CoreSight™ Components

Technical Reference Manual

ARM®

ARM DDI 0314C

CoreSight Components
Technical Reference Manual

Release Information

Change history

Date	Issue	Confidentiality	Change
29 September 2004	A	Non-confidential	First release for r0p0.
24 March 2005	B	Non-confidential	Updated for r0p1. Programmer's model revised.
31 July 2006	C	Non-confidential	Changed to CoreSight Components Technical Reference Manual. Serial Wire and JTAG (SWJ) information added to Chapter 3. Chapter 11 (SWV), Chapter 12 (SWO), Chapter 13 (ITM), and Appendix C (SWD and JTAG Trace Connector) added.

Product Status

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

Web Address

`http://www.arm.com`

Contents
CoreSight Components Technical Reference Manual

List of Tables
CoreSight Components Technical Reference Manual

List of Figures
CoreSight Components Technical Reference Manual

Preface

This preface introduces the *CoreSight Components Technical Reference Manual*. It contains the following sections:

- *About this document* on page xxii
- *Feedback* on page xxviii.

About this document

This is the *Technical Reference Manual* (TRM) for the CoreSight components.

Product revision status

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

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

Intended audience

This manual is written for the following target audiences:

- Hardware and software engineers who want to incorporate a CoreSight System Component into their design and produce real-time instruction and data trace information from an ASIC.
- Software engineers writing tools to use CoreSight components.

This TRM assumes that readers are familiar with AMBA bus design and JTAG methodology.

Using this manual

Note

Details of CoreSight component memory maps, registers, and CoreSight component programmer's models are in the relevant chapters.

This manual is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an overview of the CoreSight components. This chapter also lists and classifies all CoreSight components.

Chapter 2 *CoreSight Programmer's Model*

Read this chapter for a description of the CoreSight programmer's model.

Chapter 3 ***Debug Access Port***

Read this chapter for a description of the *Debug Access Port* (DAP). The chapter gives an overview of the DAP and detailed descriptions of the following components:

- SWJ-DP
- JTAG-DP
- SW-DP
- JTAG-AP
- AHB-AP
- APB-AP
- APB-multiplexor
- ROM table.

Chapter 5 ***Embedded Cross Trigger***

Read this chapter for a description of the *Embedded Cross Trigger* (ECT). Contains ECT connectivity recommendations for cores and other components.

Chapter 4 ***CoreSight Trace Sources***

Read this chapter for a brief description of CoreSight trace sources, *AMBA Trace Macrocell* (HTM) and CoreSight *Embedded Trace Macrocells* (ETMs).

Chapter 6 ***ATB 1:1 Bridge***

Read this chapter for a description of the *AMBA Trace Bus* (ATB) 1:1 bridge.

Chapter 7 ***ATB Replicator***

Read this chapter for a description of the ATB replicator.

Chapter 8 ***CoreSight Trace Funnel***

Read this chapter for a description of the Trace Funnel.

Chapter 9 ***Trace Port Interface Unit***

Read this chapter for a description of the *Trace Port Interface Unit* (TPIU).

Chapter 10 ***Embedded Trace Buffer***

Read this chapter for a description of the *Embedded Trace Buffer* (ETB). Differences from other ETBs are listed at the end of this chapter.

Chapter 11 ***Serial Wire Viewer***

Read this chapter for a description of the *Serial Wire Viewer* (SWV).

Chapter 12 ***Serial Wire Output***

Read this chapter for a description of the *Serial Wire Output* (SWO).

Chapter 13 ***Instrumentation Trace Macrocell***

Read this chapter for a description of the *Instrumentation Trace Macrocell* (ITM).

Appendix A ***CoreSight Port List***

Read this appendix for a description of the CoreSight component signals.

Appendix B ***CoreSight Components and Clock Domains***

Read this appendix for a description of the CoreSight components and their respective clock domains.

Appendix C ***Serial Wire Debug and JTAG Trace Connector***

Read this appendix for a description of the SWD and JTAG trace connector used for debug targets.

Appendix D ***Deprecated SWJ-DP Switching Sequences***

Read this appendix for a description of the switching sequences used in earlier versions of the SWJ-DP.

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

Conventions

Conventions that this manual can use are described in:

- *Typographical* on page xxv
- *Timing diagrams* on page xxv
- *Signals* on page xxvi
- *Numbering* on page xxvi.

Typographical

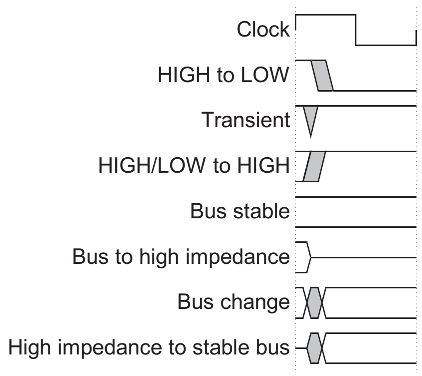
The typographical conventions are:

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.
<u>mono</u>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

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

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

Signals

The signal conventions are:

Signal level	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.
Prefix H	Denotes *Advanced High-performance Bus* (AHB) signals.
Prefix n	Denotes active-LOW signals except in the case of AHB or *Advanced Peripheral Bus* (APB) reset signals.
Prefix P	Denotes APB signals.
Suffix n	Denotes AXI, AHB, and APB reset signals.

Numbering

The numbering convention is:

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

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

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

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for the ARM family of processors.

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

ARM publications

This document contains information that is specific to the CoreSight DK. Refer to the following documents for other relevant information:

- *CoreSight System Design Guide*, ARM DGI 0012
- *CoreSight Architecture Specification*, ARM IHI 0029
- *CoreSight Components Implementation Guide*, ARM DII 0143
- *CoreSight DK9 Implementation and Integration Manual*, ARM DII 0131
- *CoreSight DK11 Implementation and Integration Manual*, ARM DII 0092
- *CoreSight DK-A8 Integration Manual*, ARM DII 0135
- *CoreSight Serial Wire Debug Integration Manual*, ARM DII 0095
- *CoreSight Serial Wire Viewer Integration Manual*, ARM DII 0096
- *CoreSight ETM9 Technical Reference Manual,* ARM DDI 0315
- *CoreSight ETM9 Implementation Guide*, ARM DDI 0093
- *CoreSight ETM9 Integration Manual*, ARM DII 0094
- *CoreSight ETM11 Technical Reference Manual,* ARM DDI 0318
- *CoreSight ETM11 Implementation Guide,* ARM DII 0097
- *CoreSight ETM11 Integration Manual,* ARM DII 0098
- *ETM Architecture Specification,* ARM IHI 0014
- *AMBA™ AHB Trace Macrocell (HTM) Technical Reference Manual*, ARM DDI 0328
- *Systems IP ARM11 AMBA (Rev 2.0) AHB Extensions*, ARM IHI 0023
- *AMBA 3 APB Protocol*, ARM IHI 0024
- *ARM Architecture Reference Manual*, ARM DDI 0100
- *RealView ICE User Guide*, ARM DUI 0155.

Feedback

ARM Limited welcomes feedback both on the CoreSight components and the documentation.

Feedback on the CoreSight Components

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

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

Feedback on this manual

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

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

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter contains an introduction to the CoreSight components. It contains the following sections:

1.1 About the CoreSight components

The CoreSight™ components address 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.

1.1.1 Capabilities

The CoreSight components provide the following capabilities for system-wide trace:

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

1.1.2 Structure of the CoreSight Design Kit

The CoreSight components include the following:

Control and access components

Control and access components configure, access, and control the generation of trace. They do not generate trace, nor process the trace data. Examples include:

- *Debug Access Port* (DAP).
  See Chapter 3 *Debug Access Port*.
- *Embedded Cross Trigger* (ECT).
  See Chapter 5 *Embedded Cross Trigger*.

Sources

Sources generate trace data for output through the *AMBA Trace Bus* (ATB). Examples include:

- *AHB Trace Macrocell* (HTM), documented separately.
  See *Further reading* on page xxvii
- CoreSight *Embedded Trace Macrocells* (ETMs), documented separately.
  See *Further reading* on page xxvii.

Links

Links provide connection, triggering, and flow of trace data. Examples include:

- Synchronous 1:1 ATB bridge.

  See Chapter 6 *ATB 1:1 Bridge*.
- Replicator.

  See Chapter 7 *ATB Replicator*.
- Trace funnel.

  See Chapter 8 *CoreSight Trace Funnel*.

Sinks

Sinks are the end points for trace data on the SoC. Examples include:

- *Trace Port Interface Unit* (TPIU) for output of trace data off-chip.

  See Chapter 9 *Trace Port Interface Unit*.
- *Embedded Trace Buffer* (ETB) for on-chip storage of trace data in RAM.

  See Chapter 10 *Embedded Trace Buffer*.

1.2 CoreSight block summary

Table 1-1 shows the main CoreSight blocks.

Table 1-1 CoreSight block summary

Block name	Description	Revision
CSBRIDGESYNC1T1	Synchronous 1:1 ATB Bridge	r0p0
CSCTI	Cross Trigger Interface	r0p0
CSCTM	Cross Trigger Matrix	r0p0
CSETB	Embedded Trace Buffer	r0p0
CSREPLICATOR	ATB Replicator	r0p0
CSITM	Instrumentation Trace Macrocell	r0p0
CSSWO	Serial Wire Output	r0p0
CSSWV	Serial Wire Viewer	r0p0
CSTFUNNEL	Trace Funnel	r0p0
CSTPIU	Trace Port Interface Unit	r0p1
Debug Access Port blocks:		
DAPAHBAP	AHB Access Port	r0p2
DAPAPBAP	APB Access Port	r0p0
DAPAPBMUX	APB Multiplexor	r0p0
DAPJTAGAP	JTAG Access Port	r0p1
DAPROM	ROM Table	r0p0
DAPSWJDP	Serial Wire and JTAG Access Port	r0p1

Note

See the Release Notes for a list of the blocks supplied with the version of the product you have received.

1.3 Typical CoreSight Design Kit debugging environment

Figure 1-1 shows example CoreSight components in one possible debugging environment.

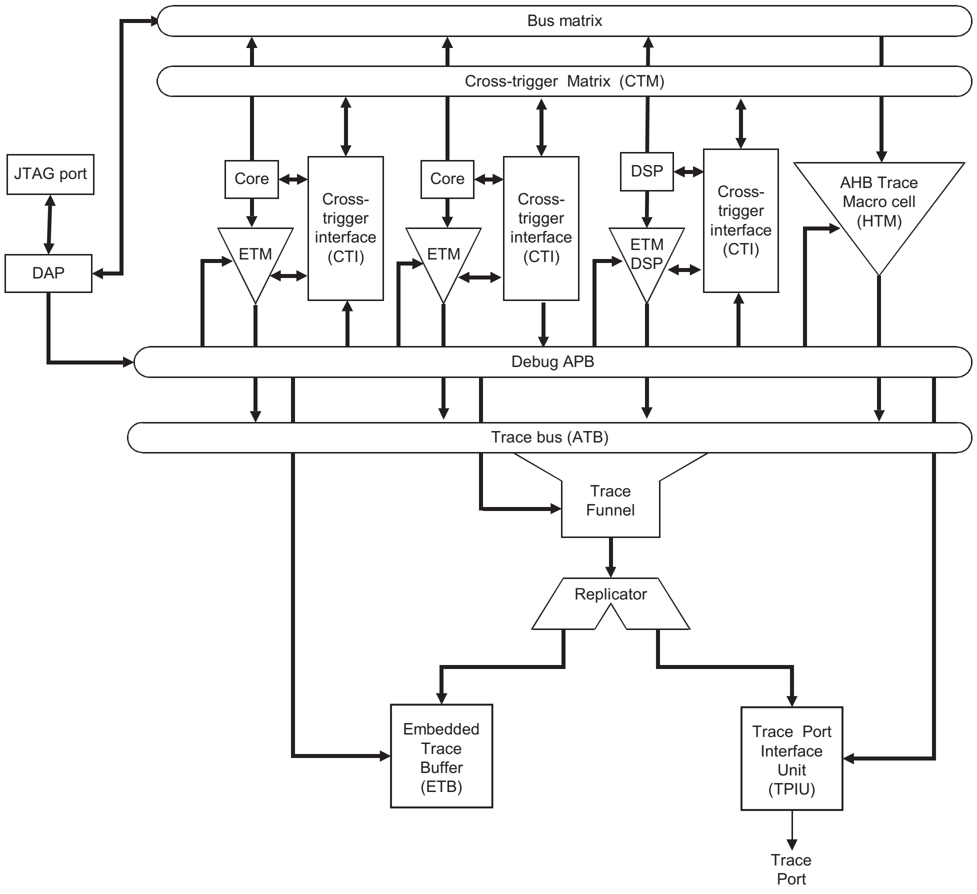

Bus matrix
Cross-trigger Matrix (CTM)
JTAG port
DAP
Core
Cross-trigger interface (CTI)
ETM
Core
Cross-trigger interface (CTI)
ETM
DSP
Cross-trigger interface (CTI)
ETM DSP
AHB Trace Macro cell (HTM)
Debug APB
Trace bus (ATB)
Trace Funnel
Replicator
Embedded Trace Buffer (ETB)
Trace Port Interface Unit (TPIU)
Trace Port

Figure 1-1 CoreSight debugging environment

Chapter 2
CoreSight Programmer's Model

This chapter describes the CoreSight programmer's model. It contains the following sections:

2.1 About the CoreSight programmer's model

Each CoreSight component has a 4KB location block in the CoreSight memory map. The base address of the CoreSight location block is not fixed but the address offsets are fixed.

The following apply to all CoreSight registers:

- All registers are word-aligned and must be accessed as 32-bit.
- Reserved or unused address locations must not be accessed because this can result in Unpredictable behavior.
- Reserved or unused bits of registers must be written as zero, and ignored on read unless otherwise stated in the relevant text.
- All register bits are reset to a logic 0 by a system or power-on reset unless otherwise stated in the relevant text.
- All registers support *Read and Write* (R/W) accesses unless otherwise stated in the relevant text. A *Write-Only* (WO) access updates the contents of a register and a *Read-Only* (RO) access returns the contents of the register.

Note

Only the following components are relevant in the CoreSight programmer's model:

- ROM table
- Cross-trigger interface
- ETM
- HTM
- Trace Funnel
- TPIU
- ETB.

