes	Yes	Yes	Yes	Yes
Integration registers	Yes	Yes	Yes	Yes	Yes

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

3.3 Architectural features of ARM trace sources

The ETM and PTM are trace sources that monitor ARM processors. Each ETM and PTM are associated with certain processor lines, and each ETM and PTM version conforms to certain ETM and PTM architectures. The architecture consists of a generic programmers model and a trace protocol. The ETM programmers model is consistent for the main revisions of the architecture but the protocol has developed.

ETMv1 The first ETM Architecture. Used on ETM9 and ETM7 devices. Pipeline execution of instructions is represented within a 3-bit bus on a cycle-by-cycle basis of the activity of the core, **PIPESTAT**, and data trace appears on a separate, independent bus, **TRACEPKT**.

ETMv2 An extension of ETMv1 with more structured information appearing on the secondary information pipeline, **TRACEPKT**, with the introduction of P-Headers, packet header, and an increase of the **PIPESTAT** bus to 4-bits to provide more optimal indication of processor execution.

ETMv3 Major revision to previous ETM protocols. All information, data transfers, and instruction execution, is based on byte-size packets with no pipeline status. Data suppression enhances FIFO usage and reduces overflow regularity. The byte protocol and the removal of PIPESTAT make it possible to implement asynchronous trace outputs and support CoreSight.

PFTv1 A new protocol designed to only offer program flow trace, where only branches and exceptions are traced using minimal trace bandwidth. The protocol is byte-based, similar to ETMv3. The PFT architecture is fully CoreSight-complaint.

For more information about the ETM and the various architectural differences, see the *ETM Architecture Specification*.

For more information on the PTM architecture, see the *PTM Architecture Specification*.

Table 3-11 and Table 3-12 on page 3-16 list the features of the ETMs.

Table 3-11 ARM trace source component features, part 1

Feature	ETM9, medium plus	ETM11RV	CoreSight ETM9	CoreSight ETM11
Architecture version	ETMv1.0-v1.3	ETMv3.1	ETMv3.2	ETMv3.2
Address comparator pairs	4	4	4	4
Data comparators	2	2	2	2
Context ID comparators	-	1	1	1
MMDs	8	0	0	0
Counters	2	2	2	2
Sequencer	Yes[a]	Yes	Yes	Yes
Start/stop block	Yes	Yes	Yes	Yes
EmbeddedICE comparators	2	0	2	0
External inputs	4	4	4	4
External outputs	1	2	2	2
Extended external inputs	-	20	0	20

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table 3-11 ARM trace source component features, part 1 (continued)

Feature	ETM9, medium plus	ETM11RV	CoreSight ETM9	CoreSight ETM11
Extended external input selectors	0	2	0	2
ASIC Control Register bits	8	8	8	8
Data suppression	-	Yes	Yes	Yes
Software access to registers	-	Coprocessor	Memory	Memory
Readable registers	-	Yes	Yes	Yes
Fifo size	45	69	60	72
CoreSight-compliant	No	No	Yes	Yes
Fetch comparisons	Yes	No	No	No

a. Yes = supported.

Table 3-12 ARM trace source component features, part 2

Feature	CS PTM-A9	CS ETM-A8	CS ETM-A5	CS ETM-R4	CS ETM-M3
Architecture version	PFTv1.0	ETMv3.3	ETMv3.5	ETMv3.3	ETMv3.4
Address comparator pairs	4	4	4	4	0
Data comparators	0	2	2	2	0
Context ID comparators	1	1	1	1	0
MMDs	0	0	0	0	0
Counters	2	2	2	2	0
Sequencer	Yes	Yes	Yes	Yes	No
Start/stop block	Yes	Yes	Yes	Yes	Yes
EmbeddedICE comparator inputs	0	0	0	0	4
External inputs	4	4	4	4	2
External outputs	2	2	2	2	0
Extended external inputs	52	49	30	47	0
Extended external input selectors	2	2	2	2	0
ASIC Control Register bits	8	8	8	8	0
Data suppression	N/A	Yes	Yes	Yes	No
Software access to registers	Memory	Memory	Memory	Memory	Memory
Readable registers	Yes	Yes	Yes	Yes	Yes

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 4
Debug Access

This chapter describes debug-related features in a system using the DAP and ECT, and methods of connecting components together for interoperability. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.1 About debug access

This chapter describes:

- debug related features in a system
- methods of connecting components together for interoperability

This chapter focusses on the DAP and ECT structure.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.2 Access to the system

Most methods of debugging on a SoC begin with access to the SoC through a JTAG connection. This section describes:

- *JTAG direct to core access*
- *DAP access*.

4.2.1 JTAG direct to core access

Maintaining the existing access mechanism of a SoC has the following advantages:

- it does not require tool modifications
- it reuses existing methodologies for connection and testing.

Because this method of access goes through the core, it provides any native memory address translations from virtual to physical. It has the disadvantages that it requires an understanding of how the core accesses memory, and might also take several cycles depending on the instruction set.

Figure 4-1 shows an external JTAG connection connected to some memory through a processor.

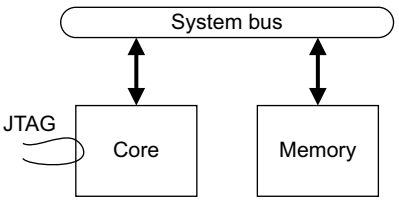


Figure 4-1 JTAG connection

External tools connected through the JTAG link can access the system memory by instructing the processor, in Halted-Debug Mode, to fetch and save values from the memory directly. This method permits the automatic translation of the *Virtual Modified Address* (VMA) that the core sees, to the physical memory addressing of the system.

4.2.2 DAP access

The purpose of the DAP is to create a bridge between different bus structures and external tools. At a basic level, it has one input controlling interface, the *Debug Port* (DP), and several output masters, the APs.

The DAP can provide a bridge to:

- JTAG through the JTAG-AP, that permits the separation of off-chip and on-chip links without one affecting the other.
- AHB through the AHB-AP, for connection to the bus matrix. This permits access to all on-chip peripherals that the core uses, such as RAM. The DAP can support other bus protocols with an AHB bridge, for example AHB to AXI.
- APB through the APB-AP. This is wired in conjunction with an APB Multiplexor in the DAP that arbitrates between accesses from the APB-AP and the primary system bus. This bus is connected to all CoreSight-compliant debug and trace components.

The structure of the DAP provides a single flexible and scalable platform for access to locations throughout an entire SoC. The internal design of the DAP gives each AP independence from the controlling DP permitting connections across different clock and power domains.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

The DAP supplied with the Design Kit permits an independent AHB power domain and can manage transactions from the System APB in a different power domain. However, the main DAP and the Debug APB must use the same clock.

Individual Access and Debug Ports within the DAP provide different features that might not be present in existing solutions:

AHB-AP This AP gives direct access to an AHB bus, or AXI bus through a bridge, and does not require a processor to stop or scan instructions though the processor. The AHB-AP only performs single transactions over AHB so that invasiveness is kept low. To decrease invasiveness more, it is recommended that the bus arbiter selects the AHB-AP as a low priority master.

The AHB-AP connects directly to the bus, therefore there is no overhead for transfers because there is no requirement to translate reads and writes into Op Codes for the processor.

Note

The AHB-AP bypasses the processor to access the memory subsystem and views the physical bus addresses. This is not the same for the processor that views the virtual addresses.

JTAG-AP This component maintains the existing JTAG based debugging of cores. With independent JTAG chains, it is possible to have a JTAG chain attached to a core that might power-down, for example as part of an IEM domain, without affecting the JTAG access to other chains.

APB-AP The DAP permits the APB-AP to be a dedicated channel for control of and access to debug and trace components. This use of an AP for debug components ensures that there is no invasiveness onto the system bus.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.3 Access to debug components

Both externally hosted debug agents and on-chip debug agents require access to debug components. For example, a debug monitor. Within CoreSight, these debug components are provided on a dedicated bus, the Debug APB that ensures a clear separation between system memory space and debug memory space. External tools can directly access debug components through the APB-AP within the DAP, on-chip agents must navigate the system memory bus first before being multiplexed with external debug access in the APB-MUX. Both of the memory mapped regions that is debug space within system memory and direct access to debug memory have the same offsets. However, system memory typically places this debug region at non-zero offset. This section describes:

- *Debug memory overview*
- *System interface*
- *Debug memory decoding and the ROM table* on page 4-6
- *Example memory system* on page 4-6
- *Example memory map* on page 4-7.

4.3.1 Debug memory overview

The DAP is setup with a ROM table located at address offsets `0x0000_0000` and `0x8000_0000` for system and APB-AP accesses respectively. Although the full address range is available for debug components, it is split into the following regions:

- `0x0000_0000 to 0x7FFF_FFFF`
- `0x8000_0000 to 0xFFFF_FFFF`

The upper half with **PADDRDBG[31]** equal to `1'b1` can only be reached by external debug tools accessing through the APB-AP. The ROM table is located at the bottom of this region as the BASE register within the APB-AP indicates.

The lower half, **PADDRDBG[31]** equal to `1'b0`, can only be accessed from the system interface that is enforced through the restriction of the APB system input to the APB-MUX only accepting **PADDRSYS[30:2]**. This division of memory enables bypassing of lock-access mechanisms that can be used by debug components to prevent accidental damage to debug control registers. For more information on the use of **PADDRDBG[31]**, see the *CoreSight Architecture*.

4.3.2 System interface

To connect the Debug APB into the system memory, an APB multiplexer is provided within the DAP. The bus type of the system must be converted to APB, for example through an AXI to APB bridge or an AHB to APB bridge. The address bits used to decode the region within the system memory must not be passed into the interface, only the address bits for that region, these are then decoded to individual components within the Debug APB address decoder. Because only a reduced address range is used for the Debug APB, this must exist on an aligned boundary. It is possible to translate system memory space into an aligned region, but this must be done prior to connection to the system interface on the DAP to ensure that the ROM table and its contents are still correct. Any high address bit that was used to decode the debug address region within the system memory, must be tied LOW on the system input to the DAP. For example, if debug region exists within the range `0x3F50_0000` to `0x3F50_FFFC`, only **PADDRSYS[15:2]** must be connected with **PADDRSYS[30:16]** = `0x0000`.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.3.3 Debug memory decoding and the ROM table

It is advisable to place a ROM table for the debug system at the bottom of the debug address range for the CoreSight DK DAP, and one is already supplied at this fixed location. From this base address, offsets can be generated at 4KB intervals for allocation to CoreSight components present on the Debug APB. The address decoder for the bus must then correspondingly decode those offsets specified within the ROM table. Where features such as the bypassing of lock access, are implemented on **PADDRDBG[31]** being HIGH, the ROM table must be present at this offset location, that is `0x8000_0000` rather than `0x0000_0000` such as the APB-AP in the DAP.

To ensure that the address decoder for the Debug APB behaves in the same way for both types of accesses, the decoder must ignore **PADDRDBG[31]** but still decode all the remaining address bits, that is, **PADDRDBG[30:12]** even though the system accesses only have a limited range from `0x0000_0000`.

4.3.4 Example memory system

The example system shows the address bus breakdown for the system with:

- A bus master. For example, an ARM processor.
- Three system slaves. For example, memory and peripherals.
- A fourth slave that is the bridge to the debug memory.

Accesses from the bus master are multiplexed with those from the APB-AP inside the DAP and decoded to a ROM table and four debug components. When a slave is not selected, the decoder selects the default slave that is a dummy slave that typically returns an error to the master.

An example memory map shows how the memory map can be broken down. The left memory breakdown shows the entire range available to the bus master in the example system with the presence of the three slave devices and the debug region added into the address range `0x3F50_0000` to `0x3F50_FFFF`. When an unspecified region is accessed, the decoder selects the Default Slave for the system bus. The right memory map shows the breakdown of the debug memory region which, although limited on system accesses, extends from `0x0000_0000` to `0xFFFF_FFFF`. The system accesses are only able to address in the region of `0x0000_0000` to `0x0000_0FFF` because of the addressing restriction in the system memory bus that is limited to `0x3F50_0000` to `0x3F50_FFFF`, and the unavailable address bits. The address[30:16] in the *Example memory system for access to debug components* on page 4-7 being tied to zero. External debug agents can also access this resultant region. However, because the ROM table is indicated as being located at `0x8000_0000`, external tools access the debug components through the lock-bypass alias **PADDRDBG[31]** equal to `1'b1` as all offsets point tools to components within this region. With reference to the example memory map, in the example system, the system address decoder decodes the following select lines for the system bus slaves:

```
Select_Slave_1         = (Address[31:29] == 3'b000)
Select_Slave_2         = (Address[31:30] == 2'b01)
Select_Slave_3         = ((Address[31:28] == 4'b1000)|
                          (Address[31:28] == 4'b1110))
Select_Debug_Interface = (Address[31:16] == 16'h3F50)
Select_Default_Slave   = !(Select_Slave_1        |
                           Select_Slave_2        |
                           Select_Slave_3        |
                           Select_Debug_Interface)
```

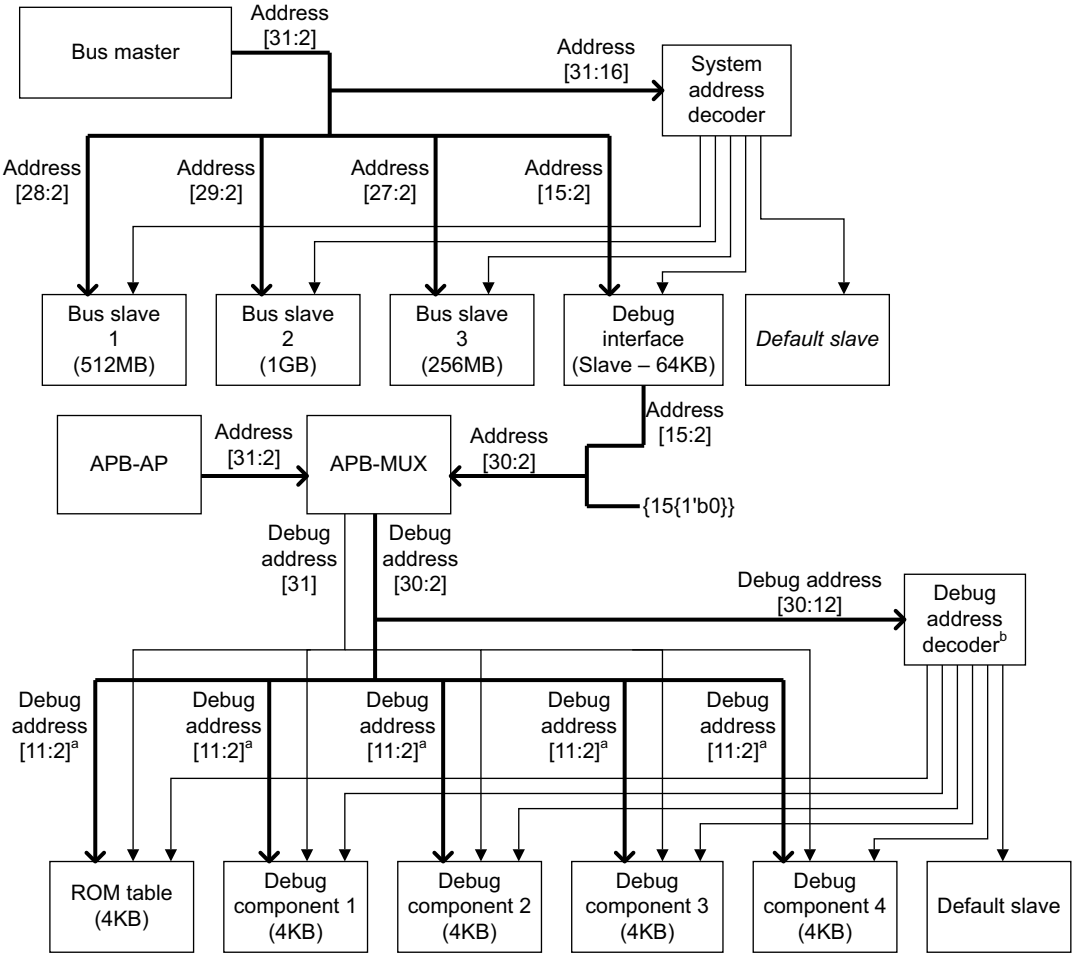
The debug address decoder decodes accordingly:

```
Select_ROM_Table     = (Debug_Address[30:16] == 15'h0000)
Select_Debug_Comp_1  = (Debug_Address[30:16] == 15'h0001)
Select_Debug_Comp_2  = (Debug_Address[30:16] == 15'h0002)
Select_Debug_Comp_3  = (Debug_Address[30:16] == 15'h0003)
Select_Debug_Comp_4  = (Debug_Address[30:16] == 15'h0004)
```


Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

```
Select_Default_Slave = !(Select_ROM_Table    |
                         Select_Debug_Comp_1 |
                         Select_Debug_Comp_2 |
                         Select_Debug_Comp_3 |
                         Select_Debug_Comp_4 )
```

Figure 4-2 shows an example memory system for the access to debug components.