2.2 CoreSight component programmer's model

This section defines the standardized set of registers that must be implemented on all CoreSight components. Components might only use a subset of them where permitted but these register locations must be readable on all components claiming CoreSight compliance.

The register set is standardized across all CoreSight components as defined in the *CoreSight Architecture Specification*.

The register structure is taken from the Peripheral ID Register Structure used by PrimeCells. This defines two identification registers:

- A Component Identification Register, which indicates that the identification registers are present. It extends the PrimeCell specification by allowing a Component Class to be specified, which can in turn specify that additional registers are present. All components have PrimeCell identifiers.
- A Peripheral Identification Register, which uniquely identifies the component for a designer.

Each Component Class identified in the Component Identification Registers specifies further registers that might be present and implementation-defined.CoreSight components are part of either CoreSight class or ROM class. Figure 2-1 shows the CoreSight Class layout for a 4KB block, with the management registers identified in the upper 0.5KB of memory.

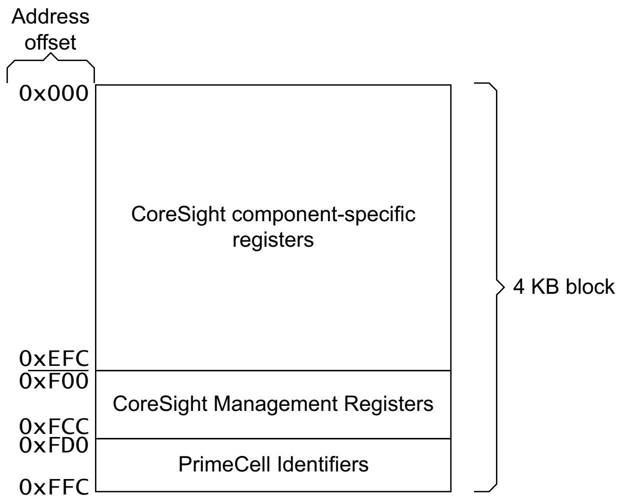


Figure 2-1 CoreSight component memory map

2.3 Component Identification Registers, 0xFF0-0xFFC

Table 2-1 shows the identification registers.

Table 2-1 Identification registers

Offset	Name	Bits	Value	Description
`0xFF0`	Component ID0	[7:0]	`0x0D`	Preamble
`0xFF4`	Component ID1	[3:0]	`0x0`	Preamble
		[7:4]	-	Component class [a]
`0xFF8`	Component ID2	[7:0]	`0x05`	Preamble
`0xFFC`	Component ID3	[7:0]	`0xB1`	Preamble

a. This field is used to indicate the classification of the 4KB memory block to which these identification registers bind. See Table 2-2 for the permitted values.

Table 2-2 shows the permitted values for component classes.

Table 2-2 Component class permitted values

Value	Class description
`0x0`	Reserved.
`0x1`	ROM table as defined in *Class 0x1, ROM table* on page 2-23.
`0x2-0x8`	Reserved.
`0x9`	CoreSight component. See *CoreSight component programmer's model* on page 2-3.
`0xA`	Reserved.
`0xB`	*Peripheral Test Block* (PTB)[a].
`0xC`	Reserved.
`0xD`	OptimoDE *Data Engine SubSystem* (DESS) component[a].
`0xE`	Generic IP component[a].
`0xF`	PrimeCell or system component with no standardized register layout. This is provided for backward compatibility.

a. Not used in CoreSight components.

2.4 Peripheral Identification Registers, 0xFD0-0xFEC

The read-only registers provide the following options of the peripheral:

- Part number
- Designer, JEP 106 code
- Revision information
- Customer modified field
- Memory footprint size.

This information is contained in the peripheral identification registers, Peripheral ID0-7. These registers are described in *Peripheral identification registers* on page 2-7.

2.4.1 Part number

This is selected by the designer of the component and must not conflict with any previously created components from that designer, unless this is an update to a component. Table 2-3 shows the values used in CoreSight components.

Table 2-3 CoreSight component part number values

CoreSight component	Value
CoreSight ETM9	`0x910`
CoreSight ETM11	`0x920`
HTM	`0x917`
CTI	`0x906`
CSTF	`0x908`
ETB	`0x907`
TPIU	`0x912`
ROM table	User defined, see *Class 0x1, ROM table* on page 2-23

2.4.2 JEP106

This indicates the designer of the component and not the implementer, except where the two are the same. JEDEC is an organization that maintains a document listing manufacturer codes, JEP106. To obtain a number contact JEDEC (`http://www.jedec.org`).

ARM has been allocated the 59th slot in bank 5, with `0x3B` being the JEDEC-defined 8-bit representation. This appears as a JEP106 code of `0x7F7F7F7F3B`. This is represented as:

- Peripheral ID4[3:0] is `0x4`, indicating the 5th bank
- Peripheral ID2[2:0] is `0x3`, bits [6:4] of the `0x3B`
- Peripheral ID1[7:4] is `0xB`, bits [3:0] of the `0x3B`.

Note

The ROM table does not have a defined value, and is given the JEP value of the implementor.

2.4.3 Revision

The Revision field is an incremental value starting at `0x0` for the first design of this component. This only increases by 1 for both major and minor revisions and is a look-up to establish the exact major or minor revision. The ROM table revision field value is implementation-defined.

2.4.4 Customer modified

This value must indicate whether the customer has modified the component from the delivered RTL. If no RTL modifications are allowed, this field is always zero. All CoreSight components have a value of zero.

2.4.5 RevAnd

CoreSight components must use the RevAnd bits only for the Tie-off RevAnd information. Modifications to this field reflect any minor changes, such as metal fixes, and not an RTL change. All CoreSight components have a value of zero.

2.4.6 4KB count

This is a four-bit value that indicates the total contiguous size of the memory window used by this device in powers of 2 from the standard 4KB. All CoreSight components have a value of zero, indicating that their memory size is 4KB. For more information on the value of this field, see the *CoreSight Architecture Specification*. For ROM tables, this value must be zero. See *ROM table registers* on page 2-23.

2.4.7 Peripheral identification registers

The peripheral identification registers are eight 8-bit registers that span address locations `0xFD0-0xFEC`. These registers are described in:

- *Peripheral ID0 Register, 0xFE0*
- *Peripheral ID1 Register, 0xFE4* on page 2-8
- *Peripheral ID2 Register, 0xFE8* on page 2-8
- *Peripheral ID3 Register, 0xFEC* on page 2-9
- *Peripheral ID4 Register, 0xFD0* on page 2-10
- *Peripheral ID5 Register, 0xFD4* on page 2-11
- *Peripheral ID6 Register, 0xFD8* on page 2-11
- *Peripheral ID7 Register, 0xFDC* on page 2-11.

Peripheral ID0 Register, 0xFE0

The Peripheral ID0 Register is hard-coded and the fields in the register determine the reset value.

Figure 2-2 shows the bit assignments.

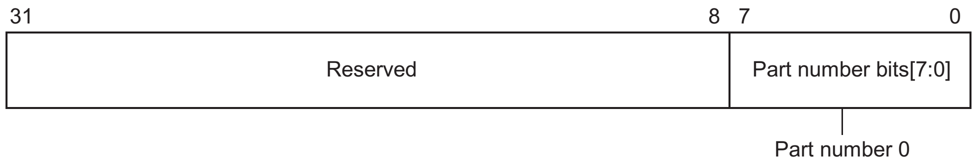


Figure 2-2 Peripheral ID0 Register bit assignments

Table 2-4 shows the bit assignments.

Table 2-4 Peripheral ID0 Register bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	Reserved RAZ.
[7:0]	Part number[7:0]	RO	See Table 2-3 on page 2-5.

Peripheral ID1 Register, 0xFE4

The Peripheral ID1 Register is hard-coded and the fields in the register determine the reset value.

Figure 2-3 shows the bit assignments.

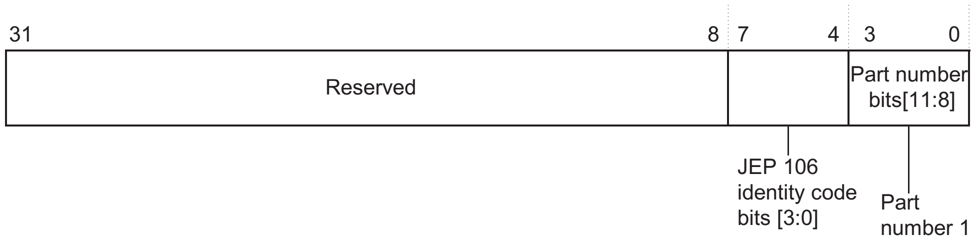


Figure 2-3 Peripheral ID1 Register bit assignments

Table 2-5 shows the bit assignments.

Table 2-5 Peripheral ID1 Register bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	Reserved RAZ.
[7:4]	JEP identity code [3:0]	RO	`0xB`, except for the ROM table. See *JEP106* on page 2-5.
[3:0]	Part number 1	RO	See *Part number* on page 2-5.

Peripheral ID2 Register, 0xFE8

The Peripheral ID2 Register is hard-coded and the fields in the register determine the reset value.

Figure 2-4 on page 2-9 shows the bit assignments.

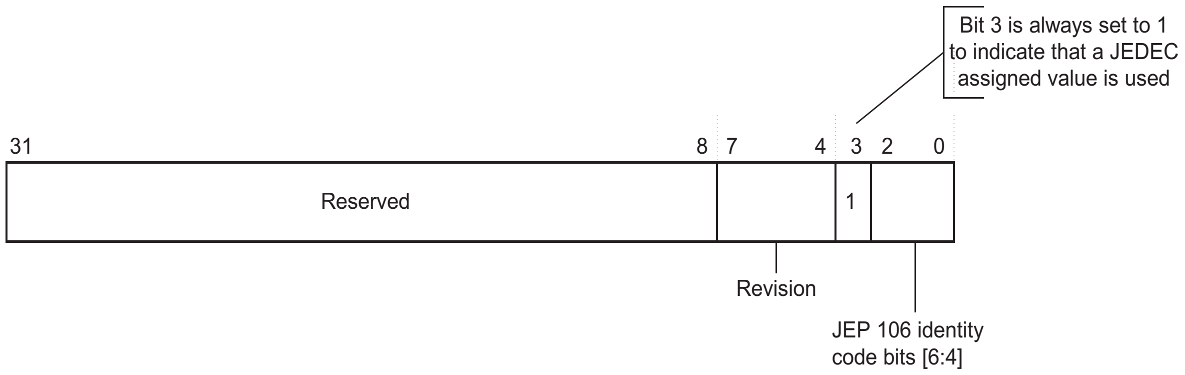

Bit 3 is always set to 1 to indicate that a JEDEC assigned value is used
31
8
7
4
3
2
0
Reserved
1
Revision
JEP 106 identity code bits [6:4]

Figure 2-4 Peripheral ID2 Register bit assignments

Table 2-6 shows the bit assignments.

Table 2-6 Peripheral ID2 Register bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	Reserved RAZ.
[7:4]	Revision	RO	See *Revision* on page 2-6.
[3]	-	RO	Always `0x1`, indicating a JEP106 value used.
[2:0]	JEP106 Identity [6:4]	RO	`0x3`, except for the ROM table. See *JEP106* on page 2-5.

Peripheral ID3 Register, 0xFEC

The Peripheral ID3 Register is hard-coded and the fields in the register determine the reset value.

Figure 2-5 shows the bit assignments.

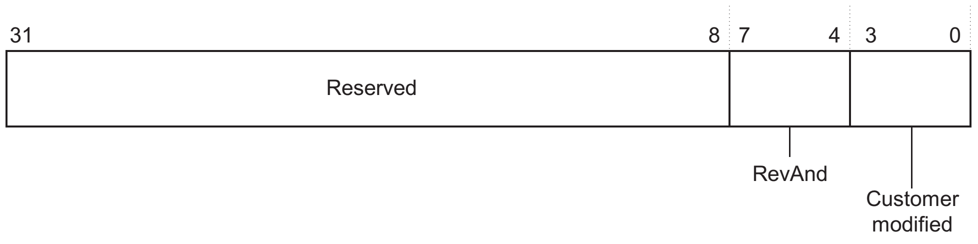


Figure 2-5 Peripheral ID3 Register bit assignments

Table 2-7 shows the bit assignments.

Table 2-7 Peripheral ID3 Register bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	Reserved RAZ.
[7:4]	RevAnd	RO	Read as `0x0` for all CoreSight components. See *RevAnd* on page 2-6.
[3:0]	Customer modified	RO	Read as `0x0` for all CoreSight components.

Peripheral ID4 Register, 0xFD0

The Peripheral ID4 Register is hard-coded and the fields in the register determine the reset value.

Figure 2-6 shows the bit assignments.

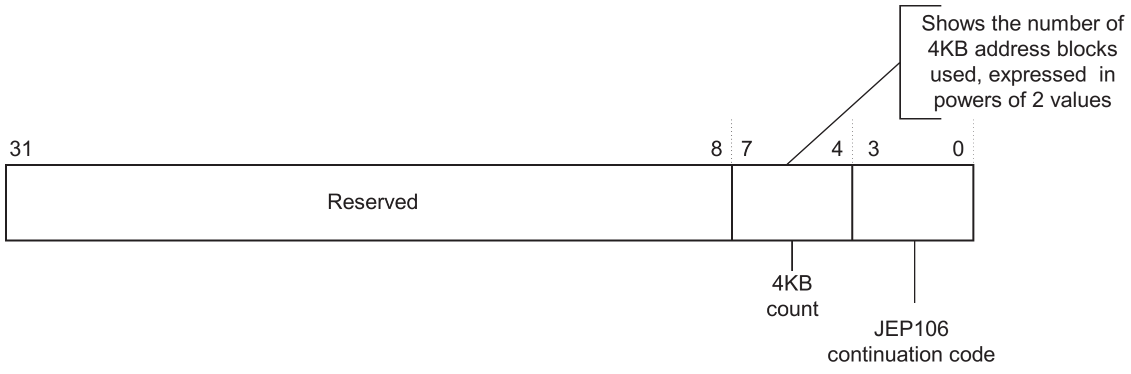

Shows the number of 4KB address blocks used, expressed in powers of 2 values
31
8
7
4
3
0
Reserved
4KB count
JEP106 continuation code

Figure 2-6 Peripheral ID4 Register bit assignments

Table 2-8 shows the bit assignments.

Table 2-8 Peripheral ID4 Register bit assignments

Bits	Name	Type	Function
[31:8]	Reserved	-	Reserved RAZ.
[7:4]	4KB count	RO	Read as `0x0` for all CoreSight components. See *4KB count* on page 2-6.
[3:0]	JEP continuation code	RO	Read as `0x4` for all CoreSight components, except the ROM table. See *JEP106* on page 2-5.

Peripheral ID5 Register, 0xFD4

The Peripheral ID5 Register is unused and is reserved for future use.

Peripheral ID6 Register, 0xFD8

The Peripheral ID6 Register is unused and is reserved for future use.

Peripheral ID7 Register, 0xFDC

The Peripheral ID7 Register is unused and is reserved for future use.

2.5 Class 0x9, CoreSight Component

The CoreSight component registers are described in:

- *CoreSight Management Registers summary*
- *Integration Mode Control Register, ItCtrl, 0xF00* on page 2-15
- *Claim tag registers* on page 2-16
- *Lock registers* on page 2-17
- *Authentication Status Register, AuthStatus, 0xFB8* on page 2-19
- *Device configuration information* on page 2-20.

2.5.1 CoreSight Management Registers summary

Table 2-9 shows the CoreSight Management Registers.

Table 2-9 CoreSight Management Registers

Offset	Name	Type	Bits	Reset value	Description
CoreSight management registers					
`0xF00`	ItCtrl	R/W	[0]	`0x0`	Integration Mode Control. Changes the component from a functional mode to allow for topology and integration checking. See *Integration Mode Control Register, ItCtrl, 0xF00* on page 2-15.
`0xF04-0xF9C`	-	-	-	-	Reserved RAZ/SBZP.
`0xFA0`	ClaimSet	R/W	[n-1:0]	`0xFF` ETM `0xF` HTM and other components	Claim Tag Set. Claims set by writing to this register. n=8 for ETM. n=4 HTM and other CoreSight components. See *Claim tag registers* on page 2-16.
`0xFA4`	ClaimClr	R/W	[n-1:0]	Zero	Claim Tag Clear. n=8 for ETM. n=4 for HTM and other CoreSight components. Read returns current claims, and resets to zero. See *Claim tag registers* on page 2-16.
`0xFA8-0xFAC`	-	-	-	-	Reserved RAZ/SBZP.

Table 2-9 CoreSight Management Registers (continued)

Offset	Name	Type	Bits	Reset value	Description
`0xFB0`	LockAccess	WO	[31:0]	-	Lock Access. Writeable location to obtain write access to all functional registers within a component. See *Lock registers* on page 2-17.
`0xFB4`	LockStatus	RO	[2:0]	-	Lock Status. Indicates presence and state of a lock mechanism. `0x0` when **PADDRDBG31** is HIGH. `0x3` when **PADDRDBG31** is LOW. See *Lock registers* on page 2-17.
`0xFB8`	AuthStatus	RO	[7:0]	-	Authentication Status. Four pairs of bits indicating security levels. See *Authentication Status Register, AuthStatus, 0xFB8* on page 2-19.
`0xFBC-0xFC4`	-	-	-	-	Reserved RAZ/SBZP.
`0xFC8`	DevId	RO	-	-	Component Configuration. Implementation specific register identifying resource and configuration. See *Device configuration information* on page 2-20.
`0xFCC`	DevType	RO	[7:0]	-	Device Type Identifier. Enables basic device identification. See *Device Type Identifier Register, DevType, 0xFCC* on page 2-21.
Peripheral identification registers					
`0xFD0`	Peripheral ID4	RO	[7:4]	0	4KB Count. Number of 4KB address blocks used, in powers of 2. All CoreSight blocks occupy 4KB only. See *Component Identification Registers, 0xFF0-0xFFC* on page 2-4.
		RO	[3:0]	4	JEP106 continuation code. See *JEP106* on page 2-5.
`0xFD4`	Peripheral ID5	RO	[7:0]	`0x00`	Reserved for future use.
`0xFD8`	Peripheral ID6	RO	[7:0]	`0x00`	Reserved for future use.

Table 2-9 CoreSight Management Registers (continued)

Offset	Name	Type	Bits	Reset value	Description
`0xFDC`	Peripheral ID7	RO	[7:0]	`0x00`	Reserved for future use.
`0xFE0`	Peripheral ID0	RO	[7:0]	-	PartNumber0. Middle and lower BCD value of Device number. See *Part number* on page 2-5.
`0xFE4`	Peripheral ID1	RO	[7:4]	`0xB`	JEP106 identity code [3:0]· (ARM). See *JEP106* on page 2-5.
		RO	[3:0]	-	PartNumber1. Upper BCD value of Device number. See *Part number* on page 2-5.
`0xFE8`	Peripheral ID2	RO	[7:4]	-	Revision. Revision number of Peripheral, starting from `0x0`. See *Revision* on page 2-6.
		RO	[3]	`0x1`	Always set. Indicates that a JEDEC assigned value is used. See *JEP106* on page 2-5.
		RO	[2:0]	`0x3`	JEP106 identity code [6:4], ARM. See *JEP106* on page 2-5.
`0xFEC`	Peripheral ID3	RO	[7:4]	`0x0`	RevAnd, at top level. See *RevAnd* on page 2-6.
		RO	[3:0]	`0x0`	Customer Modified number. Should be `0x0` when supplied. See *Customer modified* on page 2-6.
`0xFF0`	Component ID0	RO	[7:0]	`0x0D`	Preamble.

`0xFF4`	Component ID1	RO	[7:4]	`0x9`
				`0x1`
					CoreSight Component class.
					See *Class 0x1, ROM table* on page 2-23.

		RO	[3:0]	`0x0`	Preamble.
`0xFF8`	Component ID2	RO	[7:0]	`0x05`	Preamble.
`0xFFC`	Component ID3	RO	[7:0]	`0xB1`	Preamble.

2.5.2 Integration Mode Control Register, ItCtrl, 0xF00

The Integration Mode Control Register enables the component to switch from a functional mode, default behavior, into integration mode where the inputs and outputs of the component can be directly controlled for the purpose of integration testing or topology solving. All CoreSight components have integration registers to enable integration testing and topology detection.

Note

- Topology detection and topology solving are described in the *CoreSight Architecture Specification.*
- Integration testing is described in the applicable CoreSight Integration Manual.

Figure 2-7 shows the bit assignments.

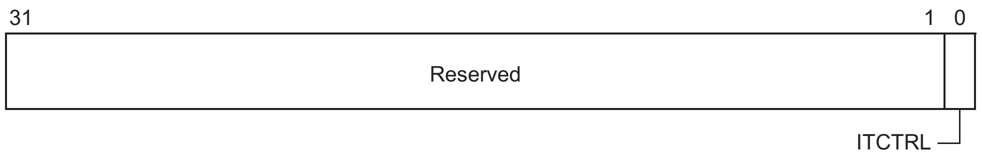


Figure 2-7 Integration Mode Control Register bit assignments

Table 2-10 shows the bit assignments.

Table 2-10 Integration Mode Control Register bit assignments

Bits	Description
[31:1]	Reserved RAZ/SBZP.
[0]	If set, the component reverts to an integration mode to enable topology detection or to enable integration testing. When no integration functionality has been put into a component, because it is not required, writing a HIGH to this location must be ignored and must return a LOW on read. At reset integration functionality is disabled.

2.5.3 Claim tag registers

This section describes the claim tag registers:

- *Claim Tag Set Register, ClaimSet, 0xFA0*
- *Claim Tag Clear Register, ClaimClr, 0xFA4*.

Claim Tag Set Register, ClaimSet, 0xFA0

The Claim Tag Set Register is used with the Claim Tag Clear Register. This register returns the number of bits that can be set on a read, and enables individual bits to be set on a write.

All CoreSight components implement a minimum of a 4-bit claim tag (n=4). CoreSight ETMs implement an 8-bit claim tag field.

Table 2-11 shows the bit assignments.

Table 2-11 Claim Tag Set Register bit assignment

Bits	Description
[31:n]	Reserved RAZ/SBZP. Unimplemented locations return a zero on read.
[n-1:0]	A bit-programmable register bank that reads the *Claim Tag Value* (CTV) Reads: A read returns `32'h000000FF` indicating that this claim tag is eight bits wide. `32'h0000000F` indicates that this claim tag is four bits wide. Logic 1 indicates that this bit can be set. Logic 0 indicates that this bit is unimplemented, that is, it cannot be set. Writes: Bit-clear has no effect on the CTV. Bit-set marks the bit in the CTV.

Claim Tag Clear Register, ClaimClr, 0xFA4

The Claim Tag Clear Register is used in conjunction with the Claim Tag Set Register. See *Claim Tag Set Register, ClaimSet, 0xFA0*.

Table 2-12 shows the bit assignments.

Table 2-12 Claim Tag Clear Register bit assignments

Bits	Description
[31:n]	Reserved RAZ/SBZP.
[n-1:0]	A bit-programmable register bank that sets the CTV. Reads: A read returns a value indicating the current claim value. Writes: Bit-clear has no effect on CTV. Bit-set removes the bit in the CTV. This is 0 on reset.

For more information about the Claim Tag Registers see the *CoreSight Architecture Specification*.

2.5.4 Lock registers

This section describes the lock registers:

- *Lock Access Register, LockAccess, 0xFB0*
- *Lock Status Register, LockStatus, 0xFB4* on page 2-18.

Lock Access Register, LockAccess, 0xFB0

The Lock Access Register enables a write access to component registers.

Table 2-13 shows the bit assignments.

Table 2-13 Lock Access Register bit assignments

Bits	Description
[31:0]	Write Access Code. A write of `0xC5ACCE55` enables further write access to this component. An invalid write has the effect of removing write access. Reserved RAZ/SBZP.

All CoreSight components implement a 32-bit Lock Access Register value indicated as present in bit 0 of the Lock Status Register.

Note

If your system has a mixture of 8-bit and 32-bit Lock Access Registers, it is possible to use the same routine by writing the value `0xC5ACCE55` repeated with different amounts of rotation. This causes either the 32-bit lock access register or the 8-bit version to unlock. See the *CoreSight Architecture Specification* for more information.

Lock Status Register, LockStatus, 0xFB4

The Lock Status Register indicates the status of the lock control mechanism. The Lock Status Register is used with the Lock Access Register.

Figure 2-8 shows the bit assignments.

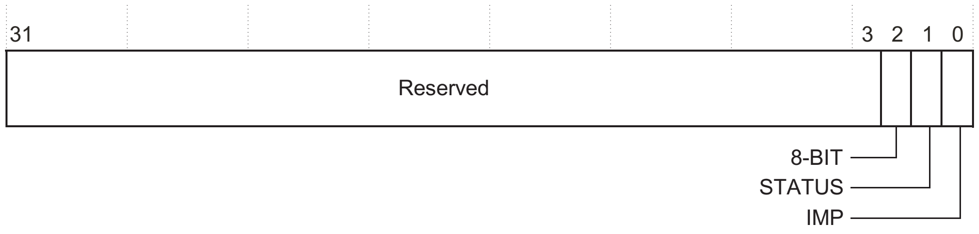


Figure 2-8 Lock Status Register bit assignments

Table 2-14 shows the bit assignments.

Table 2-14 Lock Status Register bit assignments

Bits	Description
[31:3]	Reserved RAZ/SBZP.
[2]	1 = device implements an 8-bit lock register. 0 = device implements a 32-bit lock register.
[1]	Lock status. 1 = device write access is locked. 0 = device write access is granted.
[0]	Indicates that a lock control mechanism exists for this component. 1 = lock mechanism is implemented. 0 = lock mechanism is not implemented.

Note

All CoreSight components implement 32-bit lock access registers but they are only present on accesses when **PADDRDBG31** is LOW. **PADDRDBG31** is used to bypass the lock control mechanism and must be connected to **PADDRDBG[31]** which is only HIGH on accesses from external tools. For more information about the Lock Access Register and **PADDRDBG[31]** see the *CoreSight Architecture Specification*.

2.5.5 Authentication Status Register, AuthStatus, 0xFB8

The Authentication Status Register reports the required security level and current status of the security enable bit pairs. Where functionality changes on a given security level, this change is reported in this register.

See individual CoreSight components for specific values. Where no values are specified this register reads as zero.

Figure 2-9 shows the bit assignments.

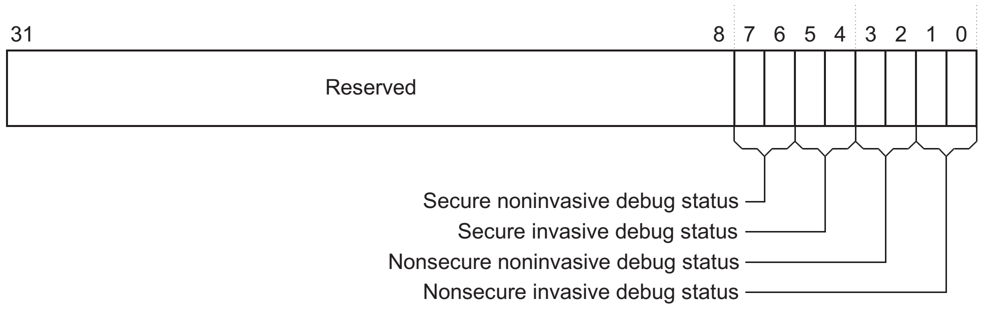

31
8
7
6
5
4
3
2
1
0
Reserved
Secure noninvasive debug status
Secure invasive debug status
Nonsecure noninvasive debug status
Nonsecure invasive debug status

Figure 2-9 Authentication Status Register bit assignments

Table 2-15 shows the bit assignments.

Table 2-15 Authentication Status Register bit assignments

Bits	Description
[31:8]	Reserved RAZ/SBZP
[7:6]	Secure Noninvasive Debug
[5:4]	Secure Invasive Debug
[3:2]	Nonsecure Noninvasive Debug
[1:0]	Nonsecure Invasive Debug

The description for each pair of bits in each debug set is shown in Table 2-16.