[a] Although debug components accept Debug address[11:2] for local selection of internal registers, they can also accept Debug Address[31] to create two memory views of components, one only accessible by external tools through APB-AP that can set Debug Address[31]

[b] The address decoding for the ROM table and the ROM table itself, is provided as part of the DAP. Decoding for any user-specified debug components must be performed in a separate address decoder and those used addresses placed in the ROM table.

Figure 4-2 Example memory system for access to debug components

4.3.5 Example memory map

Figure 4-3 on page 4-8 shows an example memory map for the access to debug components.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

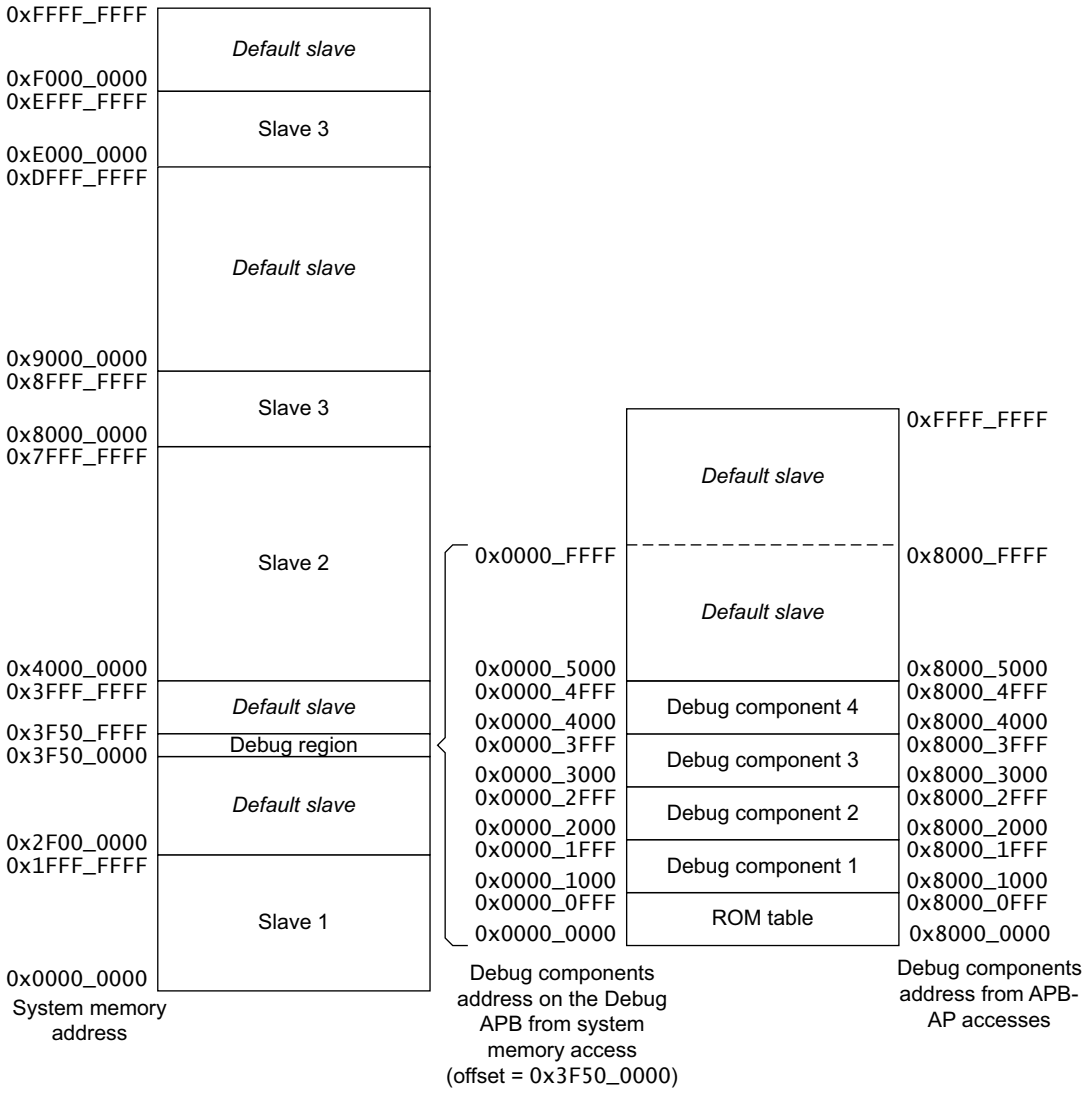


Figure 4-3 Example memory map for access to debug components

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.4 Mixed legacy and DAP debug

It is possible to have some of the advantages of CoreSight Technology with the ability to directly access system busses, but without having to alter tool requirements for JTAG access to existing cores. This section describes:

- *Mixing JTAG devices with JTAG-DP*
- *Mixing JTAG devices with SWJ-DP*.

4.4.1 Mixing JTAG devices with JTAG-DP

Existing tools expect JTAG controllers to appear on the main JTAG chain. This method reduces the requirement to modify existing tools, but has the disadvantage that it loses the ability to independently remove power to TAP controllers without stopping the operation of any other TAP controller on the main chain, in this case the DAP. This method of linking devices in series is not recommended for TAP controllers implemented within IEM boundaries that might power-down.

Figure 4-4 shows a core and DAP connected in parallel with JTAG.

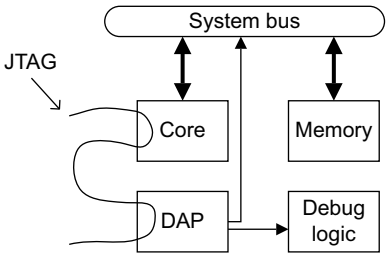


Figure 4-4 JTAG core connected in parallel with DAP

4.4.2 Mixing JTAG devices with SWJ-DP

It is possible to enable direct access to existing TAPs and gain the pin-count advantage of SWD when only debug access is required by disabling access to the TAPs when switching to SWD. The SWJ-DP component of the DAP provides additional outputs that indicate the mode of operation that is in force, JTAG or SWD, to enable a debugger to change between them without causing the TAPs to enter into UNPREDICTABLE states.

Figure 4-5 on page 4-10 shows an example JTAG connection that supports multiple TAP controllers that describes how to connect an additional TAP with the DAP enabling either serial access when using JTAG, or access to the DAP when using the reduced pin-count mode of SWD.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

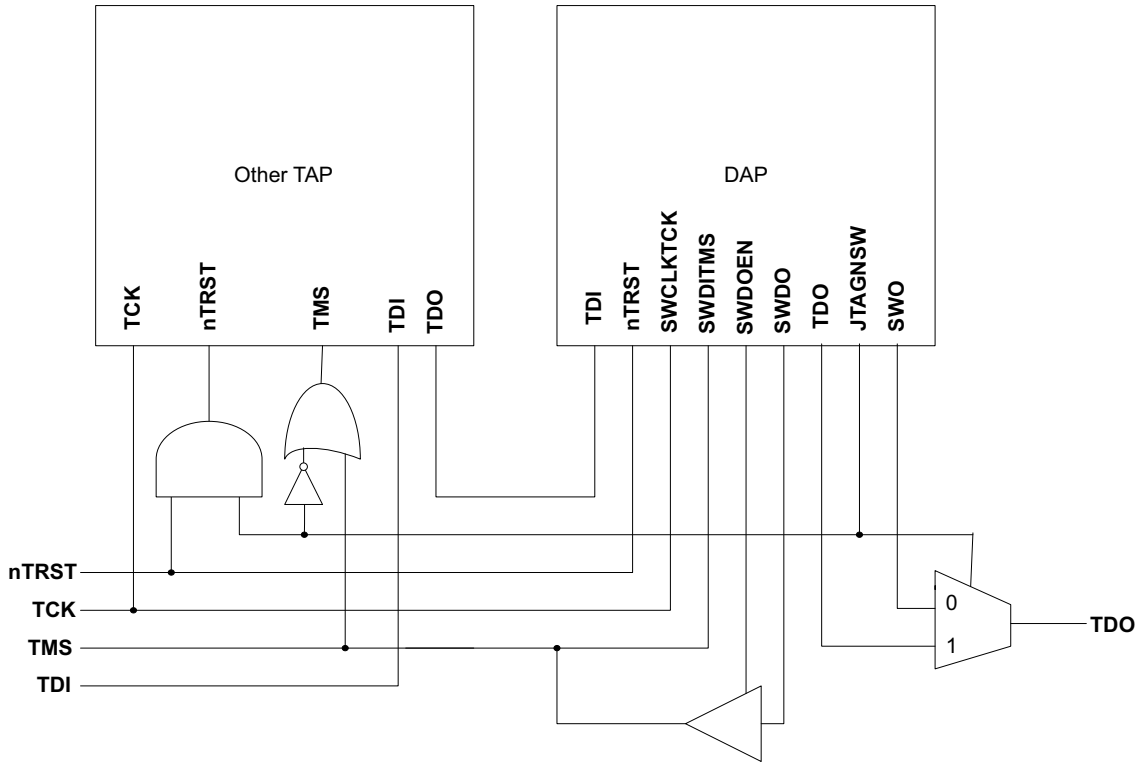


Figure 4-5 An example JTAG connection

Note

You must fit an external tri-state buffer on TMS to enable output of data when operating in SWD mode and a multiplexor on TDO for re-use with SWO. In Figure 4-5, when in SWD mode, the other JTAG TAP state machine is held in reset through the assertion of **nTRST** and clamping of TMS HIGH. In some devices, this can cause unexpected consequences so it is strongly recommended that you read the documentation for the TAP as to a safe method of disabling the device.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.5 Debug activity across the chip

This section describes:

- *Direct links* on page 4-12
- *Linking with an ECT* on page 4-12.

When you debug a single device, such as a processor in isolation, all the influences on that device are self-contained. In more complex systems, interactions occur between several devices, either several processors or single processors with several peripherals. These more complex systems require the ability to alter debug behavior depending on the nature of the interactions.

For example, when two processors interact with each other, it might be useful to maintain synchronization with both processors, and stop them together or halt one when the other reaches a predetermined point.

For activity in separate devices to link together, there must be some form of interconnection. Figure 4-6 shows an example interaction between processors.

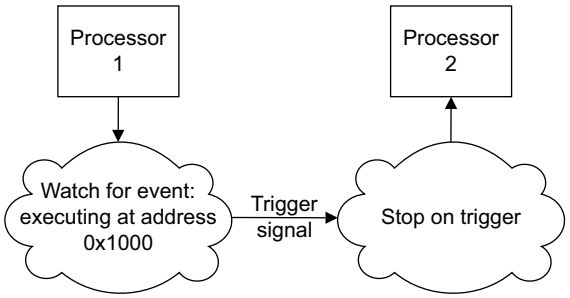


Figure 4-6 Processor interaction

The following definitions provide terms that describe system interactions:

Event An event is the expression that the system waits for to be true. When the event occurs, it generates a trigger each time that the expression is true. For example, in a simple event, the system waits for a signal to be HIGH. A complex event can result from a series of simple events, for example a primary event when an instruction executes and a secondary event that waits for the primary event to occur ten times.

Triggers A trigger is an activity event that originates in one device but is used by other devices, or the same device. Triggers can range from an output pin toggling, for example to indicate that a buffer is full, to the result of an operation on complex counters and comparisons configured within a device, for example pulse on execution of code line 1000. It is possible to cascade trigger events and combine triggers within some devices to generate complex events that result in a single trigger.

Signals of interest

For any system, only certain signals are useful for debug. These are known as signals of interest. Any one device can have a large number of inputs and outputs, but not all of them require trigger connections. Signals of interest might indicate the state of a control program flow. For example:

- interrupt inputs might make suitable inputs for triggers, and the stimulus attached to interrupts might be useful triggers to other devices
- the results of internal comparators or performance counters presented on top-level outputs can be trigger events on other connected devices, or used as a feedback loop to the same device.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Figure 4-7 shows possible signals of interest for a core and interrupt unit.

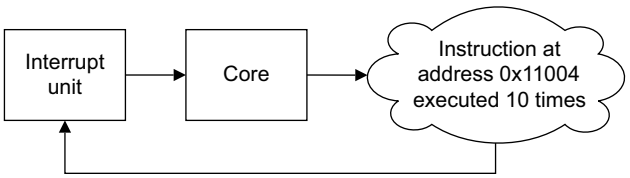


Figure 4-7 Signals of interest

Because of the variety and complexity of devices possible within a single SoC it is impossible to give an exhaustive list of trigger connections.

4.5.1 Direct links

A solution to linking together activity from the sections of a SoC, is to wire device inputs and outputs together directly. The primary advantage is that this requires minimal extra logic. You must take care when you link up arbitrary ports.

Consider the following when you use direct links to coordinate activity across a chip:

- ensure that connections cross clock domains safely and that no meta-stability occurs
- choose the correct connections at system design time because you cannot alter connections after implementation, that is, the wires are inflexible and require careful selection
- perform enough integration testing to ensure that the connections between devices are correct
- where necessary, include shaping logic on triggers, for example to convert edge-based output events into logic-level stimulus ports.

Note

Because of the variety of options for direct links, tools might not support the direct links you choose.

4.5.2 Linking with an ECT

The ECT is the structure that enables the correlation of triggers from the various parts of the SoC. It is composed of one or more CTIs that act as the primary matrix between the trigger inputs and outputs and the channel structure that propagates activity to other areas of the SoC.

When you require more than two CTIs, you can link them with CTMs that permit safe linking of channels to and from CTIs.

The CTI and CTM enable you to:

- produce a scalable system
- build a subsystem and provide connections for communication with other subsystems that you intend to design and implement at a later time.

Exporting a channel interface out of your subsystem provides a standard interface for more system-wide communication of debug events.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Tools can identify the programmable interfaces of the CTIs and then flexibly alter the connections of trigger sources and destinations over the various channels to adapt the propagated activity for the required purpose. You can isolate individual CTIs to stop the transmission of activity over the CTM to other CTIs.

Each CTI provides a set number of trigger inputs and outputs on the following styles of sampling:

- handshaking for asynchronous trigger signals
- edge-based events for synchronous triggers.

You can use these sampling schemes to convert to and from other styles of trigger signals, for example:

Conditioned Keeps the output active for one clock cycle after an acknowledgement is received.

Level/pulse Used when the trigger destination is level sensitive, the signal is active for only one clock cycle.

Sticky Keeps the output active until a corresponding clear register clears it.

NoAck Used when the output follows the input trigger and does not require an acknowledgement.

Software programmable

Used by the application to trigger events under software control. This enables a debugger to force an event using the configuration registers.

Note

Because the ECT safely crosses asynchronous clock domains, it is possible for multi-shot events that occur close to each other to be masked within the cross-trigger structure and only result in a single pulse on a trigger output. You must only use the ECT with single shot events, or multi-shot where safe, where time delay characteristics are not important.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.6 Typical trigger signals

This section lists components that have inputs and outputs suitable for use as triggers across a chip during debug. For specific CTI connection information, see the *appropriate CoreSight Design Kit Integration Manual*. This section describes:

- *CPU connections*
- *ETM connections*
- *Other trace source connections* on page 4-15
- *TPIU and ETB connections* on page 4-15.

4.6.1 CPU connections

Table 4-1 lists the inputs and outputs for CPUs.

Table 4-1 CPU connections

Trigger connection	ARM10 / ARM11	ARM7 / ARM9
Outputs	**DBGACK** **!nPMUIRQ**,	**DBGACK** **DBGRQI** **RANGEOUT[0]** / **DBGRNG[0]** **RANGEOUT[1]** / **DBGRNG[1]**
Output acknowledgements	-	-
Inputs	**EDBGRQ** **!nIRQ**	**(E)DBGRQ** **!nIRQ** **DBGEXT[0]** / **EXTERN0** **DBGEXT[1]** / **EXTERN1**
Input acknowledgements	-	-

4.6.2 ETM connections

Table 4-2 lists the inputs and outputs for ETMs.

Table 4-2 ETM connections

Trigger connection	CoreSight ETM	Pre-CoreSight ETM / ETM Single
Outputs	**EXTOUT[1:0]** **TRIGOUT**	**EXTOUT[1:0]**
Output acknowledgements	**EXTOUTACK[1:0]** **TRIGOUTACK**	**EXTOUTACK[1:0]**
Inputs	**EXTIN[3:0]**	**EXTIN[3:0]**
Input acknowledgements	**EXTINACK[3:0]**	**EXTINACK[3:0]**

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

4.6.3 Other trace source connections

Table 4-3 lists the inputs and outputs for HTM, ITM, and STM.

Table 4-3 HTM, ITM, and STM connections

Trigger connection	HTM connection	ITM connection	STM connection
Outputs	**HTMEXTOUT[1:0]** **HTMTRIGGER**	**TRIGOUT**	**TRIGOUTSPTE** **TRIGOUTSW** **TRIGOUTHETE** **ASYNCOUT**
Output acknowledgements	**HTMTRIGGERACK**	**TRIGOUTACK**	-
Inputs	**HTMEXTIN[1:0]** **HTMTRACEDISABLE**	-	**2x HWEVENTS** **2x ~HWEVENTS**
Input acknowledgements	-	-	-

4.6.4 TPIU and ETB connections

Table 4-4 lists the inputs and outputs for the TPIU, ETB, and TMC.