Table 2-16 Authentication Status Register pairs

Value	Description
2'b00	Functionality not implemented or controlled elsewhere.
2'b01	Reserved.
2'b10	Functionality disabled.
2'b11	Functionality enabled.

See the *CoreSight Architecture Specification* for more information.

2.5.6 Device configuration information

This section describes:

- *Device ID Register, DevId, 0xFC8*
- *Device Type Identifier Register, DevType, 0xFCC* on page 2-21.

Device ID Register, DevId, 0xFC8

The Device ID Register is implementation defined for each Part Number/Designer. This indicates the capabilities of the device. The entire 32-bit field can be used because the data width is determined by the particular device, which is 32 bits for all CoreSight components. Unused or surplus bits read as zero.

If the design has configurable code, this register reflects any changes to a standard configuration, irrespective of the configurable code existing internally or externally to the block, such as external multiplexors for example.

Table 2-17 shows the bit assignments.

Table 2-17 Device ID Register bit assignments

Bits	Description
[31:0]	Implementation Defined. An eight-bit device can implement the lower eight bits only.

See individual CoreSight components for the definition of this register.

Device Type Identifier Register, DevType, 0xFCC

The Device Type Identifier Register enables devices to be identified by their CoreSight Class.

Figure 2-10 shows the bit assignments.

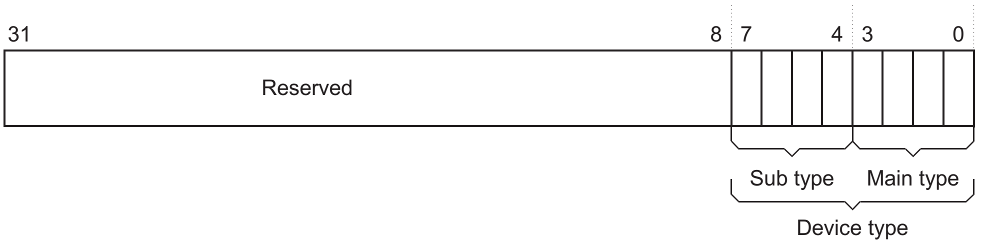


Figure 2-10 Device Type Identifier Register bit assignments

Table 2-18 shows the bit assignments.

Table 2-18 Device Type Identifier Register bit assignments

Bits	Description
[31:8]	Reserved RAZ/SBZP.
[7:4]	Sub type, for trace sources this is broken down into cores, DSPs or buses.
[3:0]	Main type/class.

The classifications are defined to enable any interrogating tools to obtain more information about the device in the absence of understanding the Part Number field. This information can then be reported back to the user of the debugger.

Table 2-19 shows the main class and sub type values for the CoreSight components.

Table 2-19 CoreSight main class and sub type values

Main class [3:0]		Sub type [7:4]	
Value	Description	Value	Description
`0x4`	Debug Control	`0x1`	Cross Trigger Matrix, CTI
`0x3`	Trace Source	`0x4`	HTM, stimulus from bus activity
		`0x1`	ETM, associated with processor or core

Table 2-19 CoreSight main class and sub type values (continued)

Main class [3:0]		Sub type [7:4]	
Value	**Description**	**Value**	**Description**
`0x2`	Trace Link	`0x1`	Trace Funnel
`0x1`	Trace Sink	`0x2`	ETB
		`0x1`	TPIU

Note

The Replicator and Synchronous 1:1 Bridge do not appear in Table 2-19 on page 2-21 because they do not have programmable interfaces.

For the current list of all recognizable components see the *CoreSight Architecture Specification*.

2.6 Class 0x1, ROM table

A ROM table must be present in all CoreSight systems, and must contain a list of components in the system.

2.6.1 ROM table registers

The ROM table registers are described in the following sections:

- *Component Identification Register, 0xFF0-0xFFC*
- *Peripheral Identification Register, 0xFD0-0xFEC*.

Component Identification Register, 0xFF0-0xFFC

The value is `0xB105100D`, spread over the lower byte of each word. For more details, see *Component Identification Registers, 0xFF0-0xFFC* on page 2-4.

Peripheral Identification Register, 0xFD0-0xFEC

The ROM table has a specific PrimeCell class as specified in Table 2-2 on page 2-4.

The Peripheral Identification Register uniquely identifies the SoC or platform. If any of the components pointed to by the ROM are changed or any of their connections are changed, this register must be updated. Where a ROM table is implemented as part of a standard component, the following fields must be available for customer modification:

- JEP106 continuation code, 4 bits
- Customer modified, 4 bits
- Revision, 4 bits
- JEP106 identity code, 7 bits
- PartNumber, 12 bits.

The Peripheral ID is used by the debugger to name the description of the chip. When topology detection is performed, the description of the chip is saved with the Peripheral ID. If that chip is subsequently connected to the debugger, it reads the Peripheral ID and retrieves the saved description.

If two different chips have the same Peripheral ID, the debugger might retrieve the wrong description. In these circumstances, the user must request that the debugger perform topology detection again. See *Peripheral Identification Registers, 0xFD0-0xFEC* on page 2-5 for more information on Peripheral ID registers.

Reserved RAZ/SBZ, 0xF00-0xFE0

This region is reserved for future expansion.

2.6.2 ROM entries, 0x000 onwards

There are two formats for ROM entries, 8-bit and 32-bit. All entries must be of the same format.

ROM table entries eight-bit format

In the eight-bit format, each entry takes 16 bytes of address space as shown in Table 2-20.

Table 2-20 ROM table entry eight-bit format

Offset in entry	Bits	Description
`0xC`	[7:0]	Entry[31:24]
`0x8`	[7:0]	Entry[23:16]
`0x4`	[7:0]	Entry[15:8]
`0x0`	[7:0]	Entry[7:0]

ROM table entries 32-bit format

In the 32-bit format, each entry takes four bytes of address space as shown in Table 2-21.

Table 2-21 32-bit ROM table entries

Offset in entry	Bits	Description
`0x0`	[31:0]	Entry[31:0]

ROM table entries take the format shown in Table 2-22.

Table 2-22 ROM table entry format

Bits	Name	Description
[31:12]	Address offset	Base address of the component, relative to the ROM address. Negative values are permitted using two's complement.
[11:2]	-	Reserved RAZ.
[1]	Format	1 = 32-bit format 0 = 8-bit format
[0]	Entry present	Always set. The last entry has the value `0x00000000`, which is reserved.

ROM table hierarchy

Each ROM table entry can point to CoreSight components or other ROM tables. If it points to another ROM table, that ROM table must be read for CoreSight components, and that ROM table itself can have entries for more ROM tables.

The ID of a ROM table can be set:

- to a value representing the subsystem it describes, enabling that subsystem to be implemented on its own in the future
- to a value reserved by an implementor for such ROM tables
- to the ID of the parent ROM.

These are only examples of the values that might be used for the Peripheral ID Registers of ROM tables.

See the *CoreSight Architecture Specification* for more information.

2.7 Locating the ROM table

Locating the ROM table and ROM table entries is described in the following sections:

- *Location from the Debug Access Port (DAP)*
- *Location from ARM Cortex cores*
- *Location from other ARM cores*
- *Location from other cores*.

2.7.1 Location from the Debug Access Port (DAP)

Each memory *Access Port* (AP), such as AHB-AP or APB-AP, contains a register that provides a pointer entry. This register can:

- indicate the base address of a ROM table
- indicate the base address of a single CoreSight component, which must be the only CoreSight component accessible from that access port
- indicate that no CoreSight components are directly accessible from this access port by returning `0FFFFFFFF`.

2.7.2 Location from ARM Cortex cores

See the *ARM v7 Architecture Specification* for more information.

2.7.3 Location from other ARM cores

The operating system or debug monitor must incorporate knowledge of the memory map of the system into its software so that it can find the ROM table.

2.7.4 Location from other cores

This is implementation-defined.

Chapter 3
Debug Access Port

This chapter describes the *Debug Access Port* (DAP). The DAP provides multiple master driving ports, all accessible and controlled through a single external interface port to provide system-wide debug. It contains the following sections:

- *JTAG-AP byte command protocol* on page 3-122
- *JTAG-AP clock domains* on page 3-126
- *ROM table* on page 3-127
- *Connections to CoreSight components and system interfaces* on page 3-129
- *Authentication requirements for Debug Access Port* on page 3-130
- *Clocks and resets* on page 3-131
- *DAP transfer aborting* on page 3-132.

3.1 About the Debug Access Port

Components that access the DAP are called *Debug Ports* (DPs), and components that access internal system interfaces are called *Access Ports* (APs).

The debug port and access ports together are referred to as the *Debug Access Port* (DAP).

The DAP provides real-time access for the debugger without halting the core to:

- AMBA system memory and peripheral registers
- All debug configuration registers.

The DAP also provides debugger access to JTAG scan chains. Figure 3-1 shows the functional blocks of the DAP.

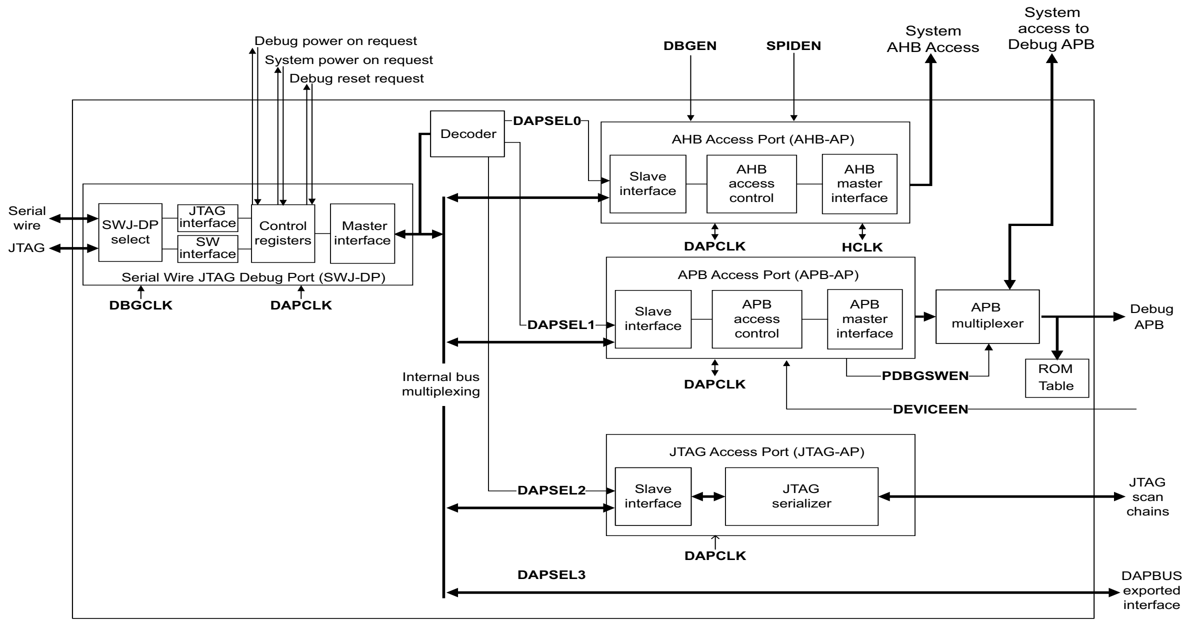


Figure 3-1 Structure of the CoreSight DAP components

The DAP enables debug access to the complete SoC using a number of master ports. Access to the CoreSight Debug APB is enabled through the *APB Access Port* (APB-AP) and *APB MUltipleXor* (APB-MUX), and system access through the *AHB Access Port* (AHB-AP).

The DAP comprises the following interface blocks:

- External debug access using the *Serial Wire JTAG Debug Port* (SWJ-DP). The SWJ-DP enables selection of either:
  - — external serial wire access using the *Serial Wire Debug Port* (SW-DP)
  - — external JTAG access using the *JTAG Debug Port* (JTAG-DP).
- Internal system access using:
  - — *AHB Access Port* (AHB-AP)
  - — *APB Access Port* (APB-AP)
  - — *JTAG Access Port* (JTAG-AP)
  - — DAPBUS exported interface.
- An APB multiplexor 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. The ROM table indicates the position of all CoreSight components in a system and assists in topology detection. See the *CoreSight Architecture Specification* for more information on topology detection.

The access ports specified for CoreSight are:

AHB Access Port (AHB-AP)

The AHB-AP provides an AHB-Lite master for access to a system AHB bus.

APB Access Port (APB-AP)

The APB-AP provides an APB in AMBA v3.0 master for access to the Debug APB bus.

JTAG Access Port (JTAG-AP)

The JTAG-AP provides JTAG access to on-chip components, operating as a JTAG master port to drive JTAG chains throughout the ASIC.

DAPBUS exported interface

Used for external connection to access port in certain processors.

3.1.1 DAP flow of control

Figure 3-2 on page 3-5 shows the flow of control for the DAP when used with an off-chip debugging unit such as RealView ICE.

The DAP, as a whole, acts as a component to translate data transfers from one type of interface, the external JTAG or serial wire link from tools, to different internal transactions. The debug port receives JTAG or serial wire transfers but controls the JTAG-AP, AHB-AP, and APB-AP through a standard bus interface. JTAG-AP receives these bus transactions and translates them into JTAG instructions for control of any connected TAP controllers such as a core for example. The AHB-AP is a bus master, along with any connected cores, on the system AHB Matrix that can access slaves connected to that bus, shared memory for example. The APB-AP can only access the Debug APB but control is also possible from the AHB Matrix, through the APB-Mux, resulting in control and access of various CoreSight components.

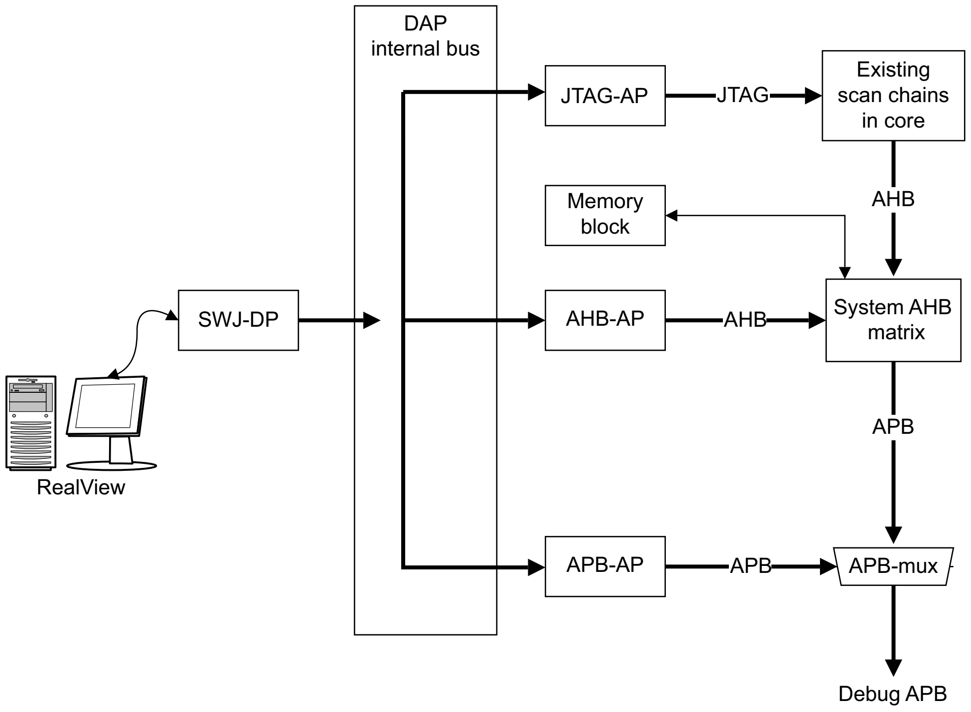


Figure 3-2 DAP flow of control

The external hardware tools, for example RealView, directly communicate with the SWJ-DP in the DAP and perform a series of operations to the debug port. Some of these accesses result in operations being performed on the DAP internal bus.

The DAP internal bus implements memory mapped access to the components which are connected using parallel address, read data and write data buses.

The debug port, SWJ-DP, is the bus master and initiates transactions on the DAP internal bus in response to some of the transactions that are received over the debug interface.

Debug interface transfers are memory mapped to registers in the DAP, both the bus master (debug port) and the slaves (access ports) contain registers. This DAP memory map is independent of the memory maps that exist within the target system.

Some of the registers in the access ports can translate interactions into transfers on the interconnects that they are connected to, such as JTAG, AHB, and APB. For example, in the JTAG-AP a number of registers are allocated for reading and writing commands that result in *Test Access Port* (TAP) instructions on connected devices, for example, a core. The core is also a bus master on a system memory structure to which the AHB-AP has access, so both the core and AHB-AP have access to shared memory devices, or other bus slave components.

3.2 DAP internal bus

The internal bus, the DAP bus, supports the connection of a debug port to multiple access ports. It operates in accordance with the ARM AMBA3™ APB Protocol with the addition of DAP transfer aborting. The following sections describe accessible features of the internal bus:

- *DAP access port address decoding*
- *Bus interconnection* on page 3-9
- *DAP transfer aborting* on page 3-132.

3.2.1 DAP access port address decoding

Figure 3-3 shows the encoding to select access ports and individual registers in each access port. Each access port has an addressable register bank. Accesses are word-aligned.

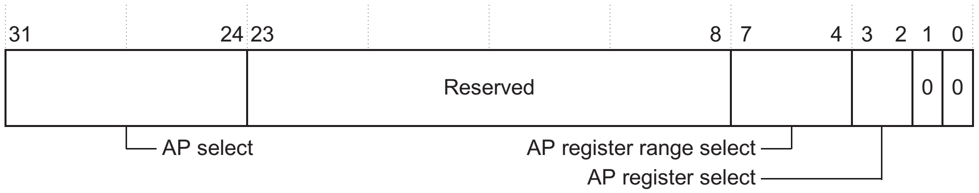


Figure 3-3 DAP internal bus address encoding

Table 3-1 shows the bit assignments.

Table 3-1 DAP bus address bit assignments

Bits	Name	Size	Description
[31:24]	AP Select	8	Used by DAP decoder for access port **DAPSELx** generation. AP Select is an 8-bit binary encoded field supporting up to a maximum of 256 access ports. `0xFF-0x04` = Default slave, as described in *Default slave* on page 3-8 `0x03` = DAPBUS exported interface `0x02` = JTAG-AP `0x01` = APB-AP `0x00` = AHB-AP.
[23:8]	-	-	Reserved SBZ

Table 3-1 DAP bus address bit assignments (continued)

Bits	Name	Size	Description
[7:4]	AP Reg Rng Select	4	AP Register Range Select Bits. AP Reg Rng Select provides access to 16 different banks of the 4 registers, accessible with AP Reg Select. This provides a total of 64 unique locations per access port.
[3:2]	AP Reg Select	2	Two bits are used for AP Reg Select. These bits are used to differentiate the majority of transactions to the DAP, and form part of the *Access Port ACCess* (APACC) instruction in the debug port.
[1:0]	-	-	Reserved SBZ

AP Select selects an individual access port, and is used by the Decoder for the generation of **DAPSELx**. All unused DAP Decoder locations default to the selection of the default slave described in *Default slave*.

Registers in the access ports are divided into banks of four. The two AP Reg Select bits are scanned into the debug port in line with the data packet to be transferred to an access port. These bits can then be used to provide fast accesses to four different AP Register locations and form bits [3:2] on the DAP address bus, without having to reload in a new instruction with a full address every debug port access.

The AP Register banks are changed by writing to the address registers in the debug port. This updates the base address of AP Reg Rng Select. Each access port only has to decode **DAPADDR[7:2]** because each access port is limited to a 6-bit addressable range with a maximum of 64 word registers.

DAPBUS exported interface

The DAPBUS exported interface enables you to connect to the access port of processors that have a DAPBUS compliant debug interface, such as the Cortex-M3. You must set bits[31:24], APSelect, to 3 when accessing this port.

Default slave

If the memory map of a system does not define the full 256 access port address space then a default slave is required. The default slave is selected when an access is attempted to the empty areas of the memory map. The default slave is present in all DAP configurations that do not define all 256 access ports. Read accesses to undefined addresses return zero, and writes are ignored. A transfer to the default slave is a two-cycle transaction, with no wait states inserted.

Programmer's model

To determine the type of access port present at each memory location, access the Identification Register in each access port at address offset `0xFC`. A returned ID of `0x00000000` indicates the default slave, that is, no functional access port is present.

Bus interconnection

DAPRDATA, DAPREADY, and **DAPSLVERR** are multiplexed using a read data multiplexor to the debug port, using **DAPSELx** from the DAP Decoder as the controlling signal. Figure 3-4 on page 3-10 shows the bus interconnect.

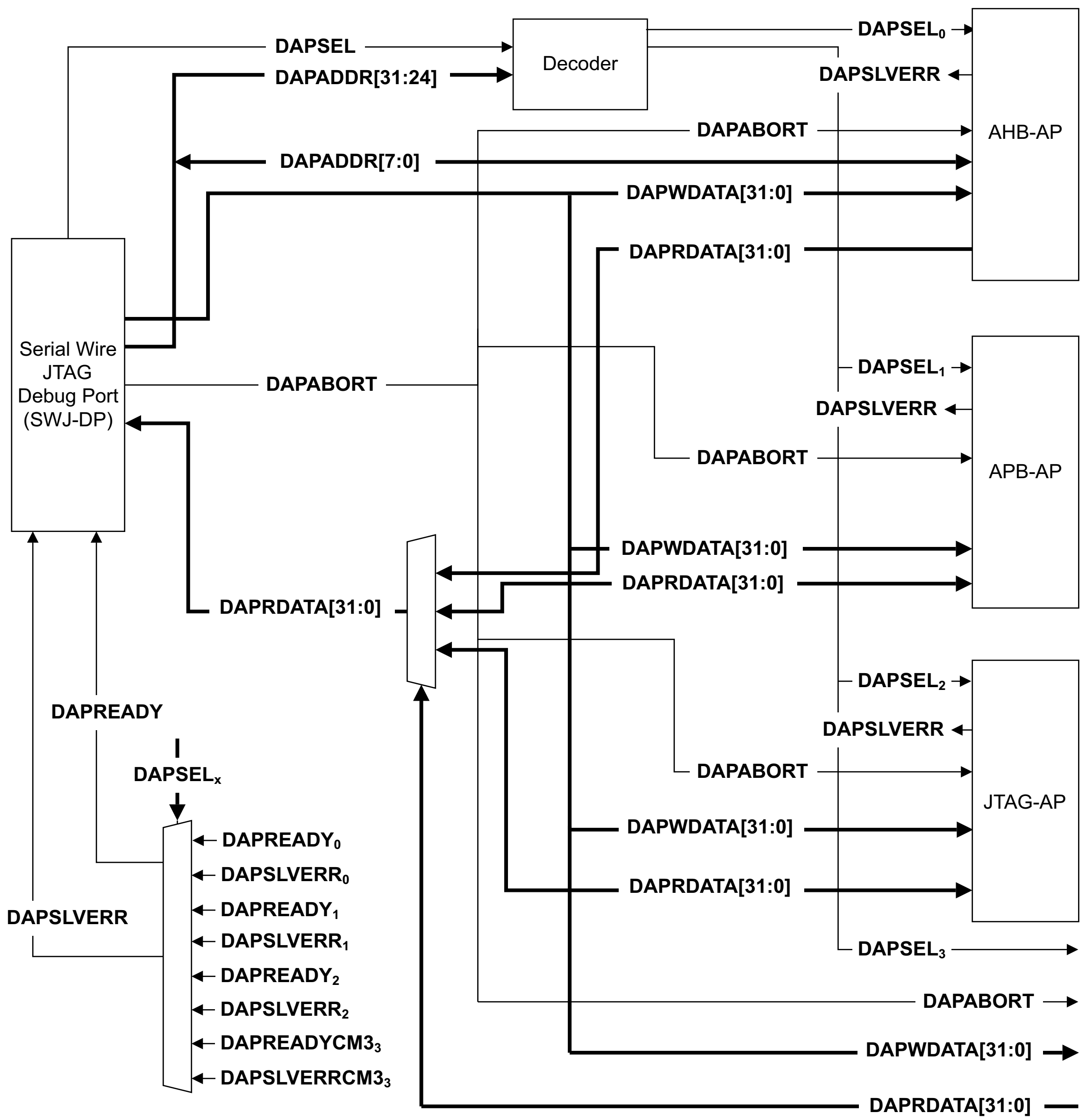


Figure 3-4 DAP internal bus interconnect

Note

In Figure 3-4 on page 3-10:

- **DAPREADYCM3** is connected to **DAPREADY$_3$**
- **DAPSLVERRCM3** is connected to **DAPSLVERR$_3$**
- **DAPSELCM3** is connected to **DAPSEL$_3$**.

3.2.2 Internal DAP bus signals

Internal DAP bus interface on page A-21 lists the DAP internal signals.

3.3 About the Debug Port

The debug port is the host tools interface to access the DAP. This interface controls any access ports provided within the DAP. The CoreSight DK supports two possible debug port implementations, JTAG and *Serial Wire Debug* (SWD), with a mechanism that supports switching between them:

- The JTAG-DP is based on the IEEE 1149.1 *Test Access Port* (TAP) and Boundary Scan Architecture, widely referred to as JTAG, and provides a JTAG interface to the DAP. For more information, see *JTAG-DP* on page 3-20,
- The SW-DP provides a two-pin (clock + data) interface to the DAP. For more information, see *SW-DP* on page 3-36.

The SWJ-DP provides the auto-detect logic that selects between JTAG and SWD. This enables the JTAG-DP and SW-DP to share the same pins. For more information, see *SWJ-DP* on page 3-13.

These alternative debug port implementations provide different mechanisms for debug access.

Note

Only one debug port can be used at once, and switching between the two debug ports must only be performed when neither debug port is in use.

3.4 SWJ-DP

SWJ-DP is a combined JTAG-DP and SW-DP that enables either a *Serial Wire Debug* (SWD) or JTAG probe to be connected to a target. It is the standard CoreSight debug port, and enables access either to the JTAG-DP or SW-DP blocks. To make efficient use of package pins, serial wire shares, or overlays, the JTAG pins, using an autodetect mechanism that switches between JTAG-DP and SW-DP, depending on which probe is connected. A special sequence on the **SWDITMS** pin is used to switch between JTAG-DP and SW-DP. The SWJ-DP behaves like a pure JTAG target if normal JTAG sequences are sent to it.

Note

The JTAG-DP and SW-DP are described in *JTAG-DP* on page 3-20 and *SW-DP* on page 3-36.

Figure 3-5 shows the external connections to the SWJ-DP.

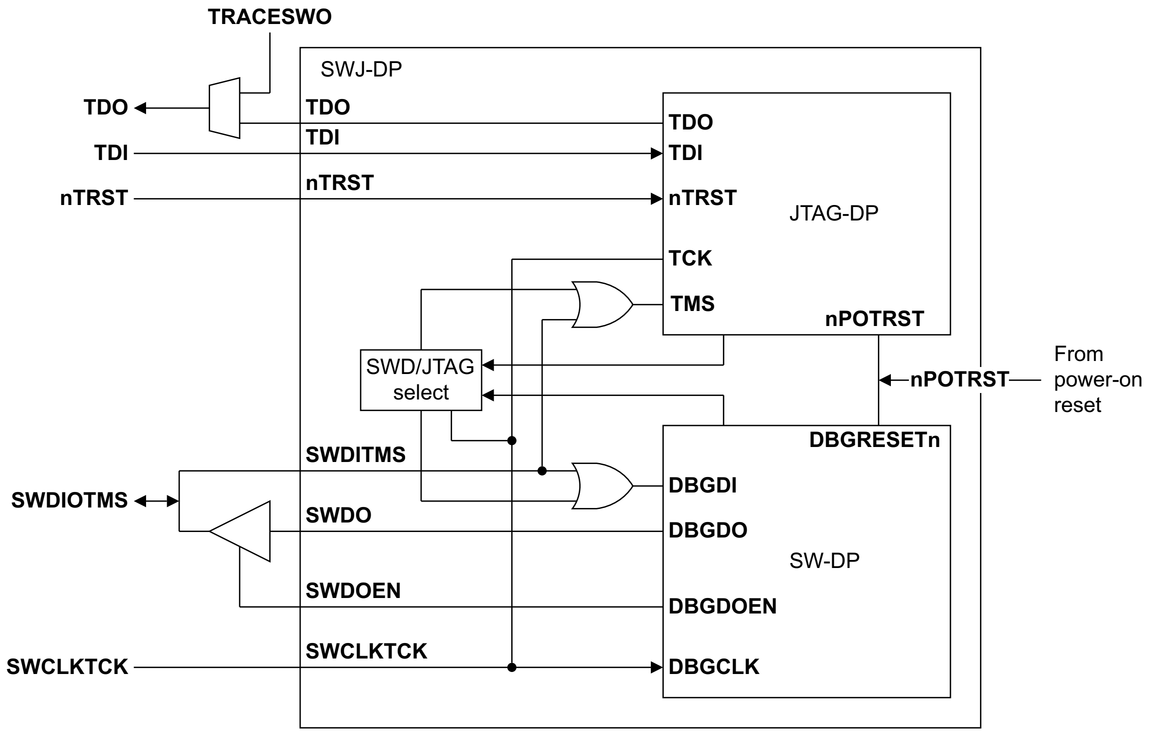


Figure 3-5 SWJ-DP external connections

The SWJ-DP is described in more detail in:

- *Structure*
- *Operation*
- *JTAG and SWD interface* on page 3-15
- *Clock and reset control interface* on page 3-15
- *SWD and JTAG select mechanism* on page 3-16.