Table 4-4 TPIU, ETB, and TMC connections

Trigger connection	ETB connection	TPIU connection	TPIU Lite	TMC connection
Outputs	**FULL** **ACQCOMP**	-	-	**FULL** **ACQCOMP**
Output acknowledgements	-	-	-	-
Inputs	**FLUSHIN** **TRIGIN**	**FLUSHIN** **TRIGIN**	- **TRIGIN**	**TRIGIN** **FLUSHIN**
Input acknowledgements	**FLUSHINACK** **TRIGINACK**	**FLUSHINACK** **TRIGINACK**	**TRIGINACK**	-

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 5
Trace Capture

This chapter describes the performance requirements, bandwidth implications and trace generation capabilities for your SoC. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

5.1 About trace capture

The trace that CoreSight trace sources generate must be captured by one or more *Trace Capture Devices* (TCDs). The following common forms of TCD exist:

- on-chip trace buffer
- off-chip Logic Analyzer
- off-chip low-cost TPA.

Logic Analyzers are expensive and are less well supported by development tools, but can often capture trace at higher speeds than is possible with a TPA. Most developers capture trace using a TPA or on-chip trace buffer.

The CoreSight ETB and ETR is an ATB slaves and connects to the CoreSight system directly to enable capture of trace data on-chip. A TPA or Logic Analyzer must connect to the pins of a trace port, that is driven by a TPIU.

Most systems implement either one ETB or one TPIU. However, it is possible to implement multiple trace sink components using a CoreSight Replicator. See *Systems with an ETB and a TPIU* on page 5-5.

Figure 5-1 shows a system that implements an ETB and a TPIU connected to a TPA.

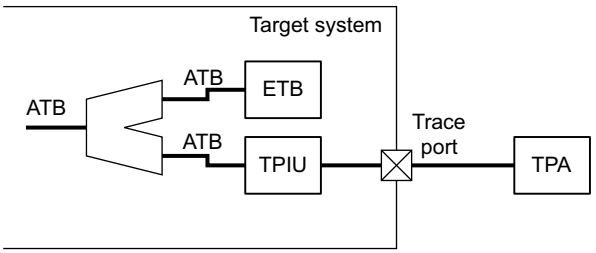


Figure 5-1 Example system with ETB and TPIU

5.1.1 Operation of a TCD

A TCD has at its centre a large circular buffer. Trace is written into this buffer as it is generated. Trace capture does not stop when the buffer becomes full, but instead overwrites old trace.

A TCD is sensitive to two special signals, that the ETB or TPIU generate:

- trigger
- trace disabled.

A TPIU indicates these signals to a TPA as follows:

- Using the optional **TRACECTL** pin. This is the easiest way for a TPA to detect this information.
- Using the CoreSight Formatter Protocol. This requires a TPA that can extract this information from the formatter protocol, and results in a trace port that is one pin smaller. For more information on the formatter protocol, see *Overhead of formatter protocol* on page 5-7 and the *CoreSight Architecture Specification*.

Trigger

The trigger is an input to the trace sink, connected to a CTI. If there is more than one trace sink, each can receive a different condition as its trigger. Most trace sources, for example an ETM or HTM, can output a trigger signal. Usually, the CTIs are configured to send a trigger to all trace sinks when any trace source outputs its trigger signal.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

When a trigger is detected, the TCD counts a programmable number of trace records before it stops trace capture. After this point, it ignores any more trace. By setting the appropriate number of programmable trace records, you can select a window of trace to capture around the trigger condition. Figure 5-2 shows this context.

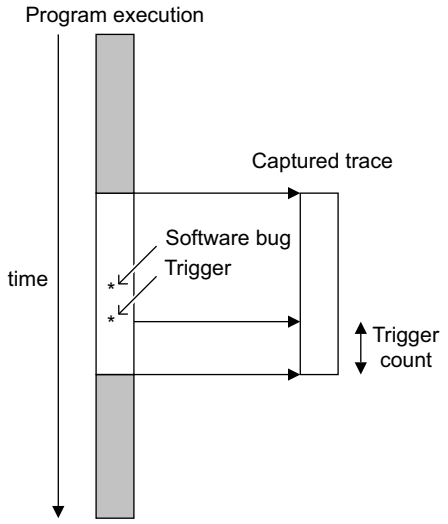


Figure 5-2 Use of the trigger to set a trace window

You can configure the trigger to output when the system detects a bug. The window of trace indicates the behavior of the system before and after the bug occurred.

You can use the trigger count in a number of ways:

- Set the trigger count to a small value. This gives a window of trace mostly before the trigger occurred, capturing the software bug under investigation.
- Set the trigger count to a value slightly smaller than the size of the buffer. This gives a window of trace mostly after the trigger occurred.
- Set the trigger count to roughly half the size of the buffer. This gives a window of trace before and after the trigger occurred.

When trace capture has stopped, the development tools download the trace from the TCD.

Trace disabled

Trace disabled indicates to the TCD that there is no trace to capture. It ensures that the values of the trace port pins are only captured when trace data is available.

Usually the ETB, ETR or TPIU waits until there is sufficient trace to use all the pins of the trace port before any trace is captured in the on-chip memory or output over the Trace Port. For example, if only one byte of trace is available in a system that implements a 16-bit Trace Port, no trace is output until a second byte of trace is available.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

5.2 Designing your trace system

This section describes issues to consider when you design a trace system:

- *Differences between on-chip and off-chip storage*
- *Calculating the number of trace port pins* on page 5-5
- *Calculating the size of on-chip RAM required* on page 5-8
- *ATB bandwidth* on page 5-8
- *Using additional buffers in trace systems* on page 5-9.

5.2.1 Differences between on-chip and off-chip storage

To decide when to implement an ETB or ETR for on-chip storage, or a TPIU for off-chip storage in a TPA, you must consider the following:

- You can capture much more trace in a TPA than in an ETB or ETR. This has the following advantages:
  - — You require fewer trace runs to identify the cause of a bug. The cause of the bug is less likely to have been overwritten.
  - — The trace requires less filtering. You can increase the effective size of the buffer considerably by using filtering resources to trace only important events, for example when the processor executes a particular function. This can lead to the loss of important trace because the system accidentally filters it out. Filtering resources also take time to set up, slowing down the development process.
  - — You can perform more accurate profiling. Code profiling requires trace over a long time to be accurate.
- An ETB and ETR can capture trace at a much higher speed. This enables the system to capture far more detailed trace over short periods. For example, a very high speed system might not be able to dedicate enough pins to capture full data trace from a processor off-chip, but might be able to capture the same trace on-chip in an ETB or ETR.
- An ETB or ETR do not require any trace pins. Your system might require a large number of pins if:
  - — It must trace a large amount of data simultaneously. For example, if you want to provide full data trace of several cores at the same time, your system requires a large number of pins.
  - — The trace port speed is much less than the speed of the components being traced. For example, in a system with a 250MHz trace port clock, four times as many pins are required to trace a processor running at 1GHz than are required to trace the same processor running at 250MHz.
- On-chip trace storage requires considerable silicon area. The size of the on-chip RAM can be substantial, depending on the maximum size of trace window that you have to support You can offset this with the reduction in I/O pads that also use silicon area or by reusing the memory for run-time usage when trace is not required.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Systems with an ETB and a TPIU

Sometimes it is advantageous to implement a TPIU with its Trace Port, and an on-chip buffer such as an ETB, in the same chip. See Figure 5-1 on page 5-2. Usually the Trace Port in such devices is only capable of a small amount of trace, for example, it might only have sufficient bandwidth for instruction trace. This permits you to use the trace in different ways:

- Use the on-chip buffer when you require full trace over a short period. This is most useful when debugging the behavior of a well-defined section of software, when you can use filtering and the time between an error in the code and the detection of the bug by the trigger condition is small.
- Use the TPIU and TPA when you require trace over a long period. For example, when you cannot use filtering, or the time between an error in the code and the detection of the bug by the trigger condition is large.
- Use the TPIU and TPA when you derive profiling information from the trace. This requires a large amount of information and is usually concerned only with instruction trace.

In systems where the same ATB feeds both the on-chip trace sink and TPIU, it is not recommended that you enable both devices at the same time. If you do this, the device with the higher potential bandwidth receives trace at the same rate as the slower one. For example, a 32-bit ETB can only have a quarter of its maximum bandwidth as a similarly clocked TPIU if the TPIU is configured to only use 8-bits of the Trace Port and is operating on the same trace data stream. This is because a replicator stalls its input if either output stalls.

5.2.2 Calculating the number of trace port pins

The number of trace port pins that your system requires depends on:

- The number of trace sources that trace at the same time.
- The number of bits of trace per cycle that each trace source generates, averaged over the size of the FIFO of that trace source. Figure 5-3 on page 5-6 shows how a larger FIFO can reduce the trace bandwidth by smoothing over bursts of trace. In this example, a FIFO of at least 21 bytes requires only a 4-bit trace port to output the trace without overflow, but a FIFO of only 13 bytes requires an 8-bit trace port.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

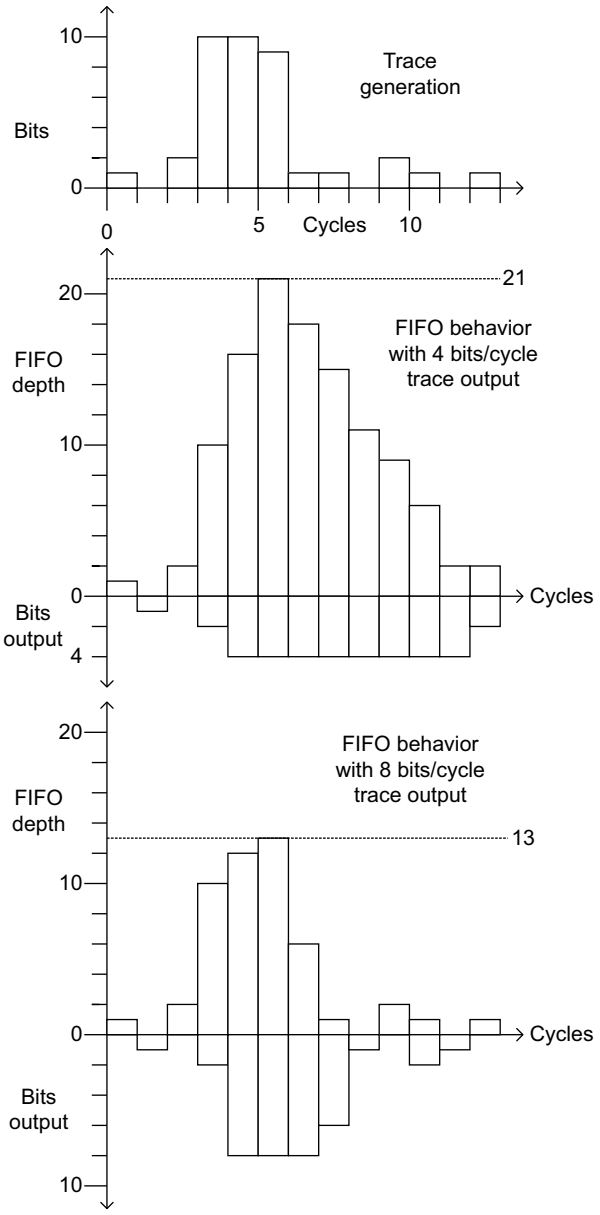


Figure 5-3 Effect of FIFO size on required trace bandwidth

- The ratio between the clock speed of the trace generation of each trace source and the speed of the trace port pins. A 250MHz trace port requires half as many trace port data pins as a 125MHz trace port to achieve the same trace bandwidth.
- The acceptable number of overflows. When you exceed the trace bandwidth, the trace source FIFOs can overflow. If you do not require to trace reliably through bursts of trace, then you can choose a lower trace port bandwidth.
- When several trace sources trace at the same time, if these sources are likely to produce simultaneous trace bursts. Normally, you must add up the bandwidth requirements for each trace source to determine the total bandwidth requirement. However, if each source only occasionally requires high bandwidth to trace a burst, then it is unlikely that the sources require this high bandwidth at the same time. The total bandwidth is therefore less than the sum of the individual bandwidths.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- If there is any additional buffering in the trace system permitting peaks in trace activity to be more averaged over time. See *Using additional buffers in trace systems* on page 5-9.

Bandwidth required by ETMs

The bandwidth that ETMs require depends on the clock speed of the processor and the average number of instructions that your application executes per cycle. It is recommended that, to estimate your requirements, you trace the software that you intend to trace in your SoC in a development environment. Contact ARM for more information about the bandwidth that ETMs require to trace popular benchmarks.

The bandwidth the ETM trace requires is greatly affected by enabling:

- cycle-accurate tracing
- data address tracing
- data value tracing.

Table 5-1 shows the typical difference in trace bandwidth between these levels of trace. You must calculate the average number of instructions per data transfer and the number of instructions per processor cycle for your application to determine the trace bandwidth you require.

Table 5-1 Effect of different tracing levels on ETM bandwidth requirements

Data trace level	Cycle-accurate mode	Average bits per instruction	Average bits per data transfer
Instruction plus data	No	4	40
Instruction only	No	1.2	N/A
Instruction plus data	Yes	8	40
Instruction only	Yes	6	N/A

You can support one level of tracing when an ETM is the only trace source using the trace port, and a lower level of tracing when other trace sources share the trace port. For example, you can provide sufficient trace bandwidth so that:

- an ETM can perform full data tracing when it is the only trace source enabled
- the same ETM can only reliably perform instruction tracing when combined with other trace sources.

Overhead of formatter protocol

The TPIU, ETB, ETR and ETF implement a formatter protocol to enable multiple trace protocols to share the same trace port. The formatter protocol wraps the trace protocols from the trace sources in the system, and adds Trigger and Trace Disabled information. For a description of this, see *Operation of a TCD* on page 5-2. The trigger information indicates when the TPIU or ETB received a trigger, and is sometimes useful when analyzing the trace.

You can configure the ETB to bypass the formatter if both the following conditions apply:

- only one trace source is enabled
- you do not have to record when in the trace the ETB received a trigger.

You can configure the TPIU to bypass the formatter if all of the following conditions apply:

- only one trace source is enabled
- you do not have to record when in the trace the TPIU received a trigger

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- your system implements the **TRACECTL** trace port pin
- the TPA is able to understand the information on **TRACECTL**.

You can program the ETB, ETR and ETF configurations of the TMC to bypass the formatter if the following conditions apply:

- only one trace source is enabled
- you do not have to record when a trigger is received
- circular buffer mode is selected
- in ETF configuration only, the buffer drain mode is not enabled.

The formatter adds:

- A fixed overhead of 6% to the trace bandwidth.
- An overhead of one byte every time the trace bus switches between trace sources. To minimize the switching cost configure the trace funnels appropriately. For more information on this, see *Arbitration* on page 5-13.

5.2.3 Calculating the size of on-chip RAM required

Consider the following factors when you decide how much memory to use for an ETB or ETR:

- The length of time the trace history must cover.
- The trace bandwidth of the trace capture. For information on factors that affect trace bandwidth, see *Calculating the number of trace port pins* on page 5-5.
- The complexity and novelty of the system in question, including software.
- If it is practical to perform filtering, that increases the effective length of time the history covers.
- The trade-off of system cost to design time.
- If your system is a low volume development chip or a high volume production part.

Some of these factors are highly system-dependent. Most designs use between 4KB and 16KB of RAM, but in some circumstances, smaller or larger sizes might be appropriate. For example, larger sizes might be appropriate for a development chip with a large number of trace sources.

5.2.4 ATB bandwidth

The bandwidth of an ATB bus is usually sufficient to support full tracing of trace sources to a high-bandwidth trace sink without overflows. However, you can exceed the bandwidth of an ATB bus if:

- Many trace sources generate trace at the same time.
- The ATB clock speed is much lower than the clock speed of the device being traced. For example, with a CPU running at 1GHz but the ATB operating at 250MHz, the trace source only has a quarter of the bandwidth available to output date from its internal buffer.

To determine whether this is an issue, follow the guidelines in *Calculating the number of trace port pins* on page 5-5. The ATB bus is equivalent to a 32 bit trace port.

You can prevent you system exceeding the ATB bandwidth by:

- Increasing the clock speed of the ATB bus.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

- Implementing more than one local trace sink so that the same ATB bus does not have to carry the trace from all trace sources. Figure 5-4 shows an example system that provides a dedicated ETB for each of two high-performance processors.

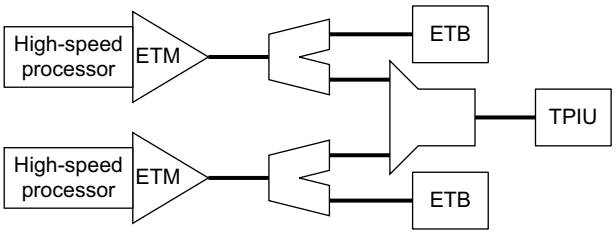


Figure 5-4 System with two ETBs

5.2.5 Using additional buffers in trace systems

Adding supplemental FIFOs within a trace system can help to alleviate narrowing of bandwidths across a system and reducing the risk of losing trace data. Although trace sources are designed with FIFOs to mitigate the large burst nature, there are still situations when the ATB bandwidth can limit the amount of information that can be traced.