3.4.1 Structure

The SWJ-DP consists of a wrapper around the JTAG-DP and SW-DP. Its function is to select JTAG or SWD as the connection mechanism and enable either JTAG-DP or SW-DP as the interface to the DAP.

3.4.2 Operation

SWJ-DP enables an *Application Specific Integrated Circuit* (ASIC) to be designed which can be used in systems that require either a JTAG interface or a SWD interface. There is a trade-off between the number of pins used and compatibility with existing hardware and test equipment. There are several scenarios where the use of a JTAG debug interface must be maintained:

- to enable inclusion in an existing scan chain, generally on-chip TAPs used for test or other purposes.
- to enable the device to be cascaded with legacy devices which use JTAG for debug
- to enable use of existing debug hardware with the corresponding test TAPs, for example, in *Automatic Test Equipment* (ATE).

An ASIC fitted with SWJ-DP support can be connected to legacy JTAG equipment without any requirement to make changes. If a SWD tool is available, only two pins are required, instead of the usual four used for JTAG. Two pins are therefore released for alternative use.

These two pins can only be used when there is no conflict with their use in JTAG mode. In addition, to support use of SWJ-DP in a scan chain with other JTAG devices, the default state after reset must be to use these pins for their JTAG function. If the direction of the alternative function is compatible with being driven by a JTAG debug device, the transition to a shift state can be used to transition from the alternative function to JTAG mode.

The alternate function cannot be used while the ASIC is being used in JTAG debug mode.

The switching scheme is arranged so that, provided there is no conflict on the **TDI** and **TDO** pins, a JTAG debugger is able to connect by sending a specific sequence.

The connection sequence used for SWD is safe when applied to the JTAG interface, even if hot-plugged, enabling the debugger to continually retry its access sequence. A sequence with `TMS=1` ensures that JTAG-DP, SW-DP, and the watcher circuit are in a known reset state. The pattern used to select SWD has no effect on JTAG targets.

SWJ-DP is compatible with a free-running **TCK**, or a gated clock which is supplied by the external tools.

3.4.3 JTAG and SWD interface

The external JTAG interface has four mandatory pins, **TCK**, **TMS**, **TDI**, and **TDO**, and an optional reset, **nTRST**. JTAG-DP and SW-DP also require a separate power-on reset, **nPOTRST**.

The external SWD interface requires two pins:

- a bidirectional **SWDIO** signal
- a clock, which can be input or output from the device.

The block level interface has two pins for data plus an output enable, which must be used to drive a bidirectional pad for the external interface, and clock and reset signals.

To enable sharing of the connector for either JTAG or SWD, connections must be made external to the SWJ-DP block, as shown in Figure 3-5 on page 3-13. In particular, **TMS** must be a bidirectional pin to support the bidirectional **SWDIO** pin in SWD mode.

When SWD mode is being used, the **TDO** pin is expected to be re-used for *Serial Wire Output* (SWO). The **TDI** pin is available for use as an alternative input function.

Note

If SWO functionality is required in JTAG mode, a dedicated pin is required.

3.4.4 Clock and reset control interface

In the **SWCLKTCK** clock domain, there are registers to enable power control for the on-chip debug infrastructure. This enables the majority of the debug logic, such as ETM and ETB, to be powered down by default, and only the serial engine has to be clocked. A debug session then starts by powering up the remainder of the debug components.

In SWJ-DP, either JTAG-DP or SW-DP can make power-up or reset requests but only if they are the selected device. Even in a system which does not provide a clock and reset control interface to the DAP, it is necessary to connect these signals so it appears that a clock and reset controller is present. This permits correct handshaking of the request and acknowledge signals to be performed.

3.4.5 SWD and JTAG select mechanism

SWJ-DP enables either a SWD or JTAG protocol to be used on the debug port. To do this, it implements a watcher circuit that detects a specific 16-bit select sequence on the SWDIOTMS pin:

- one 16-bit sequence is used to switch from JTAG to SWD operation
- a different 16-bit sequence is used to switch from SWD to JTAG.

The switcher defaults to JTAG operation on power-on reset and therefore the JTAG protocol can be used from reset without sending a select sequence.

Switching from one protocol to the other can only occur when the selected interface is in its reset state. JTAG must be in its *Test-Logic-Reset* (TLR) state and SWD must be in line-reset.

The SWJ-DP contains a mode status output, **JTAGNSW**, that is HIGH when the SWJ-DP is in JTAG mode and LOW when in SWD mode. This signal can be used to:

- disable other TAP controllers when the SWJ-DP is in SWD mode, for example, by disabling **TCK** or forcing **TMS** HIGH
- multiplex the Serial Wire output, **TRACESWO**, on to another pin such as **TDO** when in SWD mode.

An additional status output, **JTAGTOP**, indicates whether the JTAG-DP TAP controller is in one of the four top states. These states are:

- Test-Logic-Reset
- Run-Test/Idle
- Select-DR-Scan
- Select-IR-Scan.

This signal can be used in conjunction with **JTAGNSW** to control multiplexers so that, for example, **TDO** and **TDI** can be reused as *General Purpose Input/Output* (GPIO) signals when the device is in SWD mode.

The watcher block puts itself to sleep when it has finished tracking a specific sequence and only wakes up again when it detects the next reset condition. Figure 3-6 on page 3-17 is a simplified state diagram that shows how the watcher transitions between sleeping, detecting, and selection states.

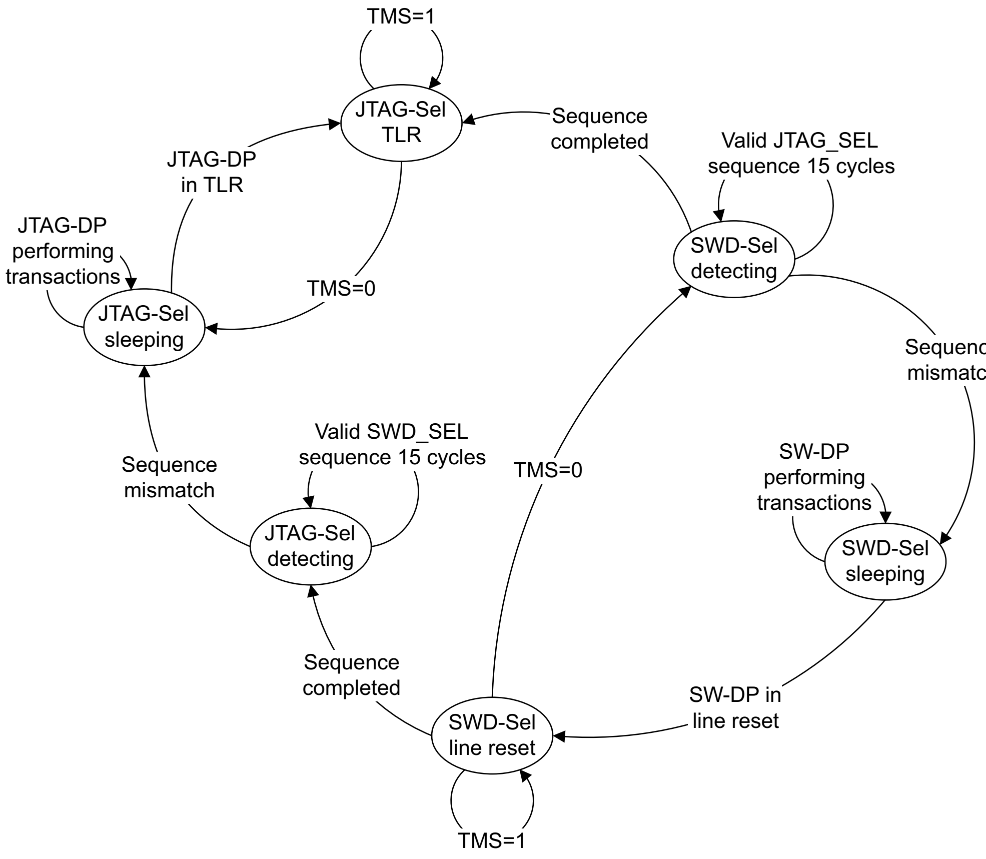


Figure 3-6 SWD and JTAG select state diagram

3.4.5.1 SWJ-DP programmer's model

The SWJ-DP programmer's model is described in:

- *JTAG to SWD switching* on page 3-18
- *SWD to JTAG switching* on page 3-18.

Note

An earlier version of SWJ-DP uses a different switching sequence which is now deprecated. See Appendix D *Deprecated SWJ-DP Switching Sequences* for details of this switching sequence.

JTAG to SWD switching

To switch SWJ-DP from JTAG to SWD operation:

- Send more than 50 **SWCLKTCK** cycles with SWDIOTMS=1. This ensures that both SWD and JTAG are in their reset states.
- Send the 16-bit JTAG-to-SWD select sequence on **SWDIOTMS**.
- Send more than 50 **SWCLKTCK** cycles with SWDIOTMS=1. This ensures that if SWJ-DP was already in SWD mode, before sending the select sequence, the SWD goes to line reset.
- Perform a READID to validate that SWJ-DP has switched to SWD operation.

The 16-bit JTAG-to-SWD select sequence is defined to be 0111100111100111, MSB first. This can be represented as 16'h79E7 transmitted MSB first or 16'hE79E when transmitted LSB first.

This sequence has been chosen to ensure that the SWJ-DP switches to using SWD whether it was previously expecting JTAG or SWD. As long as the 50 SWDIOTMS=1 sequence is sent first, the JTAG-to-SWD select sequence is benign to SW-DP, and is also benign to SWD and JTAG protocols used in the SWJ-DP, and any other TAP controllers that might be connected to **SWDIOTMS**.

SWD to JTAG switching

To switch SWJ-DP from SWD to JTAG operation:

- Send more than 50 **SWCLKTCK** cycles with SWDIOTMS=1. This ensures that both SWD and JTAG are in their reset states.
- Send the 16-bit SWD-to-JTAG select sequence on **SWDIOTMS**.
- Send at least 5 **SWCLKTCK** cycles with SWDIOTMS=1. This ensures that if SWJ-DP was already in JTAG mode, before sending the select sequence, that JTAG goes into the TLR state.
- Set the JTAG-DP IR to READID and shift out the DR to read the ID.

The 16-bit JTAG-to-SWD select sequence is defined to be 0011110011100111, MSB first. This can be represented as 16'h3CE7 transmitted MSB first or 16'hE73C when transmitted LSB first.

This sequence has been chosen to ensure that the SWJ-DP switches to using JTAG whether it was previously expecting JTAG or SWD. If the SWDIOTMS=1 sequence is sent first, the SWD-to-JTAG select sequence is benign to SW-DP, and is also benign to SWD and JTAG protocols used in the SWJ-DP, and any other TAP controllers that might be connected to **SWDIOTMS**.

3.4.5.2 Restriction on switching

It is recommended that when a system is powered up, a debug connection is made, and the mode is selected, either SWD or JTAG, the system remains in this mode throughout the debug session. Switching between modes should not be attempted while any component of the DAP is active.

Attempting to switch between modes while any component of the DAP is active can have unpredictable results. A power-on reset cycle might be required to reset the DAP before switching can be retried.

3.5 JTAG-DP

JTAG-DP contains a debug port state machine (JTAG) that controls the JTAG-DP operation, including controlling the scan chain interface that provides the external physical interface to the JTAG-DP. It is based closely on the JTAG TAP State Machine, see *IEEE Std 1149.1-2001*.

This section contains the following:

- *Scan chain interface*
- *IR scan chain and IR instructions* on page 3-23
- *DR scan chain and DR registers* on page 3-26.

3.5.1 Scan chain interface

The JTAG-DP comprises:

- a DAP State Machine (JTAG)
- an *Instruction Register* (IR) and associated IR scan chain, used to control the behavior of the JTAG and the currently-selected data register
- a number of *Data Registers* (DRs) and associated DR scan chains, that interface to the registers in the JTAG-DP.

DAP State Machine (JTAG)

Figure 3-7 on page 3-21 shows the JTAG state machine.

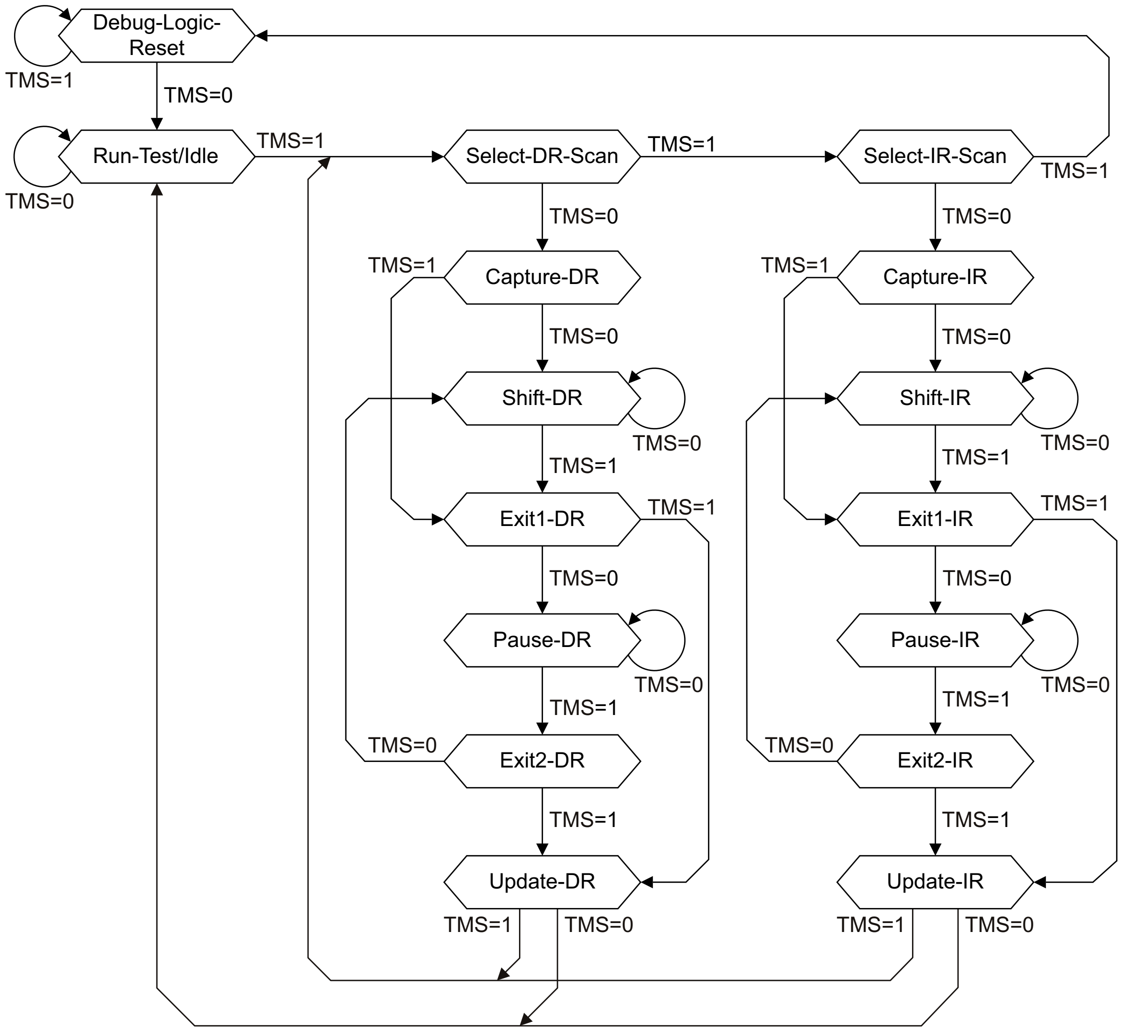


Figure 3-7 DAP State Machine (JTAG)

When using an ARM Debug Interface, for the debug process to work correctly, systems *must not* remove power from the JTAG-DP during a debug session. If power is removed, the DAP controller state is lost. However, the DAP is designed to enable the rest of the DAP and the core to be powered down and debugged, while maintaining power to the JTAG-DP.

Basic operation of the JTAG-DP

The **TDI** signal into the DAP is the start of the scan chain, and the **TDO** signal out of the DAP is the end of the scan chain.

Referring to the DAP State Machine (JTAG) shown in Figure 3-7 on page 3-21:

- When the JTAG goes through the Capture-IR state, a value is transferred onto the *Instruction Register* (IR) scan chain. The IR scan chain is connected between **TDI** and **TDO**.
- While the JTAG is in the Shift-IR state, and for the transition from Capture-IR to Shift-IR, the IR scan chain advances one bit for each tick of **TCK**. This means that on the first tick, the LSB of the IR is output on **TDO**, bit [1] of the IR is transferred to bit [0], bit [2] is transferred to bit [1], and so on. The MSB of the IR is replaced with the value on **TDI**.
- When the JTAG goes through the Update-IR state, the value scanned into the scan chain is transferred into the Instruction Register.
- When the JTAG goes through the Capture-DR state, a value is transferred from one of a number of *Data Registers* (DRs) onto one of a number of Data Register scan chains, connected between **TDI** and **TDO**.

  This data is then shifted while the JTAG is in the Shift-DR state, in the same manner as the IR shift in the Shift-IR state.
- When the JTAG goes through the Update-DR state, the value scanned into the scan chain is transferred into the Data Register
- When the JTAG is in the Run-Test/Idle state, no special actions occur. Debuggers can use this as a true resting state.

  Note

  This is a change from the behavior of previous versions of the ARM Debug Interface based on the IEEE JTAG standard. From ARM Debug Interface v5, debuggers do not have to gate the DAP clock to obtain a true rest state.

The behavior of the IR and DR scan chains is described in more detail in *IR scan chain and IR instructions* on page 3-23 and *DR scan chain and DR registers* on page 3-26.

The **nTRST** signal only resets the JTAG state machine logic. **nTRST** asynchronously takes the JTAG state machine logic to the Debug-Logic-Reset state. As shown in Figure 3-7 on page 3-21, the Debug-Logic-Reset state can also always be entered synchronously from any state by a sequence of five **TCK** cycles with **TMS** high. However, depending on the initial state of the JTAG, this might take the state machine through one of the Update states, with the resulting side effects.

In the DAP, the debug port registers are only reset on a power-on reset.

3.5.2 IR scan chain and IR instructions

This section describes the JTAG-DP *Instruction Register* (IR), accessed through the IR scan chain.

JTAG Instruction Register (IR)

Purpose Holds the current DAP Controller instruction.

Length 4 bits.

Operating mode

When in Shift-IR state, the shift section of the IR is selected as the serial path between **TDI** and **TDO**. At the Capture-IR state, the binary value b0001 is loaded into this shift section. This is shifted out, least significant bit first, during Shift-IR. As this happens, a new instruction is shifted in, least significant bit first. At the Update-IR state, the value in the shift section is loaded into the IR so it becomes the current instruction.

On debug logic reset, IDCODE becomes the current instruction, see *JTAG Device ID Code Register (IDCODE)* on page 3-26.

Order Figure 3-8 shows the bit order of the Instruction Register.

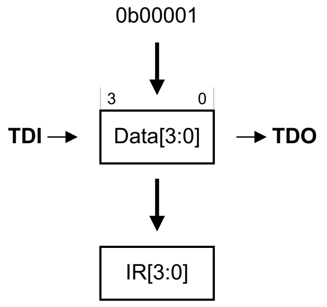


Figure 3-8 JTAG Instruction Register bit order

This register is mandatory in the IEEE 1149.1 standard.

IR instructions

The description of the JTAG Instruction Register shows how a 4-bit instruction is transferred into the IF. This instruction determines the physical Data Register that the JTAG Data Register maps onto, as described in *DR scan chain and DR registers* on page 3-26. The standard IR instructions are listed in Table 3-2, and recommended implementation-defined extensions to this instruction set are described in *Implementation-defined extensions to the IR instruction set*.

Unused IR instruction values select the Bypass register, described in *JTAG Bypass Register (BYPASS)* on page 3-26.

Table 3-2 Standard IR instructions

IR instruction value	JTAG-DP register	DR scan width	See section
b0xxx	-	-	*Implementation-defined extensions to the IR instruction set*
b1000	ABORT	35	*JTAG-DP Abort Register (ABORT)* on page 3-34
b1001	-	-	-
b1010	DPACC	35	*JTAG DP/AP Access Registers (DPACC/APACC)* on page 3-27
b1011	APACC	35	
b110x	-	-	-
b1110	IDCODE	32	*JTAG Device ID Code Register (IDCODE)* on page 3-26
b1111	BYPASS	1	*JTAG Bypass Register (BYPASS)* on page 3-26

Implementation-defined extensions to the IR instruction set

The eight IR instructions b0000 to b0111 are reserved for implementation-defined extensions to the JTAG-DP. These registers might be used for accessing a boundary scan register, for IEEE 1149.1 compliance. Table 3-3 on page 3-25 shows the recommended instructions.

Note

- ARM Limited recommends that you use separate JTAG TAPs boundary scan and debug.
- If the IR register is set to an IR instruction value that is not implemented, or reserved, then the Bypass Register is selected.

Table 3-3 Implementation-defined IR instructions for IEEE 1149.1-compliance

IR instruction value	Instruction	Required by IEEE 1149.1?
b0000	EXTEST	Yes
b0001	SAMPLE	Yes
b0010	PRELOAD	Yes
b0011	Reserved	-
b0100	INTEST[a]	No
b0101	CLAMP[a]	No
b0110	HIGHZ[a]	No
b0111	CLAMPZ[a]	No. See *CLAMPZ instruction*.

a. Reserved, if the instruction is not implemented.

If you require a boundary scan implementation you must implement the instructions that are shown as required by IEEE 1149.1 in a wrapper to JTAG-DP. The other IR instruction values listed in Table 3-3 are Reserved encodings that should be used if that function is implemented in the boundary scan. If implemented, these instructions must behave as required by the IEEE 1149.1 specification. If not implemented, they select the Bypass register.

The IEEE 1149.1 specification also requires the IDCODE and BYPASS instructions. However these are included in Table 3-2 on page 3-24.

CLAMPZ instruction

CLAMPZ is not an IEEE 1149.1 instruction.

If implemented, when the CLAMPZ instruction is selected all the 3-state outputs are placed in their inactive state, but the data supplied to the outputs is derived from the scan cells.

CLAMPZ can be implemented to ensure that, during production test, each output can be disabled when its value is 0 or 1. This encoding should be used if this function is required.

JTAG Bypass Register (BYPASS)

Purpose Bypasses the device, by providing a direct path between **TDI** and **TDO**.

Length 1 bit.

Operating mode

When the BYPASS instruction is the current instruction in the IR:

- in the Shift-DR state, data is transferred from **TDI** to **TDO** with a delay of one TCK cycle
- in the Capture-DR state, a logic 0 is loaded into this register
- nothing happens at the Update-DR state.

Order Figure 3-9 shows the operation of the Bypass Register.

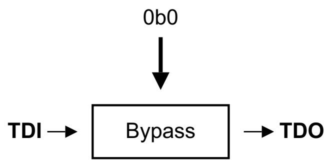


Figure 3-9 JTAG Bypass Register operation

This register is mandatory in the IEEE 1149.1 standard.

3.5.3 DR scan chain and DR registers

There are five physical DR registers:

- the BYPASS and IDCODE Registers, as defined by the IEEE 1149.1 standard
- the DPACC and APACC Access Registers
- an ABORT Register, used to abort a transaction.

There is a scan chain associated with each of these registers. As described in *IR scan chain and IR instructions* on page 3-23, the value in the IR register determines which of these scan chains is connected to the **TDI** and **TDO** signals.

JTAG Device ID Code Register (IDCODE)

Purpose Device identification. The Device ID Code value enables a debugger to identify the debug port to which it is connected. Different debug ports have different Device ID Codes, so that a debugger can make this distinction.

This is the JTAG-DP implementation of the Identification Code Register, see *Identification Code Register, IDCODE* on page 3-76.

Length 32 bits.

Operating mode

When the IDCODE instruction is the current instruction in the IR, the shift section of the Device ID Code Register is selected as the serial path between **TDI** and **TDO**:

- in the Capture-DR state, the 32-bit device ID code is loaded into this shift section
- in the Shift-DR state, this data is shifted out, least significant bit first
- the shifted-in data is ignored at the Update-DR state.

Order Figure 3-10 shows the bit order of the Device ID Code Register.

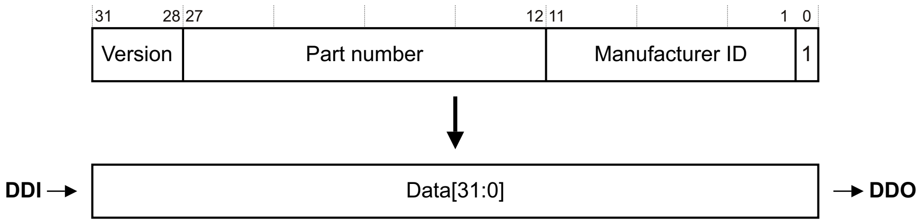


Figure 3-10 JTAG Device ID Code Register bit order

JTAG DP/AP Access Registers (DPACC/APACC)

The DPACC and APACC scan chains have the same format.

Purpose Initiate a debug port or access port access, to access a debug port or access port register. The DPACC and APACC are used for read and write accesses to registers.

The DPACC is used to access the CTRL/STAT, SELECT and RDBUFF registers, see *JTAG-DP register map* on page 3-72.

The APACC is used to access all of the access port registers, see *AHB-AP register summary* on page 3-88 for details of accessing AHB-AP registers, and *JTAG-DP registers* on page 3-72 for details of accessing JTAG-AP registers.

Length 35 bits.

Operating mode

When the DPACC or APACC instruction is the current instruction in the IR, the shift section of the DP Access Register or AP Access Register is selected as the serial path between **TDI** and **TDO**:

- In the Capture-DR state, the result of the previous transaction, if any, is returned, together with a 3-bit ACK response. Only two ACK responses are implemented, and these are summarized in Table 3-4.

  Table 3-4 DPACC and APACC ACK responses

Response	ACK[2:0] encoding	See:
OK/FAULT	b010	*OK/FAULT response to a DPACC or APACC access* on page 3-29
WAIT	b001	*WAIT response to a DPACC or APACC access* on page 3-31

  All other ACC encodings are Reserved.

- In the Shift-DR state, this data is shifted out, least significant bit first. As shown in Figure 3-11 on page 3-29, the first three bits of data shifted out are ACK[2:0], and therefore you can check the ACK response without shifting out all of the returned data, see *WAIT response to a DPACC or APACC access* on page 3-31.

  As the returned data is shifted out to **TDO**, new data is shifted in from **TDI**. This is described in *OK/FAULT response to a DPACC or APACC access* on page 3-29.

- Operation in the Update-DR depends on whether the ACK[2:0] response was OK/FAULT or WAIT. The two cases are described in:
  - *Update-DR operation following an OK/FAULT response* on page 3-29
  - *Update-DR operation following a WAIT response* on page 3-31.

Order Figure 3-11 shows the bit order of the DP and AP Access Registers.