- periods of heavy trace data generation from a trace source
- sustained coincident periods of trace data generation from multiple trace sources
- constrained size Trace Ports
- trace sinks with inconsistent bandwidths such as those interacting with a memory system for storage.

You can use a trace link component that has a large FIFO, such as the ETF that uses an SRAM for data storage, to compensate for differing bandwidth specification of devices on the inputs and output of the link. The FIFO enables for averaging of the ATB activity over larger periods of time than would be practical within a trace source, either by sharing the resources of a large buffer or because the conditions for it are not applicable in all ASICs where the trace source is implemented.

The following areas exist where an ETF can be fitted to reduce the effects that can cause loss of data:

- after a high-bandwidth trace source, such as an ETM, to reduce the effects of arbitration with other trace data streams
- after a low-bandwidth trace source that is being used more frequently to avoid the risk of FIFO overflow during peak periods of activity
- additionally, after devices such as the STM that can cause back-pressure on the memory system during periods of high sustained use
- before a trace sink with a guaranteed average bandwidth that is lower than the peak bandwidth of the ATB, such as a TPIU with a Trace Port less than the size of the ATB interconnect or operating at a frequency of the ATB
- before a trace sink that stores trace data in bursts of data, such as an ETR when it is arbitrating with other bus systems on a high-bandwidth interconnect.

Figure 5-5 on page 5-10 shows four potential locations for fitting an ETF in a trace system. In such a system, not all the ETFs have to be fitted depending on the constraints of the ATB interfaces for the trace sinks and trace sources.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

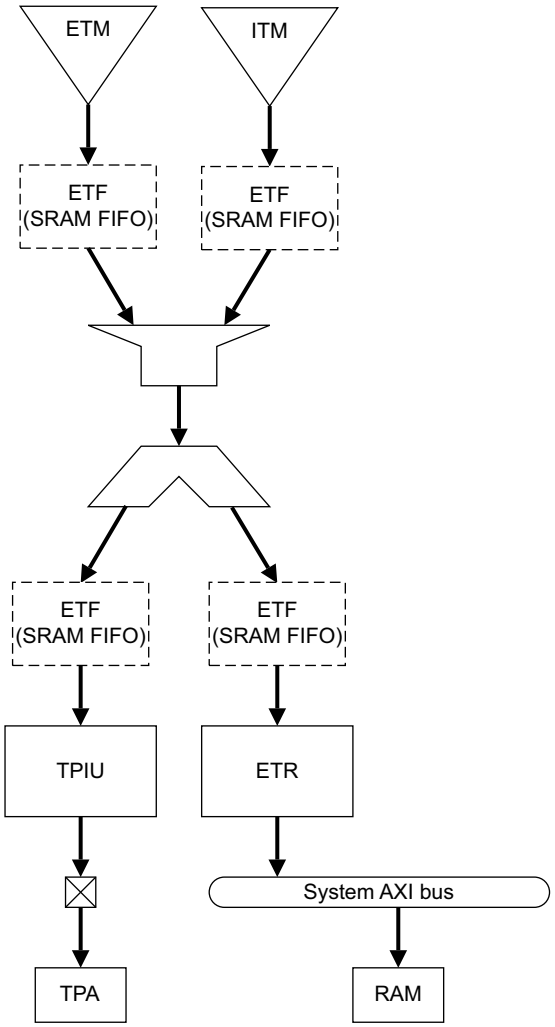


Figure 5-5 Using additional buffers for trace system

Note

An ETF with a large amount of SRAM are not going to solve all situations where FIFO overflows occur. Restrictions in bandwidth when crossing asynchronous boundaries and widths of ATB can alter the characteristics of the ATB before and after the change in bandwidth. For example, if an ETF is fitted after an ETM, to absorb large quantities of trace data produced during data-value trace, if the ATB interface is running significantly slower that the of the core, then the trace source is still liable to overflow as the bandwidth from the output of the ETM's internal FIFO is restricted between the ETM and ETF to enable the trace data to drain from the FIFO before more data is internally created.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

5.3 Using your system

This section describes what you must consider when you configure your trace system for trace capture:

- *Synchronization frequency*
- *Using the ETB for profiling* on page 5-12
- *Arbitration* on page 5-13.

For other design issues, see *Designing your trace system* on page 5-4.

5.3.1 Synchronization frequency

Most trace sources output synchronization information periodically during tracing, and often each time tracing restarts after a gap. If older trace has been overwritten, the system cannot decompress any trace before the first synchronization point that remains in the buffer.

A TPA captures a large amount of trace and many synchronization points, so only a small proportion of the trace is lost in this way. An ETB captures fewer synchronization points, so a larger proportion of the trace might be lost. Figure 5-6 shows this effect.

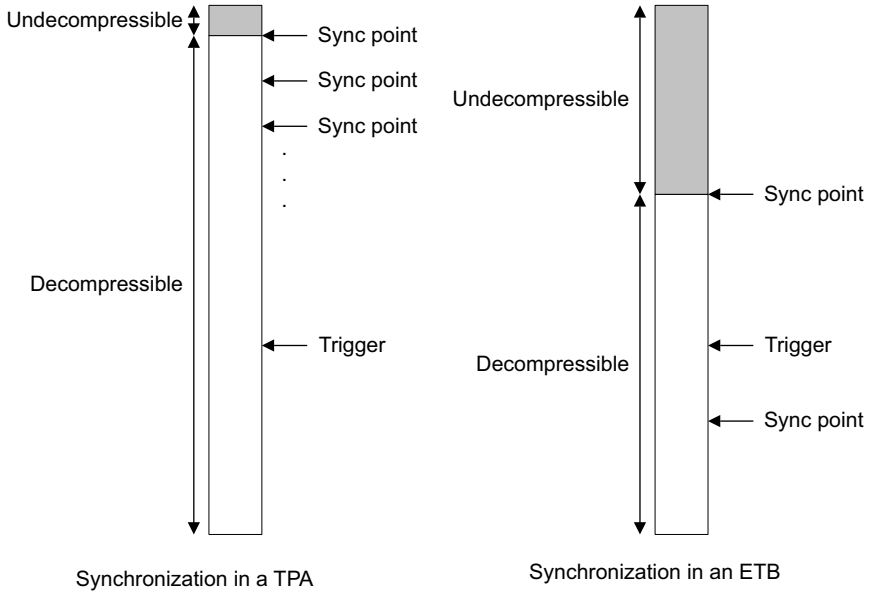


Figure 5-6 Effect of frequency compared with infrequent synchronization points

When you use an ETB, you must ensure that the synchronization frequency of trace sources is set to provide sufficient synchronization points. There must be enough points to make the amount of undecompressible trace small compared to the size of the ETB RAM. When you use a TPA, the size of the TPA buffer is usually large enough that the synchronization frequency is not significant.

Note

ETM9CS, ETM11CS, and HTM all have software-programmable synchronization periods.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Tracing more than one source

When they trace multiple sources, some trace sources can generate synchronization points much more frequently than others. If a trace source only uses a small portion of the bandwidth, then there might not be enough synchronization points for that trace source. You can program such trace sources to increase their synchronization frequencies to compensate for this.

5.3.2 Using the ETB for profiling

The ETB RAM cannot store enough trace to produce reliable profiling information. However, you can perform profiling with an ETB if the following apply:

- it is acceptable to periodically interrupt the execution of the software that you are profiling, increasing the time taken for it to run
- it is acceptable to install an additional interrupt handler
- it is acceptable for sections of trace to be missing periodically.

This method takes advantage of the **FULL** output from the ETB, that you must connect to a CTI for cross-triggering. You must also connect an interrupt request signal to one of the CTI outputs. Both signals might be on the same CTI, or they might be on different CTIs connected by a CTM.

Configure the system as follows:

1. Arrange an off-chip store to store the trace in.
2. Configure the CTI to cause the generation of an interrupt when the FULL signal occurs. This indicates that the ETB is full of trace.
3. Install an interrupt handler that is invoked when the CTI requests an interrupt. This handler must:
   - disable ETB trace capture
   - transfer the ETB RAM contents to an off-chip store
   - initialize and re-enable the ETB as before.
4. Configure the ETB to accept trace data and the trace sources to produce trace.

This drains the contents of the ETB whenever the ETB becomes full. Some trace is lost, because of latency between the output of the FULL signal and the invocation of the interrupt handler. To prevent the loss of any trace, you can configure the ETB to cause an interrupt early by causing the ETB to wrap around before it is full. To do this, configure the system as follows:

1. Arrange an off-chip store.
2. Configure the CTI to cause the generation of an interrupt when the **FULL** signal occurs.
3. Configure the CTI to output a trigger to the ETB when the **FULL** signal occurs.
4. Decide how many ETB entries (n) before the end of the buffer to reserve until after the generation of the **FULL** signal:
   - set the ETB write pointer to n
   - set the ETB trigger counter to slightly less than n.
5. Install an interrupt handler that reinitializes the ETB in this new way. The handler must place the first n entries in the ETB RAM after the remaining entries in the trace store.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

5.3.3 Arbitration

The CoreSight Trace Funnel supports the following arbitration options, that you can configure using development tools:

- You can set the relative priority of each trace source. You are recommended to set the priorities of trace sources as follows:
  - give priority to low-bandwidth trace sources over high-bandwidth trace sources
  - give priority to trace sources with small FIFOs or no FIFO, because stalling these trace sources is more likely to lead to the loss of trace data.
- You can set the maximum switching frequency. This ensures that, when multiple trace inputs output trace at the same time, the funnel does not switch priority between sources too quickly. This is important because the CoreSight Formatting Protocol adds one byte of overhead every time the funnel switches between trace sources. You are recommended to set this to a non-zero value.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Chapter 6
Implementation

This chapter describes implementation in CoreSight systems. It contains the following sections:

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.1 About implementation

To successfully implement a CoreSight system you must consider:

- interface timing
- power control and the provision of appropriate power and voltage domains
- components with low power modes
- clocks and clock domains, and resets
- control during debug.

For more information on implementation and integration of CoreSight Technology, see the *CoreSight Components Implementation Guide* and the applicable Integration Manual.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.2 Power control

CoreSight Technology enables improvements to energy efficiency through:

- more than one power domain in the SoC and support for a unique debug power domain to enable shutdown of CoreSight components using signal clamps
- support for *Intelligent Energy Management* (IEM)
- tool control of power up and down.

Implementation of these power control features is optional. During implementation, you can either turn off or improve the energy efficiency of debug logic in a system with CoreSight Technology. Your system might only support IEM in the ARM processor. You must decide during implementation whether it is necessary provide a separate power domain for the debug and trace logic in the SoC under development.

It is recommended that you implement CoreSight systems with a separate debug power and voltage domain:

- that is used by all CoreSight components and trace source ATB interfaces
- that can be powered-down so that debug and trace infrastructure is not active in in-service ASICs
- that can be powered-up through the DAP to enable tools access.

You must also provide:

- a common clock for all debug infrastructure access, so that all CoreSight Debug APB interfaces run on this common clock, synchronous to ATB
- a common reset for all debug and trace infrastructure, so that at power-on, all debug and trace infrastructure resets at the same time.

Power domain

Power domains refer to those areas of the SoC that can be completely powered-down independently of one another. A separate debug power domain is necessary to power-down the debug infrastructure, or power it up independently of the system or CPU. The SoC requires wire clamps between signals that join two independently-powered domains. These clamps hold signals at known static values between powered and unpowered domains. For production embedded systems, when the debugger does not require access to the system, the system can power-down the debug logic within the core and SoC to avoid unnecessary leakage associated with this logic.

Voltage domain

Voltage domains define the areas of an SoC running at a common voltage. You can implement *Dynamic Voltage Scaling* (DVS) throughout a voltage domain to permit the voltage to drop and still meet the stated performance requirement, therefore reducing power consumption. Communication between voltage domains requires level-shifting the voltage level signal clamping, *Level Shift and Clamp* (LSC). This guide assumes that only the CPU supports DVS, with debug and SoC domains supporting power-down only. In these scenarios, debug and SoC power domain interactions require clamps on the outputs from any block that can be powered-down, rather than LSCs.

It is normal that power and voltage domains are identical, because this significantly eases layout and placement of power grids within an SoC.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

This section describes:

- *Multiple power domains*
- *Intelligent Energy Management (IEM)*.

6.2.1 Multiple power domains

CoreSight components that cross a power domain boundary include placeholders for clamping signals from a powered-down domain.

CoreSight Technology supports the implementation of a separate power domain for debug and trace logic, referred to as the debug power domain.

6.2.2 Intelligent Energy Management (IEM)

The purpose of IEM technology is to provide a dynamic optimization between performance and power consumption.

To use IEM, you must implement your system with appropriate register slices and include it in a SoC that contains an *Intelligent Energy Controller* (IEC™). For more information, see the *Intelligent Energy Controller Technical Overview*.

CoreSight components that cross a boundary between an IEM-enabled system include:

- separate clock signals for each IEM domain
- asynchronous interfaces across IEM-enabled boundaries
- placeholders for level shifters and clamps on signals that cross an IEM-enabled boundary.

The same placeholders are used for signal clamps and level-shifters. For more information on IEM implementation, see *Power control enabled components* on page 6-7.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.3 Power domains and system design

You can implement common or several distinct power domains depending on the power requirements of the SoC. CoreSight technology supports the implementation of a separate debug power domain if you determine that it is necessary to shut down, or alternatively, IEM-enable, the debug and trace logic in the SoC.

Figure 6-1 shows a system that implements the following power domains:

Pcore The core domain.

PSoC The SoC domain.

Pon The always on domain.

In this example, the core is IEM enabled, the SoC infrastructure is not. LSCs exist between the core and SoC domains. There is a clamp between the SoC domain and the always on domain for the external SWJ interface to the DAP. The SWJ power must always be on.

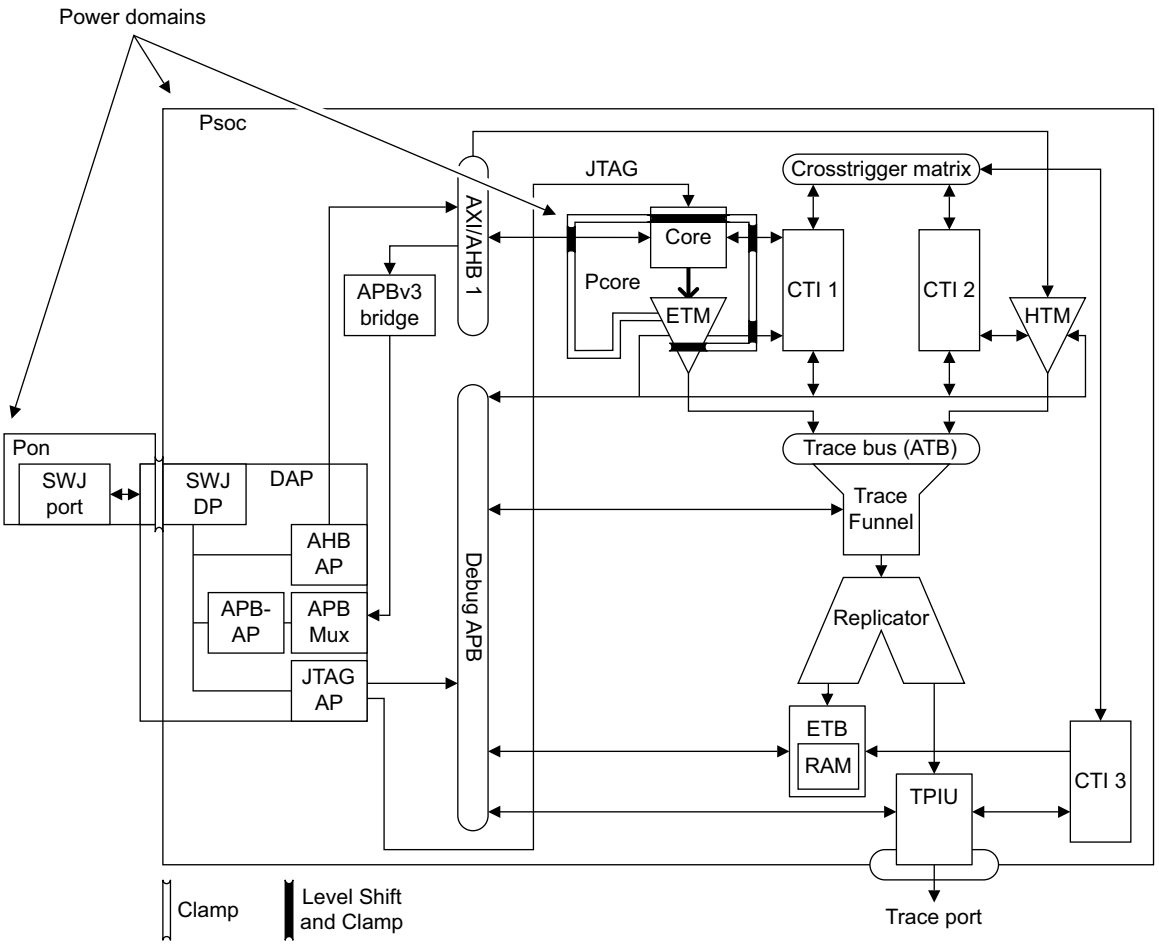


Figure 6-1 CoreSight system with no separate debug domains

Figure 6-2 on page 6-6 shows the same system enhanced to support a separate debug power domain, indicated by Pdbg. In this example, the core is IEM-enabled, and the separate debug power domain enables power-down of the entire debug and trace infrastructure independently of the rest of the system. This system must have clamps at component interfaces that cross between the debug and SoC power domains.

In Figure 6-2 on page 6-6, the core is IEM-enabled and the SoC infrastructure is not. The external SWJ interface to the DAP must be always powered-on.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

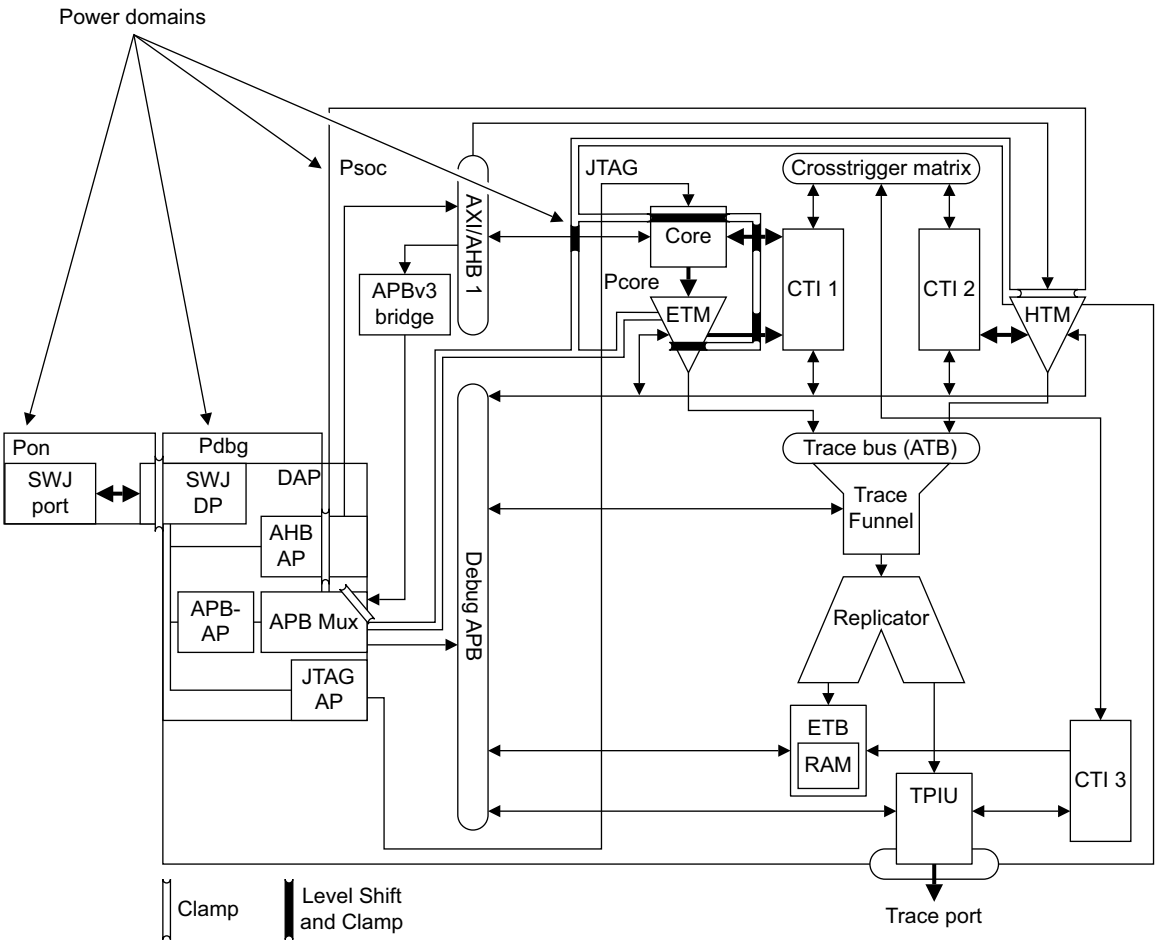


Figure 6-2 CoreSight system with a separate debug power domain

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.4 Power control enabled components

The following CoreSight components provide support for multiple power domains:

- DAP
- HTM
- ETM11CS and ETM9CS.

This section describes:

- *Debug Access Port (DAP)*
- *Power for Trace Sources, ETM11CS, ETM9CS, and HTM*.

For more information on IEM support for ETMs, see the *CoreSight ETM11 Technical Reference Manual* and the *CoreSight ETM9 Technical Reference Manual*.

6.4.1 Debug Access Port (DAP)

The DAP supports implementation over the following different power domains with asynchronous interfaces between:

- The external Debug Port interface and the DAP internal bus. There is an asynchronous interface between **TCK** (JTAG-DP), **SWCLK** (SW-DP), or **SWCLKTCK** (SWJ-DP), driven by tools, and **DAPCLK**, the DAP internal bus.
- The AHB-AP DAP internal bus and the AHB master port. There is an asynchronous interface between **DAPCLK**, the DAP internal bus, and **HCLK**, the AHB master port.
- The APB-Mux system slave interface and the APB-Mux master port. There is an asynchronous interface between **HCLK**, the system slave interface, and **DAPCLK**, the APB master port.

Each of these interfaces provides placeholders in the source code to enable the addition of signal clamps during implementation. You can enable the placeholders with Verilog `ifdef pragmas to use the clamping placeholders. The following Verilog `define must be present during compilation:

```
`define IEMSupport.
```

This `define instantiates a dummy Verilog module that describes the functionality of the clamp logic. This has the benefit that the component behaves appropriately for simulation and the implementation can include the correct library model with minimum overhead.

The same placeholders are used for signal clamps, for power-down, and level-shifters, if IEM support is implemented.

6.4.2 Power for Trace Sources, ETM11CS, ETM9CS, and HTM

A number of implementation options are possible for trace sources. The most power efficient solution for the organization of power domains, is also the most complex to implement for the trace sources. Therefore, this implementation is only likely in systems when a reduction in power consumption is critical.

The trace sources do not fit cleanly into a common power and voltage domain. Ideally, the CoreSight ETM and HTM exist within the debug power domain. This permits independent power-down of the trace components when there is no debug or trace in progress. However, both the ETM and HTM must be in the voltage domain of the component being traced. This is especially critical for the ETM because it must operate synchronously with its associated core

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

and at the same speed. It is not possible to have an IEM enabled boundary between the CPU and ETM because there must be no asynchronous boundary. Figure 6-3 shows a configuration for the ETM, and Figure 6-5 on page 6-9 shows a configuration for the HTM.

The following sections describes how to simplify the system design to address these complex implementation steps and remove the requirement for a separate voltage island:

- *ETM11CS, ETM9CS power*
- *HTM power* on page 6-9.

ETM11CS, ETM9CS power

Figure 6-3 shows an ETM connected with separate power and voltage domains.

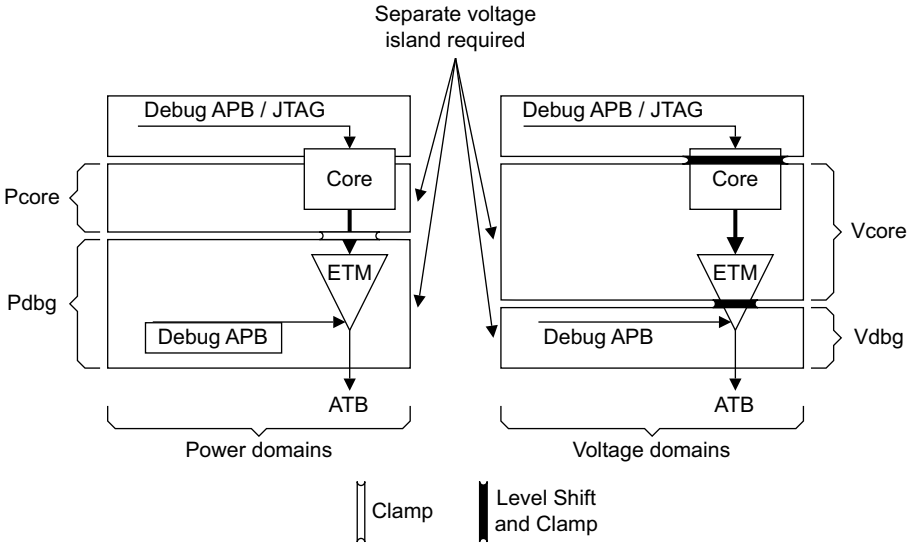


Figure 6-3 ETM power and voltage domains

To simplify system design, it is recommended that you put the ETM in the core power domain, so that it is powered-down when the core is powered-down. Figure 6-4 shows how the power and voltage boundaries are then unified, and the clamp between the core and ETM is not required.

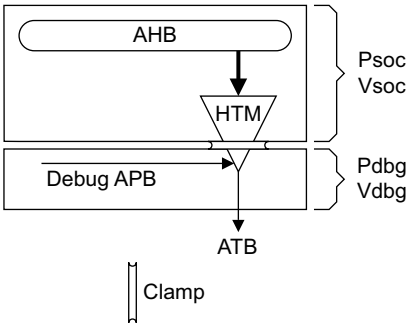


Figure 6-4 Unified power and voltage domains for ETM

For more information on the implementation of clamps in the ETM11CS, see the *CoreSight ETM 11 Technical Reference Manual* and for the ETM9CS the *CoreSight ETM 9 Technical Reference Manual*.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

HTM power

The HTM supports implementation over the following different power domains with asynchronous interfaces between:

- The debug APB interface and the AHB clocked trace generation logic. There is an asynchronous interface between **PCLKDBG** and **HCLK**.
- The AHB clocked trace generation logic and the ATB trace output. There is an asynchronous interface between **HCLK** and **ATCLK**.

PCLKDBG and **ATCLK** driven logic is synchronous, and both are common to the debug power domain. Signal clamps are implemented in the source code to enable the addition of signal clamps between the debug power domain and SoC power domain providing **HCLK**, during implementation.

To enable use of the clamping placeholders, the following Verilog `define must be present during compilation:

```
`define CSHTM_CLAMP_LOGIC.
```

This instantiates a dummy Verilog module that describes the functionality of the clamp logic. This has the benefit that the component behaves appropriately for simulation, and the implementation can include the correct library model with minimum overhead.

Figure 6-5 shows an HTM connected with separate power and voltage domains.

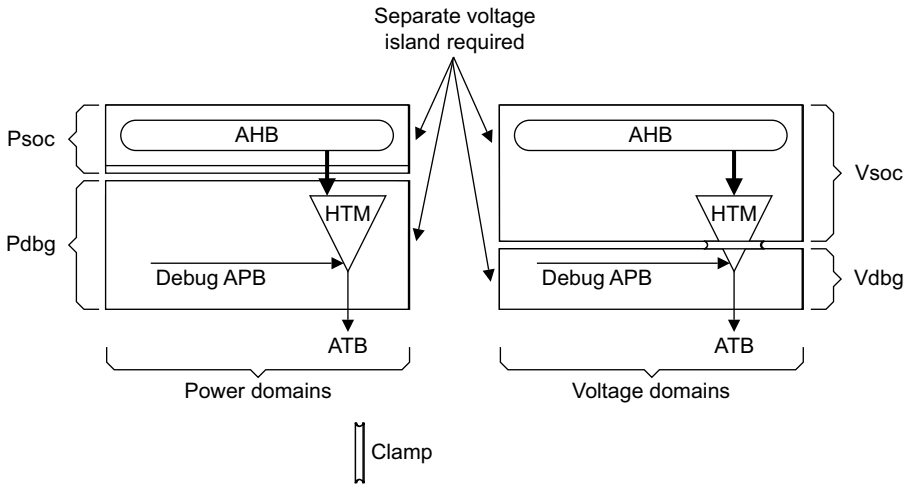


Figure 6-5 HTM power and voltage domains

To simplify system design, it is recommended that you put the HTM in the SoC power domain, if SoC and Debug are independent, so it is powered-down when the SoC domain is powered-down. Figure 6-6 on page 6-10 shows how the power and voltage boundaries are then unified.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

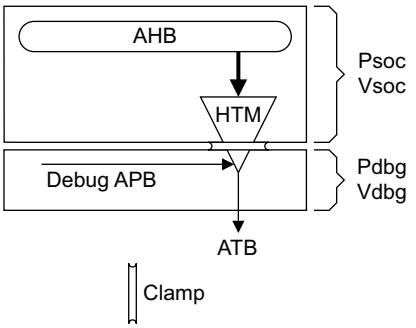


Figure 6-6 Unified power and voltage domains for HTM

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.5 Debug and system power up

The Debug Port in the DAP provides two pairs of power-on request signals to enable power-up of either the debug domain, or the complete system.

The Debug Port registers must be in an always powered-on domain to enable the system to make power-up requests to a system power controller:

- CTRL/STAT[29:28] provide **CDBGPWRUPACK** and **CDBGPWRUPREQ**. **CDBGPWRUPREQ** generates a debug domain power-up request to a power controller. The power controller must acknowledge the request with **CDBGPWRUPACK**.
- CRTL/STAT[31:30] provide **CSYSPWRUPACK** and **CSYSPWRUPREQ**. **CSYSPWRUPREQ** generates a complete system power-up request to a power controller. The power controller acknowledges the request with **CSYSPWRUPACK**. In this case, if multiple power domains exist, then they must all be powered-up.

In the majority of cases, it is expected that debuggers power on the complete SoC. If the issue for the debugger relates to energy management, the debugger might require power-up for only the debug domain. In this situation, you can design a system so that the power controller maps onto a bus segment that the DAP can access when only the debug domain is powered-on.

Note

The following apply to both system power-up and debug power-up requests and acknowledgements:

- **CxxxPWRUPREQ** must be asserted HIGH in the DP to initiate power-on.
- The power controller must power-up the corresponding power domains on receiving **CxxxPWRUPREQ** HIGH, and when it has done so **CxxxPWRUPACK** must be returned HIGH.
- Tools can only initiate a DAP internal transfer when both **CxxxPWRUPREQ** and **CxxxPWRUPACK** are HIGH for either pair of power control signals.
- When both **CxxxPWRUPREQ** and **CxxxPWRUPACK** are HIGH, the corresponding power domains are powered-on.
- The removal of a power-on request is initiated by **CxxxPWRUPREQ** being deasserted. The power controller returns **CxxxPWRUPACK** LOW when the power removal request is accepted.

The return of **CxxxPWRUPACK** LOW does not indicate that power has been removed. It indicates the request for power-down has been accepted.

Note

- It is strongly recommended that the debug power domain is powered-down on deassertion of the power-on request.
- All other power domains must gracefully power-down unless the system is operating in such a way that removal of power would affect its operation, that is, the system must stay powered-up if the power controller has other requests to maintain power.
- Before tools can make a new request for power on, **ACK** must be LOW indicating that a previous request for power-down has been accepted. This ensures that the handshaking mechanism is not violated.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Figure 6-7 shows the timing of the power signals. The DP only permits initiation of DAP transfers to an Access Port between time T2 and T3.

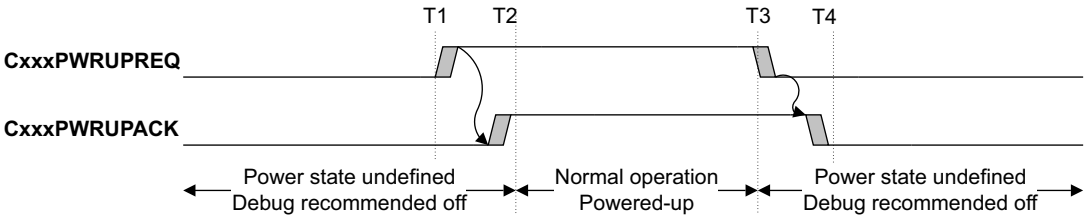


Figure 6-7 Power-up request and acknowledgement timing

Table 6-1 shows how the power-up request signals must be connected depending on the power configuration of your system.

Table 6-1 Power-up request and acknowledge signal connections

Power configuration	CDBGPWRUPACK/ CDBGPWRUPREQ	CSYSPWRUPACK/ CSYSPWRUPREQ
No power management Core, SoC, and debug are always on.	Connect **CDBGPWRUPACK** to **CDBGPWRUPREQ**.	Connect **CSYSPWRUPACK** to **CSYSPWRUPREQ**.
Core is IEM enabled, or can be shutdown. No SoC or debug power management.	Connect **CDBGPWRUPACK** and **CDBGPWRUPREQ** to system power controller. If **CDBGPWRUPREQ** is HIGH, ensure that all debug logic is powered up.	Connect **CSYSPWRUPACK** to **CSYSPWRUPREQ**.
Core is IEM enabled, or can be shutdown. SoC can be powered-off. No separate debug domain. All CoreSight infrastructure powered by SoC domain.	Connect **CDBGPWRUPACK** and **CDBGPWRUPREQ** to system power controller. If **CDBGPWRUPREQ** is HIGH, ensure that all debug logic is active.	Connect **CSYSPWRUPACK** and **CSYSPWRUPREQ** to system power controller. If **CSYSPWRUPREQ** is HIGH, ensure that SoC domain is powered-on.
Separate Core, SoC, and debug power domains. All can be powered-on or off independently.	Connect **CDBGPWRUPACK** and **CDBGPWRUPREQ** to system power controller. If **CDBGPWRUPREQ** is HIGH, ensure that SoC domain is powered on so that all debug logic is active.	Connect **CSYSPWRUPACK** and **CSYSPWRUPREQ** to system power controller. If **CSYSPWRUPREQ** is HIGH, ensure that the entire system, Core, SoC, and debug domain is powered-on.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.6 Clock domains

This section describes:

- *CoreSight system clock design*
- *Clocking in an power control enabled system* on page 6-14
- *CoreSight clocks and their inter-relationships* on page 6-14.

Table 6-2 lists the CoreSight Technology clocks

Table 6-2 CSDK clocks

Clock	Description
ATCLK	This is the *AMBA Trace Bus* (ATB) clock.
CSRTCK	Return Test Clock, target pacing signal.
CSTCK	This is the clock signal generated by the DAP (JTAG-AP) to drive JTAG interfaces other components.
CTICLK	This is the cross trigger interface clock. It can be synchronous or asynchronous to **CTMCLK**.
DAPCLK	This is the Debug Access Port (DAP) internal clock. It must be equivalent to **PCLKDBG**.
CTMCLK	This is the cross trigger matrix clock. It can be synchronous or asynchronous to **CTICLK**.
HCLK	This is the system-facing AHB clock used by the DAP (AHB-AP). It is asynchronous to **DAPCLK**.
PCLKDBG	This is the Debug APB clock. It must be synchronous, that is, equivalent to slower than **ATCLK**.
PCLKSYS	This is the system slave facing APB clock used by the DAP (APB-Mux). It can be asynchronous to **DAPCLK**.
SWCLKTCK	This is the SWJ-DP clock driven from the external debugger. It is asynchronous to **DAPCLK**.
TRACECLKIN	This is the Trace Port Interface Unit external trace clock input. It is asynchronous to **ATCLK**.

Note

CoreClk is external to CoreSight clocks, driving one or more ARM microprocessors, and is used by the ETM to correctly synchronize to the processor activity.

6.6.1 CoreSight system clock design

Requirements of the clock implementation are:

- **PCLKDBG** and **DAPCLK** must be equivalent
- **PCLKDBG** and **ATCLK** must be synchronous
- **PCLKDBG** must run synchronously at the same frequency, or run synchronously at a lower frequency to **ATCLK**.

PCLKDBG greater than **ATCLK** is unsupported.

It is expected that system-level access through the AHB-AP, running from **HCLK** and software access to the Debug APB through the DAP, running from **PCLKSYS**, are the same AMBA interconnect clock. If this is the case, **HCLK** and **PCLKSYS** are equivalent.

In summary:

PCLKDBG is equivalent to **DAPCLK**.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

PCLKDBG, **DAPCLK**, and **ATCLK** must be synchronous.

PCLKDBG is less than or equal to **ATCLK**.

HCLK and **PCLKSYS**, in a typical system are equivalent.

For system implementations where **PCLKDBG** is less than **ATCLK**, it is recommended that CoreSight components requiring both **ATCLK** and **PCLKDBG**:

- connect **ATCLK** to both **ATCLK** and **PCLKDBG** clock inputs on a component
- generate a clock enable term, derived from **ATCLK**, and connect this to **PCLKENDBG**.

For any clock domain where a clock enable is not required, connect the corresponding clock enable port HIGH.

6.6.2 Clocking in an power control enabled system

All boundaries between power domains and IEM boundaries are provided with an asynchronous boundary. Clocks must be connected as *CoreSight system clock design* on page 6-13 describes.

6.6.3 CoreSight clocks and their inter-relationships

Figure 6-8 shows the CoreSight clocks.

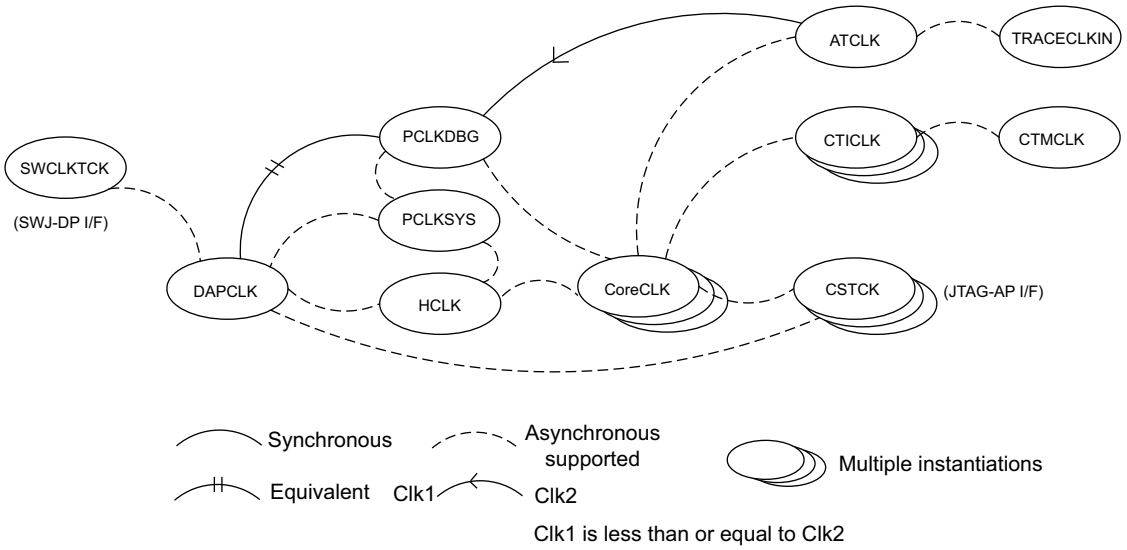


Figure 6-8 Clock domain interactions

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.7 Resets

This section describes:

- *Connection of CSDK resets*
- *Example CoreSight configurations* on page 6-16.

Table 6-3 lists the CoreSight Technology reset signals.

Table 6-3 CSDK reset signals

Reset	Description
ATRESETn	This is the ATB reset. It resets all registers in the **ATCLK** domain. It is active LOW.
nCTIRESET	This is the CTI reset signal. It resets all registers clocked by **CTICLK**. It is active LOW.
DAPRESETn	This is the DAP internal reset. It must be equivalent to **PCLKDBG**. It is active LOW.
HRESETn	This is a SoC provided reset signal that resets all of the AMBA on-chip interconnect. You must use this signal to reset the DAP, AHB-AP, and AHB master port.
HTMHRESETn	This is the HTM reset signal. It is for resetting logic in the AHB domain of the HTM and must not be the same as **HRESETn** to enable the HTM to trace AHB resets.
nCSTRST	This is an internally-generated reset signal, controlled and generated by the JTAG-AP to reset TAP controllers on connected components.
nSRSTOUT	This is an internally-generated reset signal, controlled and generated by the JTAG-AP intended to reset sub-systems associated with the TAP controllers on that scan-chain.
nCTMRESET	This is the CTM reset signal. It resets all registers clocked by **CTMCLK**. It is active LOW.
nPOTRST	This is a true power-on reset signal to the DAP SWJ-DP. It must only reset at power-on. It is active LOW.
nTRST	This is the SWJ-DP TAP state machine clock. It is asynchronous to **DAPCLK**.
PRESETDBGn	This is the Debug APB reset. It resets all registers clocked by **PCLKDBG**. It is active LOW.
PRESETSYSn	This is the DAP APB-Mux reset signal that resets the APB slave input. In a typical system where **HCLK** and **PCLKSYS** are equivalent, this is the same reset signal as **HRESETn**. It is active LOW.
TRESETn	This is the TPIU trace input reset signal. It is active LOW.

6.7.1 Connection of CSDK resets

This section describes the CoreSight Technology resets:

ATRESETn, PRESETDBGn, and DAPRESETn

You can connect all these signals to the same reset signal.

The system requires an external reset synchronizer that enables the resets to be asynchronously asserted, then synchronously deasserted.

TRESETn Can be asserted at the same time as **ATRESETn**, and so it can come from the same reset signal.

If **TRACECLKIN** equals **ATCLK**, then you can use the output of the same synchronizer to deassert **TRESETn**.

If **TRACECLKIN** is not equal to **ATCLK**, then the system requires an additional reset synchronizer that asynchronously asserts the reset, but deasserts the reset synchronously to **TRACECLKIN**.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

nPOTRST This is a true power-on reset signal for the SWJ-DP. It resets all the registers within the Debug Port clocked by TCK, but not part of the TAP state machine, reset by nTRST. It must only be driven LOW at power-on of the platform. To ensure that this signal is internally synchronized to **TCK**, you must fit an external reset synchronizer on **nPOTRST**.

6.7.2 Example CoreSight configurations

Figure 6-9 on page 6-17 shows a sample clock configuration for a typical CoreSight-enabled system.

In this configuration, **ATCLK** and the system interconnect AHB clock, **HCLK**, are equivalent.

ATCLK = **HCLK** = **PSYSCLK**.

PCLKDBG = **DAPCLK**. These are also synchronous to **ATCLK**:

- CTI 1: **CTICLK** = **CoreClk**
- CTI 2: **CTICLK** = **ATCLK**
- CTI 3: **CTICLK** = **ATCLK**.

You can make **CTMCLK** equal to **ATCLK**.

ATCLK, **HCLK**, **PSYSCLK**, **PCLKDBG**, **DAPCLK**, **CoreCLK**, CTI2:**CTICLK**, CTI3:**CTICLK** and **CTMCLK** are all synchronous.

Asynchronous clocks are:

- **CoreClk**
- **TCK**
- **TRACECLKIN**.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

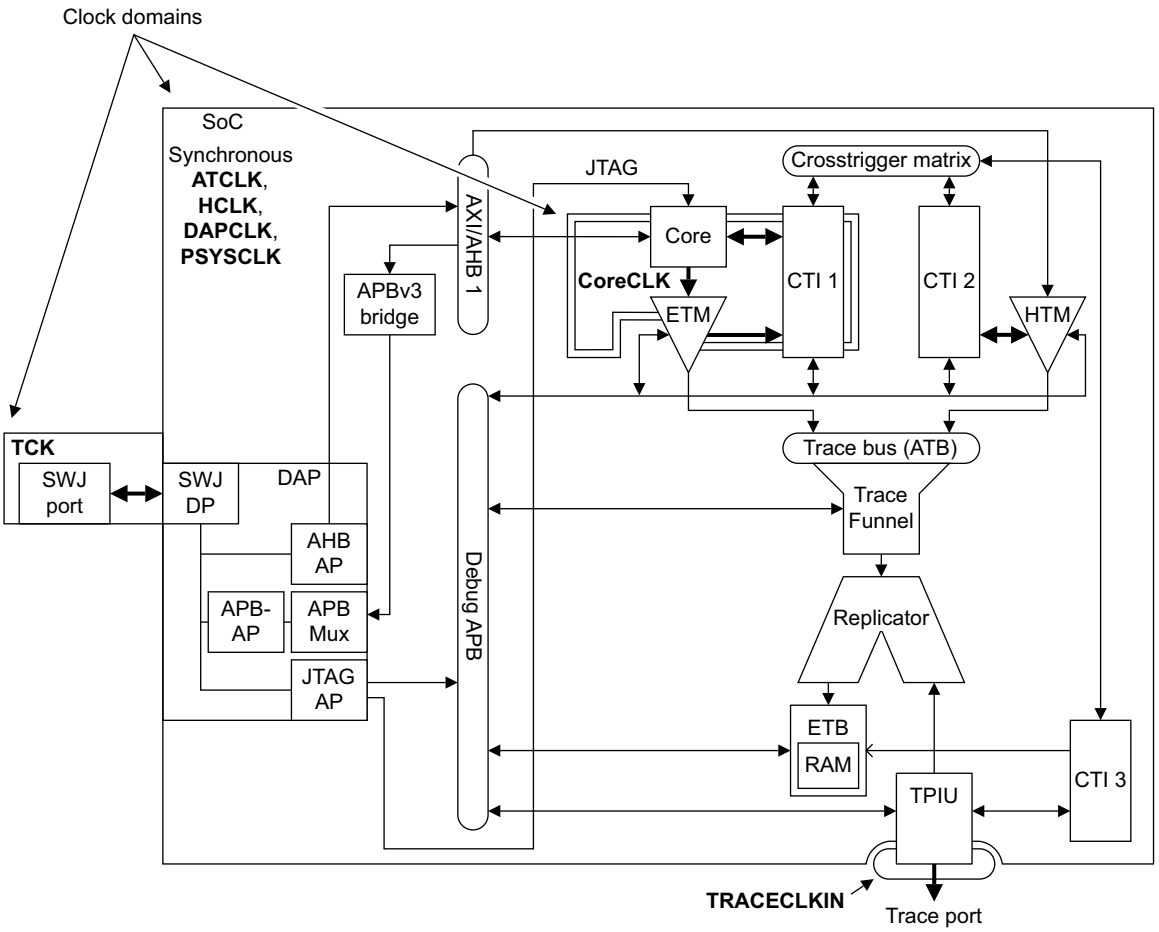


Figure 6-9 Synchronous example clock configuration

Figure 6-10 on page 6-18 shows a sample clock configuration for a CoreSight-enabled system with multiple asynchronous clock domains.

In this configuration, **ATCLK** and the system interconnect AHB clock, **HCLK**, are asynchronous. In the following list one point is asynchronous to other points, with the exception of **PCLKDBG** and **DAPCLK** that must be synchronous to **ATCLK**:

- **ATCLK** = CTI3 **CTICLK**
- **HCLK** = **PSYSCLK** = CTI2 **CTICLK** are synchronous
- **PCLKDBG** = **DAPCLK** and also synchronous to **ATCLK**
- **CoreClk** = CTI1 **CTICLK**
- **TCK**
- **TRACECLKIN**.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

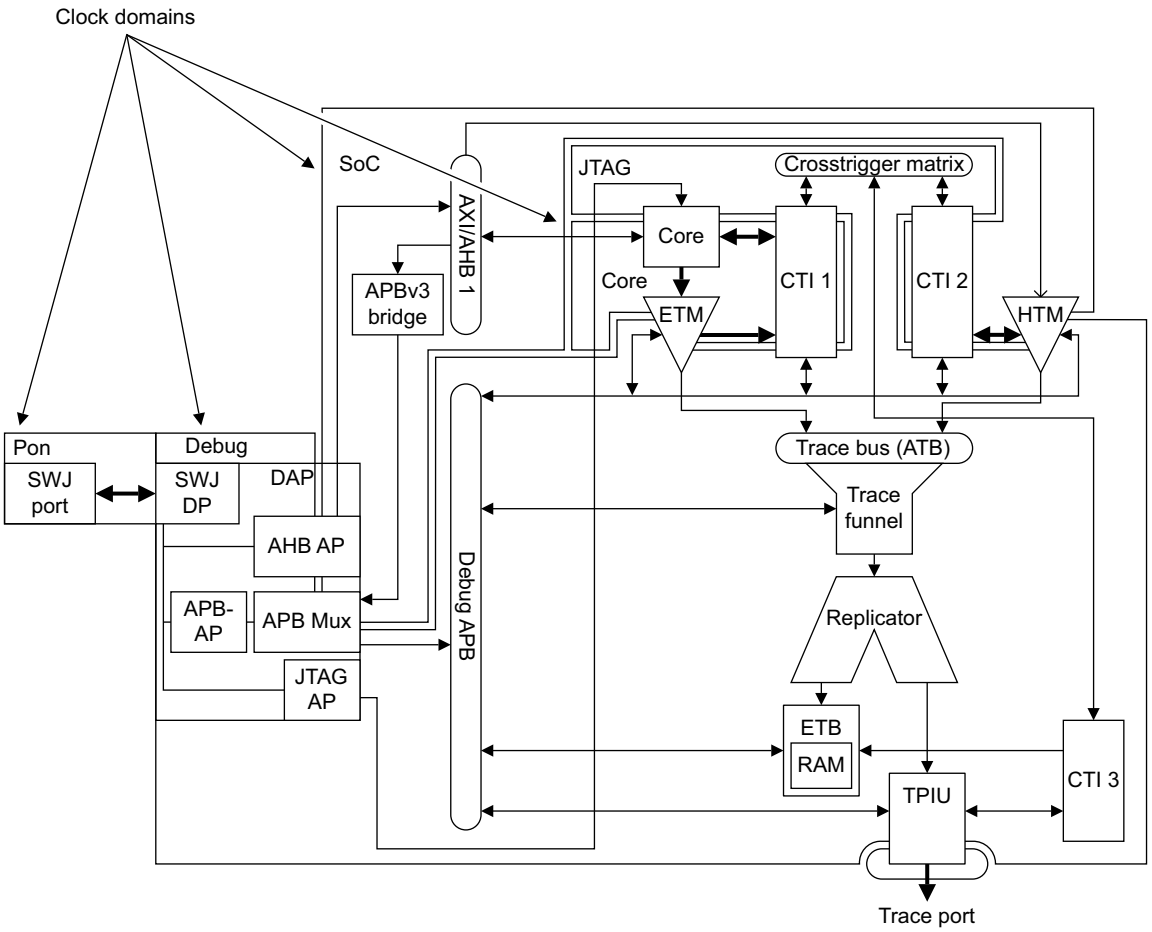


Figure 6-10 Asynchronous example clock configuration

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.8 Tools controlled debug reset

The control and status register provides two bits for reset control of the debug domain, **DAPRESETn**, **PRESETDBGn**, **ATRESETn**. The SWJ-DP registers are in the always powered-on external interface side of the SWJ-DP, and therefore, the system can drive them to make reset requests to a system reset controller. Figure 6-9 on page 6-17 shows the request and acknowledgement timing.

At time T1, a reset request is initiated. At time T2, the reset controller acknowledges that reset of the debug domain has completed. At time T3, the SWJ-DP deasserts the reset request to indicate that it is aware of the reset completion forcing the reset controller to deassert the reset acknowledgement at time T4:

- CTRL/STAT[27:26] provide **CDBGRSTREQ** and **CDBGRSTACK**. The request generates a reset request to a reset controller. When the reset completes and the debug domain has come out of reset, the controller returns **CDBGRSTACK** HIGH.

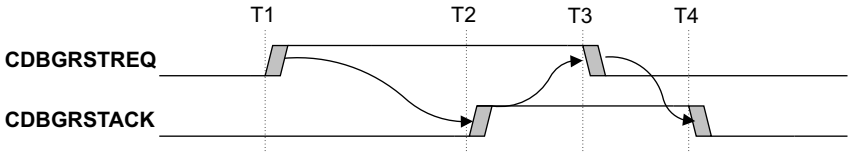


Figure 6-11 Reset handshaking mechanism

- If you require this functionality within the system, then the system reset controller must receive **CDBGRSTREQ**, and generate **CDBGRSTACK** when the reset cycle has completed.
- If you do not require this functionality, you must connect **CDBGRSTACK** LOW.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.9 Interface timing

Because most CoreSight infrastructure components are relatively small, it is expected that synthesis flows would flatten individual components along with additional SoC logic. The timing guidelines help developed components to integrate together successfully. This section describes:

- *Recommended ATB master interface timing parameters*
- *Recommended ATB slave interface timing parameters* on page 6-21.

6.9.1 Recommended ATB master interface timing parameters

Figure 6-12 shows the interface timing for the ATB master in a CoreSight system.

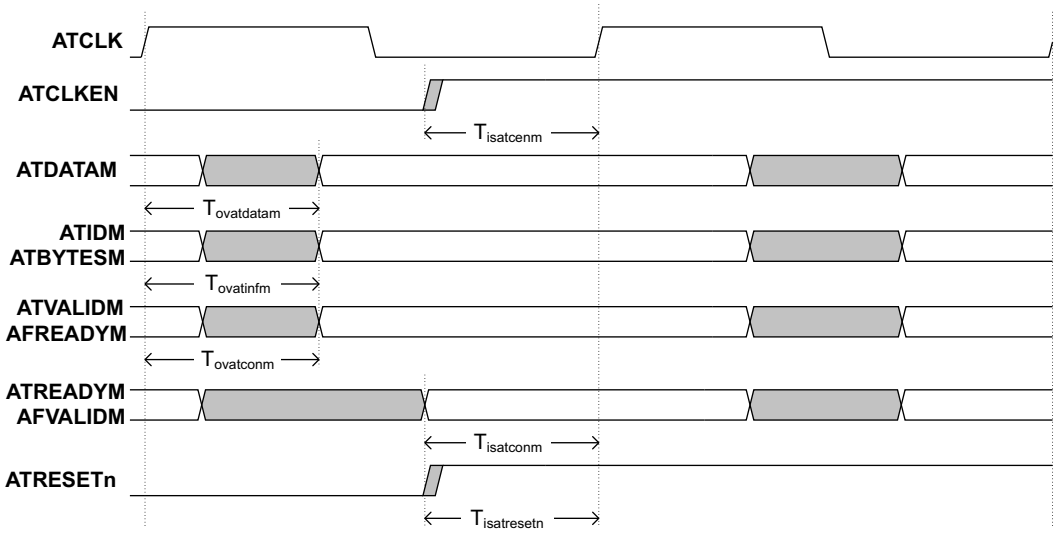


Figure 6-12 ATB master interface timing

Table 6-4 lists the timing constraints that apply for the ATB master interface.

Table 6-4 ATB master interface parameters, input to register, register to output

Parameter	Description	Maximum	Minimum
$T_{isatcenm}$	**ATCLKEN** input setup to rising **ATCLK**	-	30%
$T_{ovatdatam}$	Rising **ATCLK** to **ATDATAM** valid	40%	-
$T_{ovatinfm}$	Rising **ATCLK** to **ATBYTESM** and ATID and ID outputs valid	40%	-
$T_{ovatconm}$	Rising **ATCLK** to ATB control outputs valid	40%	-
$T_{isatconm}$	ATB control inputs setup to rising **ATCLK**	-	30%
$T_{isatresetn}$	**ATRESETn** input setup to rising **ATCLK**	-	30%

Cycle percentages are with respect to the CoreSight component:

- input setup times refer to the minimum percentage of the cycle that must be available to the component
- output valid times refer to the maximum percentage of the cycle until outputs are guaranteed to be valid.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.9.2 Recommended ATB slave interface timing parameters

Figure 6-13 shows the interface timing for the ATB slaves in a CoreSight system.

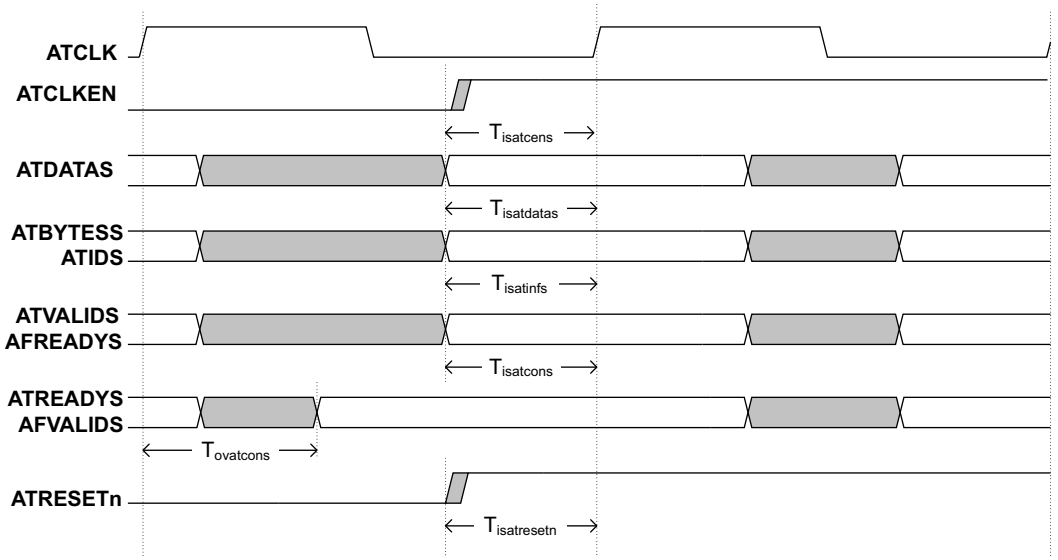


Figure 6-13 ATB slave interface timing

Table 6-5 lists the timing constraints that apply for the ATB slave interface.

Table 6-5 ATB slave interface parameters, input to register, register to output

Parameter	Description	Maximum	Minimum
$T_{isatcens}$	**ATCLKEN** input setup to rising **ATCLK**	-	30%
$T_{isatdatas}$	**ATDATAS** input setup to rising **ATCLK**	-	30%
$T_{isatinfs}$	**ATBYTESM** and **ATID** inputs to rising **ATCLK**	-	30%
$T_{isatcons}$	ATB control inputs setup to rising **ATCLK**	-	30%
$T_{ovatcons}$	Rising **ATCLK** to ATB control outputs valid	40%	-
$T_{isatresetn}$	**ATRESETn** input setup to rising **ATCLK**	-	30%

Cycle percentages are with respect to the CoreSight component:

- input setup times refer to the minimum percentage of the cycle that must be available to the component
- output valid times refer to the maximum percentage of the cycle until outputs are guaranteed to be valid.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

6.10 Timing, synthesis, and placement

This section describes:

- *DAP placement*
- *ATB 1:1 bridge synthesis*
- *TPIU TRACECLK generation*.

6.10.1 DAP placement

It is recommended that you place the DAP near the edge of the SoC because there are external pins for connection to the SWJ interface.

It is strongly recommended that all the debug and trace infrastructure on a SoC is accessible from the DAP. This includes:

- Debug APB
- JTAG interfaces.

There must be no dependencies between debug systems that are accessible from a single DAP. This enables full support for topology detection.

If several, completely independent debug and trace systems are present on completely independent systems, then each can be accessed by an individual DAP.

Debug and trace systems are not independent if there are any of the following:

- shared Debug APB buses
- connected ATB busses between the two systems
- connected JTAG chains
- ROM table references to components in other systems.

This ensures that you can build independent systems instantiated on the same ASIC while maintaining full CoreSight compliance.

6.10.2 ATB 1:1 bridge synthesis

You must use the ATB 1:1 bridge to achieve timing closure when the CoreSight components are spread out across an ASIC. This might happen if, for example, layout constraints require that a trace source is placed close to the processor or bus that it is tracing.

The ATB 1:1 bridge consists of a set of registers across the data interface and the control signals that are emitted from trace sources. This bridge has two ATB interfaces, an input and an output. Both interfaces exist in the same clock domain.

6.10.3 TPIU TRACECLK generation

TRACECLK is a divided by two, exported version of **TRACECLKIN**. The reason for creating a half clock is that the limiting factor for both the Trace Out Port is the slew rate from a zero-to-one and one-to-zero. If it is possible to detect logic 1 and logic 0 on the exported clock within one cycle, then it is also possible to detect two different values on the exported data pins.

TRACECLK can be derived from the negative edge of **TRACECLKIN** to create a sample point within the centre of the stable data, **TRACEDATA**, **TRACECTL**, on each changing edge of **TRACECLK** irrespective of the operating frequency. This method does create a issues during clock-tree synthesis, layout and static timing analysis because of the placement of a negative-edge flop and the requirement for even mark to ratio clock source.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

An alternative approach that reduces clock management issues is by fixing up the clock edges with respect to the clock by adding delays as appropriate to the signal paths after generation. The register that creates the divided by two clock is a standard positive-edge register that operates synchronously to the **TRACEDATA** and **TRACECTL** registers. This method simplifies synthesis in the early stages, and ensures when clock-tree synthesis is performed, all the registers are operating at the same time. To create the sample point at a stable point within the exported data, a you must add a delay to the path of **TRACECLK** between the register and the pad.

Figure 6-14 shows **TRACECLK** at different points within the design and its relationship to the data and control signals, **TRACEDATA** and **TRACECTL**. At the moment of creation from the final registers of the Trace Out port signals, all data edges are aligned as point A in Figure 6-14 shows.

All the signal paths to the pads are subject to delays as a result of the path lengths, at point B, from wire delay. These delays must be minimized where possible by placing the registers as close to the pads as possible. Each path must be re-balanced to remove the relative skew between signals by adding in equivalent delays. An extra delay must be incorporated on the **TRACECLK** path to ensure the waveform at point C is achieved and that the rising and falling edges of **TRACECLK** correspond to the centre of stable data on **TRACECTL** and **TRACEDATA** as Figure 6-15 on page 6-24 shows.

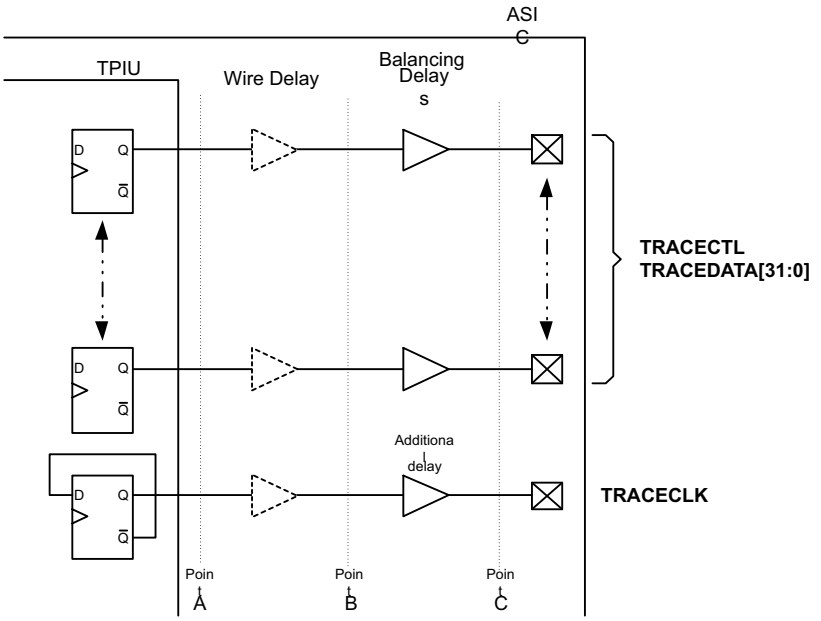


Figure 6-14 Balancing TRACECLK

Figure 6-15 on page 6-24 shows how the rising and falling edges of **TRACECLK** correspond to the centre of stable data on **TRACECTL** and **TRACEDATA** at point C.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

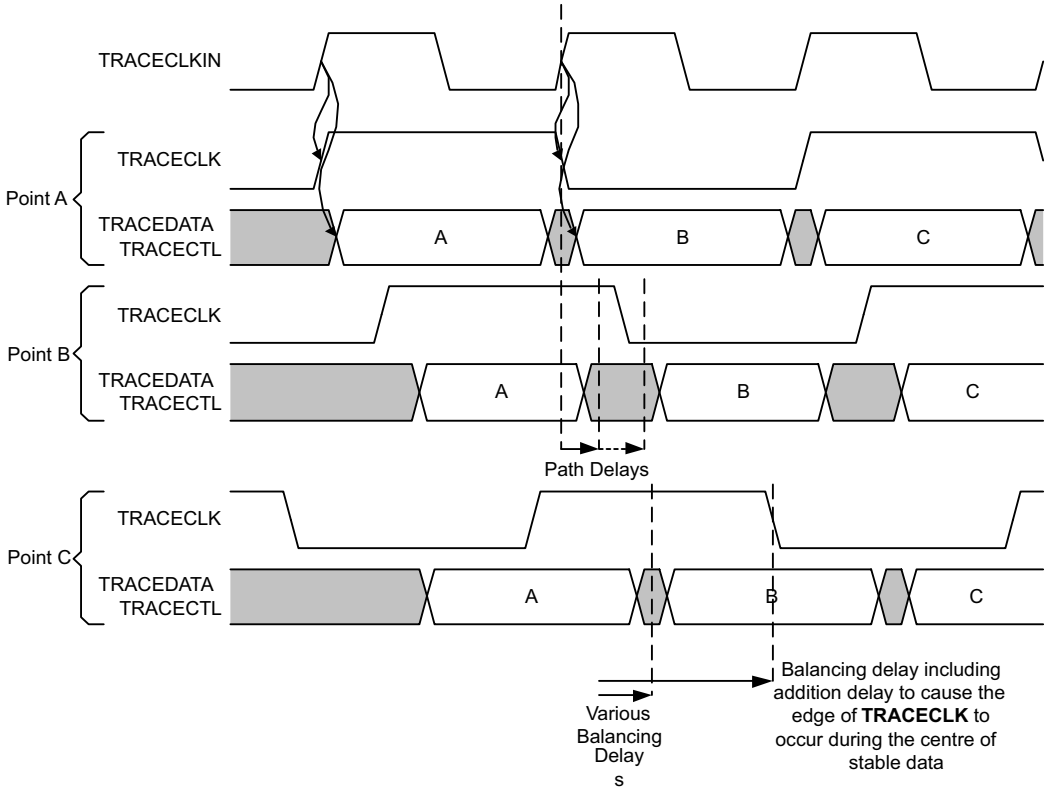


Figure 6-15 Timing balance

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Appendix A
Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Differences between issue B and issue C

Change	Location	Affects
Replaced ‘The CoreSight Design Kit’ with ‘CoreSight Technology’	Throughout book	All revisions
Replaced JTAG-Port with SWJ-Port in Figure 1-1 on page 1-4	Figure 1-1 on page 1-4	All revisions
Updated the Cross Triggering section	*Cross Triggering* on page 1-5	All revisions
Added *AMBA Advanced eXtensible Interface* (AXI) description to CoreSight components section	*CoreSight components* on page 2-4	All revisions
Updated Trace sources section	*Trace sources* on page 2-7	All revisions
Updated Trace links section	*Trace links* on page 2-9	All revisions
Updated Trace sinks section	*Trace sinks* on page 2-10	All revisions
Updated External debug hardware and software section	*External debug hardware and software* on page 2-12	All revisions
Updated About CoreSight technology and ETM architectures features section for CoreSight technology components	*About CoreSight Technology and ETM architectures features* on page 3-2	All revisions
Updated Identification Register value for SW-DP, and SWJ-DP	Table 3-1 on page 3-4	All revisions
Added STM to the Trace source HTM and ITM features table	Table 3-4 on page 3-7	All revisions
Added *Embedded Trace FIFO* (ETF) to Link component features	Table 3-6 on page 3-9	All revisions

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Table A-1 Differences between issue B and issue C (continued)

Change	Location	Affects
Added ETB to Sink component features, part 1	Table 3-7 on page 3-10	All revisions
Added ETR to Sink component features, part 2	Table 3-8 on page 3-11	All revisions
Added Cortex - A9 and Cortex - A5 to Debug components, part 2	Table 3-10 on page 3-14	All revisions
Added PFTv1 description to Architectural Features of ARM trace sources section	*Architectural features of ARM trace sources* on page 3-15	All revisions
Added CS PTM-A9 and CS ETM-A5 to ARM trace source component features, part 2	Table 3-12 on page 3-16	All revisions
Added new section for access to debug components	*Access to debug components* on page 4-5	All revisions
Added STM connection information in Table 4-3 on page 4-15	Table 4-3 on page 4-15	All revisions
Added TMC connection information in Table 4-4 on page 4-15	Table 4-4 on page 4-15	All revisions
Added new section for Mixing JTAG devices with SWJ-DP	*Mixing JTAG devices with SWJ-DP* on page 4-9	All revisions
Updated the Designing your trace system section, also added a new section, Using additional buffers in trace systems	*Designing your trace system* on page 5-4	All revisions
Replaced JTAG power and JTAG interface with SWJ power and SWJ interface respectively	*Power domains and system design* on page 6-5	All revisions
Updated DAP section	*Debug Access Port (DAP)* on page 6-7	All revisions
Updated Power up request and acknowledge signal connections table	Table 6-1 on page 6-12	All revisions
Added **CSRTCK**, return test clock to the CoreSight Technology clocks table	Table 6-2 on page 6-13	All revisions
Updated CoreSight system clock design section for description	*CoreSight system clock design* on page 6-13	All revisions
Added **HTMHRESETn** and **nSRSTOUT**, reset signals to CoreSight Technology reset signals table	Table 6-3 on page 6-15	All revisions
Updated the **nPOTRST** description in Connection of CSDK resets section	Table 6-3 on page 6-15	All revisions
Replaced JTAG-DP with SWJ-DP	*Tools controlled debug reset* on page 6-19	All revisions
Replaced JTAG interface with SWJ interface	*DAP placement* on page 6-22	All revisions
Updated the description in TPIU TRACECLK generation section	*TPIU TRACECLK generation* on page 6-22	All revisions

Table A-2 Differences between issue C and issue D

Change	Location	Affects
Updated the Figure 6-8 on page 6-14 for Clk1 and Clk2 key statement	Figure 6-8 on page 6-14	All revisions

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Glossary

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

Abort An exception caused by an illegal memory access. Aborts can be caused by the external memory system or by the memory-management hardware, that might include a *Memory Management Unit* (MMU) or a *Memory Protection Unit* (MPU).

See also Data abort, External abort and Prefetch abort.

Advanced eXtensible Interface (AXI)
A bus protocol that supports separate phases for address or control and data, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels, issuing multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.

The AXI protocol includes optional extensions for signaling for low-power operation.

Advanced High-performance Bus (AHB)
A bus protocol with a fixed pipeline between the address or control and data phases. It supports a subset of the functionality of the AMBA AXI protocol. The full AMBA AHB protocol specification includes a number of features that are not commonly required for master and slave implementations and ARM recommends using the AMBA AHB-Lite subset of the protocol.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)
The AMBA family of protocol specifications is the ARM open standard for on-chip buses. AMBA provides a strategy for the interconnection and management of the functional blocks that make up a *System-on-Chip* (SoC). Applications include the development of embedded systems with one or more processors or signal processors and multiple peripherals. AMBA defines a common backbone for SoC modules, and therefore complements a reusable design methodology.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

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

Advanced Trace Bus (ATB)
A bus used by trace devices to share CoreSight capture resources.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)
An optional component of the DAP that provides an AHB interface to a SoC.

AHB-AP *See* AHB Access Port.

AHB-Lite A subset of the full AMBA AHB protocol specification. It provides all of the basic functions required by the majority of AMBA AHB slave and master designs, particularly when used with a multi-layer AMBA interconnect. In most cases, the extra facilities provided by a full AMBA AHB interface are implemented more efficiently using an AMBA AXI protocol interface.

Aligned A data item stored at an address that is divisible by the number of bytes that defines its data size is said to be aligned. Aligned doublewords, words, and halfwords have addresses that are divisible by eight, four, and two respectively. The terms doubleword-aligned, word-aligned, and halfword-aligned therefore stipulate addresses that are divisible by eight, four, and two respectively. An aligned access is one where the address of the access is aligned to the size of an element of the access.

AMBA *See* Advanced Microcontroller Bus Architecture.

APB *See* Advanced Peripheral Bus.

APB Access Port (APB-AP)
An optional component of the DAP that provides an APB interface to a SoC, usually to its main functional buses.

APB-AP *See* APB Access Port.

ATB *See* Advanced Trace Bus.

ATB bridge A synchronous ATB bridge provides a register slice that helps timing closure by adding a pipeline stage. It also provides a unidirectional link between two synchronous ATB domains.

An asynchronous ATB bridge provides a unidirectional link between two ATB domains with asynchronous clocks. It supports connection of components with ATB ports in different clock domains.

AXI *See* Advanced eXtensible Interface.

Base register A register specified by a load or store instruction that is used as the base value for the address calculation for the instruction. Depending on the instruction and its addressing mode, an offset can be added to or subtracted from the base register value to form the virtual address that is sent to memory.

Beat Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats.

See also Burst.

Breakpoint A breakpoint is a debug event triggered by the execution of a particular instruction. It is specified in terms of one or both of the address of the instruction and the state of the processor when the instruction is executed.

See also Watchpoint.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Burst
A group of transfers to consecutive addresses. Because the addresses are consecutive, the device transmitting the data does not have to supply an address for any transfer after the first one. This increases the speed at which the burst occurs. If using an AMBA interface, the transmitting device controls the burst using signals that indicate the length of the burst and how the addresses are incremented.

See also Beat.

Coprocessor
A processor that supplements the main processor to carry out additional functions that the main processor cannot perform. The ARM architecture defines an interface to up to 16 coprocessors, CP0-CP15 for use by ARM:

- CP15 insturctions access the System Control processor
- CP14 instructions access control registers for debug, trace, and execution environment features
- CP10 an CP11 instruction space is for floating-point and Advanced SIMD instructions if supported.

CoreSight
ARM on-chip debug and trace components, that provide the infrastructure for monitoring, tracing, and debugging a complete system on chip.

See also CoreSight ECT, CoreSight ETB, CoreSight ETM, Trace Funnel, and *Trace Port Interface Unit* (TPIU).

CoreSight ETB
CoreSight ETB is a trace sink that provides on-chip storage of trace data using a configurable sized RAM.

See also CoreSight, CoreSight ETB, Embedded Trace Buffer, and Embedded Trace Macrocell.

Cross Trigger Interface (CTI)
Part of an *Embedded Cross Trigger* (ECT) device. In an ECT, the CTI provides the interface between a processor or ETM and the CTM.

Cross Trigger Matrix (CTM)
In an ECT device, the CTM combines the trigger requests generated by CTIs and broadcasts them to all CTIs as channel triggers.

CTI
See Cross Trigger Interface.

CTM
See Cross Trigger Matrix.

DAP
See Debug Access Port.

DBGTAP
See Debug Test Access Port.

Debug Access Port (DAP)
A block that acts as a master on a system bus and provides access to the bus from an external debugger.

Debug Test Access Port (DBGTAP)
A debug control and data interface based on the IEEE 1149.1 JTAG *Test Access Port* (TAP). The interface has four or five signals.

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

Device Validation Suite (DVS)
A set of tests to check the functionality of a device against the functionality defined in the Technical Reference Manual. For example these tests stress the *Bus Interface Unit* (BIU), low-level memory sub-system, pipeline, cache and *Tightly Coupled Memory* (TCM) behavior.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Digital Signal Processing (DSP)
A variety of algorithms to process signals that have been sampled and converted to digital form. Saturated arithmetic is often used in such algorithms.

DSM *See* Design Simulation Model.

DSP *See* Digital Signal Processing.

DVS *See* Device Validation Suite.

ECT *See* Embedded Cross Trigger.

Embedded Cross Trigger (ECT)
A modular system that supports the interaction and synchronization of multiple triggering events with an SoC.

Embedded Trace Buffer (ETB)
Provides on-chip storage of trace data using a configurable sized RAM.

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

EmbeddedICE logic An on-chip logic block that provides TAP-based debug support for an ARM processor. It is accessed through the DAP on the ARM processor.

EmbeddedICE-RT Hardware provided by an ARM processor to aid debugging in real-time.

Endianness The scheme that determines the order of successive bytes of a data word when it is stored in memory.

ETB *See* Embedded Trace Buffer.

ETM *See* Embedded Trace Macrocell.

Event In an ARM trace macrocell, event has a particular meaning and these events can be simple or complex:

- **Simple** An observable condition that a trace macrocell can use to control aspects of a trace.
- **Complex** A boolean combination of simple events that a trace macrocell can use to control aspects of a trace.

Exception A mechanism to handle a fault or error event. For example, exceptions handle external interrupts and undefined instructions.

Formatter In an ETB or TPIU, an internal input block that embeds the trace source ID in the data to create a single trace stream.

Half-rate clocking
In an ARM trace macrocell, 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.

Host A computer that provides data and other services to another computer. Especially, a computer providing debugging services to a target being debugged.

IEM *See* Intelligent Energy Management.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Implementation-defined
Behavior that is not defined by the architecture, but is defined and documented by the implementation.

Implementation-specific
See Implementation-defined

Imprecise tracing In an ARM trace macrocell, a filtering configuration where instruction or data tracing can start or finish earlier or later than expected. Most imprecise cases cause tracing to start or finish later than expected.

For example, if **TraceEnable** is configured to use a counter so that tracing begins after the fourth write to a location in memory, the instruction that caused the fourth write is not traced, although subsequent instructions are. This is because the use of a counter in the **TraceEnable** configuration always results in imprecise tracing.

Instrumentation trace
A component for debugging real-time systems through a simple memory-mapped trace interface, providing printf style debugging.

Intelligent Energy Management (IEM)
An ARM technology that reduces device power consumption by dynamic voltage scaling and clock frequency variation.

Interrupt handler *See* Exception handler.

Invalidate Marking a cache line as being not valid, by clearing the valid bit to 0. This must be done whenever the line does not contain a valid cache entry. For example, after a cache flush all lines are invalid.

Jazelle state In Jazelle state the processor executes Java bytecodes as part of a *Java Virtual Machine* (JVM).

See also ARM state, Thumb state, and ThumbEE state.

JTAG Access Port (JTAG-AP)
An optional component of the DAP that provides debugger access to on-chip scan chains.

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 integrated with application-specific logic.

MPU *See* Memory Protection Unit.

Multi-ICE A JTAG-based tool for debugging embedded systems.

Multi-master AHB Typically a shared, not multi-layer, AHB interconnect scheme. More than one master connects to a single AMBA AHB link. In this case, the bus is implemented with a set of full AMBA AHB master interfaces. Masters that use the AMBA AHB-Lite protocol must connect through a wrapper to supply full AMBA AHB master signals to support multi-master operation.

Power-on reset *See* Cold reset.

Prefetch abort An indication from a memory system to the processor that an instruction has been fetched from an illegal memory location. An exception must be taken if the processor attempts to execute the instruction. A Prefetch abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction memory.

See also Data abort, External abort and Abort.

Read Memory operations that have the semantics of a load. See the *ARM Architecture Reference Manual* for more information.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

RealView ICE ARM JTAG interface unit for debugging embedded processor cores that uses a DBGTAP or Serial Wire interface.

Remapping Changing the address of physical memory or devices after the application has started executing. This might be done to permit RAM to replace ROM when the initialization has completed.

Replicator In an ARM trace macrocell, a replicator enables two trace sinks to be wired together and to operate independently on the same incoming trace stream. The input trace stream is output onto two independent ATB ports.

Reserved Registers and instructions that are reserved are Unpredictable unless otherwise stated. Bit positions described as Reserved are UNK/SBZP.

SBO *See* Should Be One.

SBOP *See* Should Be One or Preserved.

SBZ *See* Should Be Zero.

SBZP *See* Should Be Zero or Preserved.

SDF *See* Standard Delay Format.

Security hole A mechanism that bypasses system protection.

Serial Wire Debug (SWD)
A debug implementation that uses a serial connection between the SoC and a debugger.

This connection normally requires a bi-directional data signal and a separate clock signal, rather than the four to six signals required for a JTAG connection.

Serial Wire Debug Port (SWDP)
The interface for Serial Wire Debug.

Set *See* Cache set.

Should Be One (SBO)
Software must write as 1, or all 1s for bit fields. Writing any other value produces Unpredictable results.

Should Be One or Preserved (SBOP)
Software must write as 1, or all 1s for bit fields, if the value is being written without having previously been read, or if the register has not been initialized. If the register has previously been read, software must preserve the field value by writing back the value that was read from the same field on the same processor.

If software writes a value that does not meet this requirement, the result is Unpredictable.

Hardware ignores writes to these fields.

Should Be Zero (SBZ)
Software must write as 0, or all 0s for bit fields. Writing any other value produces Unpredictable results.

Should Be Zero or Preserved (SBZP)
Software must write as 0, or all 0s for a bit field, if the value is being written without having previously been read, or if the register has not been initialized. If the register has previously been read, software must preserve the field value by writing back the value that was read from the same field on the same processor.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Subnormal value In floating-point operation, a value in the range ($-2^{Emin} < x < 2^{Emin}$), except for plus or minus 0. In the IEEE 754 standard format for single-precision and double-precision operands, a subnormal value has a zero exponent and a nonzero fraction field. The IEEE 754 standard requires that the generation and manipulation of subnormal operands be performed with the same precision as normal operands.

Support code In a floating-point implementation, system software that complements the hardware VFP implementation to provide compatibility with the IEEE 754 standard. The support code has a library of routines that perform supported functions, such as divide with unsupported inputs or inputs that might generate an exception, in addition to operations beyond the scope of the hardware. The support code has a set of exception handlers to process exceptional conditions in compliance with the IEEE 754 standard.

SVC *See* Supervisor Call.

SWD *See* Serial Wire Debug.

SWDP *See* Serial Wire Debug Port.

SWI *See* Supervisor Call.

Tag bits In a cache implementation, bits [31:(L+S)] of a virtual address, where L = log2 (cache line length) and S = log2 (number of cache sets). A cache hit occurs if the tag bits of the virtual address supplied by the processor match the tag bits associated with a valid line in the selected cache set.

See also Cache terminology diagram on the last page of this glossary.

TCD *See* Trace Capture Device.

TPA *See* Trace Port Analyzer.

TPIU *See* Trace Port Interface Unit.

Trace Capture Device (TCD)
A generic term to describe Trace Port Analyzers, logic analyzers, and on-chip trace buffers.

Trace funnel In an ARM trace macrocell, a device that combines multiple trace sources onto a single bus.

See also AHB Trace Macrocell, CoreSight, CoreSight ETM, and Embedded Trace Macrocell.

Trace hardware A term for a device that contains an ARM 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.

Trace Port Interface Unit (TPIU)
Drains trace data and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.

Unaligned An unaligned access is an access where the address of the access is not aligned to the size of an element of the access.

Undefined Indicates an instruction that generates an Undefined Instruction exception. See the *ARM Architecture Reference Manual* for more information.

UNK *See* Unknown.

UNK/SBOP A field that is Unknown on reads and Should Be One or Preserved on writes.

UNK/SBZP A field that is Unknown on reads and Should Be Zero or Preserved on writes.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential

Unknown An Unknown value does not contain valid data, and can vary from moment to moment, instruction to instruction, and implementation to implementation. An Unknown value must not be a security hole.

UNP *See* Unpredictable.

Unpredictable For a processor means the behavior cannot be relied on. Unpredictable behavior must not represent security holes. Unpredictable behavior must not halt or hang the processor, or any parts of the system.

Unpredictable For an ARM trace macrocell, means that the behavior of the macrocell cannot be relied on. Such conditions have not been validated. When applied to the programming of an event resource, only the output of that event resource is Unpredictable. Unpredictable behavior can affect the behavior of the entire system, because the trace macrocell can cause the processor to enter debug state, and external outputs can be used for other purposes.

WA *See* Write-Allocate cache.

Way *See* Cache way.

WB *See* Write-Back cache.

Word A 32-bit data item. Words are normally word-aligned in ARM systems.

Write Operations that have the semantics of a store. See the *ARM Architecture Reference Manual* for more information.

Copyright © 2004, 2007, 2010 ARM Limited. All rights reserved.
Non-Confidential