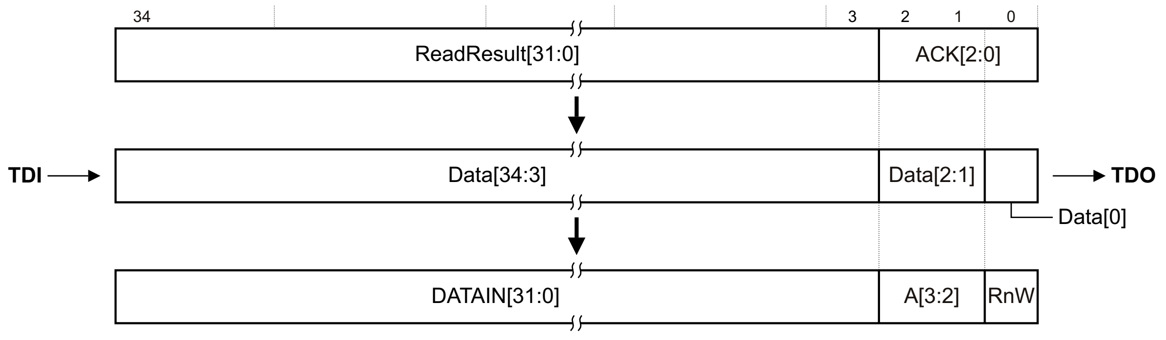


Figure 3-11 Bit order of JTAG DP and AP Access Registers

OK/FAULT response to a DPACC or APACC access

If the response indicated by ACK[2:0] is OK/FAULT, the previous transaction has completed. The response code does not show whether the transaction completed successfully or was faulted. You must read the CTRL/STAT register to find whether the transaction was successful, see *Control/Status Register, CTRL/STAT* on page 3-77:

- If the previous transaction was a read that completed successfully, then the captured ReadResult[31:0] is the requested register value. This result is shifted out as Data[34:3].
- If the previous transaction was a write, or a read that did not complete successfully, then the captured ReadResult[31:0] is Unpredictable, and if Data[34:3] is shifted out it must be discarded.

Update-DR operation following an OK/FAULT response

The values shifted into the scan chain form a request to read or write a register:

- if the current IR instruction is DPACC then **TDI** and **TDO** connect to the DPACC scan chain, and the request is to read or write a DP register
- if the current IR instruction is APACC then **TDI** and **TDO** connect to the APACC scan chain, and the request is to read or write an AP register.

In either case:

- If RnW is shifted in as 0, the request is to write the value in DATAIN[31:0] to the addressed register.
- If RnW is shifted in as 1, the request is to read the value of the addressed register. The value in DATAIN[31:0] is ignored. You must read the scan chain again to obtain the value read from the register.

The required register is addressed:

- In the case of a DPACC access, to read a debug port register, by the value shifted into A[3:2]. See *JTAG-DP register map* on page 3-72 for the addressing details.
- In the case of a APACC access, to read an access port register, by the combination of:
  - the value shifted into A[3:2]
  - the current value of the SELECT register in the DP, see *AP Select Register, SELECT* on page 3-81.

  For details of the register addressing, see:
  - *APB-AP registers* on page 3-100, if you want to access an AHB-AP register

Register accesses can be pipelined, because a single DPACC or APACC scan can return the result of the previous read operation at the same time as requesting another register access. At the end of a sequence of pipelined register reads, you can read the DP RDBUFF Register to return the result of the final register read. Reading the DP RDBUFF Register is benign, that is, it has no effect on the operation of the JTAG, see *Read Buffer, RDBUFF* on page 3-83. The section *Target response summary* on page 3-32 gives more information about how one DPACC or APACC scan returns the result from the previous scan.

If the current IR instruction is APACC, causing an APACC access:

- If any sticky flag is set in the DP CTRL/STAT Register, the transaction is discarded. The next scan returns an OK/FAULT response immediately. For more information see *Sticky flags and debug port error responses* on page 3-58, and *Control/Status Register, CTRL/STAT* on page 3-77.
- If pushed compare or pushed verify operations are enabled then the scanned-in value of RnW *must* be 0, otherwise behavior is Unpredictable. On Update-DR, a read request is issued, and the returned value compared against DATAIN[31:0]. The STICKYCMP flag in the DP CTRL/STAT register is updated based on this comparison. For more information see *Pushed compare and pushed verify operations* on page 3-61. Pushed operations are enabled using the TRNMODE field of the DP CTRL/STAT register, see *Control/Status Register, CTRL/STAT* on page 3-77 for more information.
- The AP access does not complete until the access port signals it as completed. For example, if you access a Memory Access Port (AHB-AP), the access might cause an access to a memory system connected to the AHB-AP. In this case, the access does not complete until the memory system signals to the AHB-AP that the memory access has completed.

WAIT response to a DPACC or APACC access

A WAIT response indicates that the previous transaction has not completed. The host should retry the DPACC or APACC access.

Note

The previous transaction might be either a debug port or an access port access. Accesses to the debug port are stalled, by returning WAIT, until any previous access port transaction has completed.

Normally, if software detects a WAIT response, it retries the same transfer. This enables the protocol to process data as quickly as possible. However, if the software has retried a transfer a number of times, allowing enough time for a slow interconnect and memory system to respond, it might write to the ABORT register, to cancel the operation. This signals to the active access port that it should terminate the transfer it is currently attempting, and permits access to other parts of the debug system. An access port might not be able to terminate a transfer on its ASIC interface. However, on receiving an ABORT, the access port should free its JTAG interface.

Update-DR operation following a WAIT response

No request is generated at the Update-DR state, and the shifted-in data is discarded. The captured value of ReadResult[31:0] is Unpredictable.

Note

You can detect a WAIT response without shifting through the entire DP or AP Access Register, see the response details in Table 3-4 on page 3-28.

Sticky overrun behavior on DPACC and APACC accesses

At the Capture-DR state, if the previous transaction has not completed a WAIT response is generated. When this happens, if the Overrun Detect flag is set, the Sticky Overrun flag, STICKYORUN, is set. See *Control/Status Register, CTRL/STAT* on page 3-77 for more information about the Overrun Detect and Sticky Overrun flags.

While the previous transaction remains not completed, subsequent scans also receive a WAIT response.

Once the previous transaction has completed, further APACC transactions are abandoned and scans respond immediately with an OK/FAULT response. However, debug port registers can be accessed. In particular the CTRL/STAT register can be accessed, to confirm that the Sticky Overrun flag is set, and to clear the flag after gathering any required information about the overrun condition. See *Overrun detection* on page 3-59 for more information.

Minimum response times

As explained in *OK/FAULT response to a DPACC or APACC access* on page 3-29, a debug port or access port register access is initiated at the Update-DR state of one DPACC or APACC access, and the result of the access is returned at the Capture-DR state of the following DPACC or APACC access. However, the second access generates a WAIT response if the requested register access has not completed.

The JTAG clock, **TCK**, is asynchronous to the internal clock of the system being debugged, and the time required for an access to complete includes clock cycles in both domains. However, the timing between the Update-DR state and the Capture-DR state only includes **TCK** cycles. Referring to Figure 3-7 on page 3-21, there are two paths from the Update-DR state, where the register access is initiated, to the Capture-DR state, where the response is captured:

- a direct path through Select-DR-Scan
- a path through Run-Test/Idle and Select-DR-Scan.

If the second path is followed, the state machine can spend any number of **TCK** cycles spinning in the Run-Test/Idle state. This means it is possible to vary the number of **TCK** cycles between the Update-DR and Capture-DR states.

A JTAG implementation might impose an implementation-defined lower limit on the number of **TCK** cycles between the Update-DR and Capture-DR states, and always generate an immediate WAIT response if Capture-DR is entered before this limit has expired. Although any debugger must be able to recover successfully from any WAIT response, ARM Limited recommends that debuggers should be able to adapt to any Implementation-defined limit.

In addition, when accessing access port registers, or accessing a connected device through an access port, there might be other variable response delays in the system. A debugger that can adapt to these delays, avoiding wasted WAIT scans, operates more efficiently and provides higher maximum data throughput.

Target response summary

As described in *OK/FAULT response to a DPACC or APACC access* on page 3-29 and *Minimum response times*, a debug port or access port register access is initiated at the Update-DR state of one DPACC or APACC access, and the result of the access is returned at the Capture-DR state of the following DPACC or APACC access. Table 3-5 on page 3-33 summarizes the target responses, at the Capture-DR state, for every possible DPACC and APACC access in the previous scan.

Note

The target responses shown in Table 3-5 on page 3-33 are independent of the operation being performed in the current DPACC or APACC scan. In the table, *Read result* is the data shifted out as Data[34:3], and ACK is decoded from the data shifted out as Data[2:0].

Table 3-5 JTAG target response summary

Previous scan, at Update-DR state[a]				Current scan, at Capture-DR state			Notes
R/W	IR	ADDR [b]	Sticky[c]	AP state[d]	Read result	ACK	
X	X	bXX	X	Busy	UNP[e]	WAIT	Can cause Sticky Overrun flag to be set.[f]
R	DPACC	b01	X	Not Busy	CTRL/STAT	OK/FAULT	Returns CTRL/STAT value.
		b10			SELECT		Returns SELECT value.
		b00 or b11			`0x00000000`		No readable DP registers at addresses b00 and b11.
W	DPACC	b01	X	Not Busy	UNP[e]	OK/FAULT	Write to CTRL/STAT.
		b10					Write to SELECT.
		b00 or b11					Write ignored.
R	APACC	bXX	No	Ready	See Notes	OK/FAULT	See footnote[g].
				Error	UNP[e]		Sticky Error flag is set.
W	APACC	bXX	No	Ready	UNP[e]	OK/FAULT	See footnote[h].
				Error	UNP[e]		Sticky Error flag is set.
X	APACC	bXX	Yes	X	UNP[e]	OK/FAULT	Previous transaction was discarded.

a. The Previous scan is the most recent scan for which the ACK response at the Capture-DR state was OK/FAULT. Updates made following a WAIT response are discarded.
b. ADDR[3:2] in the DPACC or APACC access.
c. The Sticky column indicates whether any Sticky flag is set in the DP CTRL/STAT register, see *Control/Status Register, CTRL/STAT* on page 3-77.
d. The state of the AP when the current scan reaches the Capture-DR state, or the response from the AP at that time.
e. UNP = Unpredictable.
f. If the Overrun Detect flag is set then this access/response sequence causes the Sticky Overrun flag to be set. See *Control/Status Register, CTRL/STAT* on page 3-77.
g. If Pushed Verify or Pushed Compare is enabled, the behavior is Unpredictable. Otherwise, returns the value of the AP Register addressed on the previous scan.
h. If Pushed Verify or Pushed Compare is enabled, the previous transaction performed the required pushed operation, which might have set the Sticky Compare flag, see *Pushed compare and pushed verify operations* on page 3-61. Otherwise, the data captured at the previous scan has been written to the AP register requested.

Host response summary

The ACK column, for the *Current scan, at Capture-DR* state section of Table 3-5 on page 3-33, shows the responses the host might receive after initiating a DPACC or APACC access.

Table 3-6 Summary of JTAG host responses

JTAG access type	ACK from target	Suggested host action in response to ACK
Read	OK/FAULT	Capture read data.
Write	OK/FAULT	No more action required.
Read or Write	WAIT	Repeat the same access until either an OK/FAULT ACK is received or the wait timeout is reached. If necessary, use the DAP ABORT register to enable access to the AP.
Read or Write	Invalid ACK	Assume a target or line error has occurred and treat as a fatal error.

JTAG-DP Abort Register (ABORT)

Purpose Access the DP Abort Register, to force a DAP abort.

This is the JTAG-DP implementation of the Abort Register, see *Abort Register, ABORT* on page 3-74.

Length 35 bits.

Operating mode

When the ABORT instruction is the current instruction in the IR, the serial path between **TDI** and **TDO** is connected to a 35-bit scan chain that is used to access the Abort Register.

The debugger must scan the value `0x0000008` into this scan chain. This value:

- writes the RnW bit as 0
- writes the A[3:2] field as b00
- writes 1 into bit 0, the DAPABORT bit, of the Abort Register.

Caution

The effect of writing any other value into this scan chain is Unpredictable.

Order Figure 3-12 shows the bit order of the ABORT scan chain.

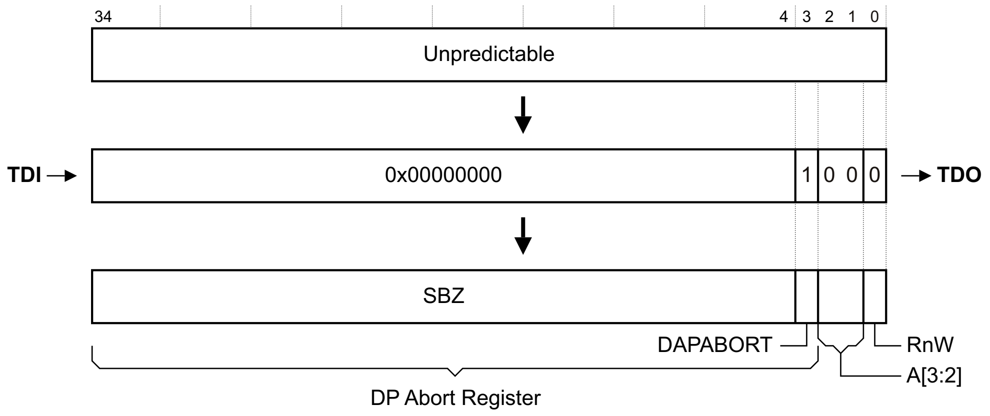


Figure 3-12 JTAG-DP ABORT scan chain bit order

3.6 SW-DP

This section describes the *Serial Wire Debug Port* (SW-DP) interface. In particular, it describes the *Serial Wire Debug* (SWD) protocol, and how this protocol provides access to the debug port registers. These registers are described in detail in *Debug port programmer's model* on page 3-72.

The SW-DP operates with a synchronous serial interface. This uses a single bidirectional data signal, and a clock signal.

Each sequence of operations on the wire consists of two or three phases:

Packet request

The external *host* debugger issues a request to the debug port. The debug port is the *target* of the request.

Acknowledge response

The target sends an acknowledge response to the host.

Data transfer phase

This phase is only present when either:

- a data read or data write request is followed by a valid (OK) acknowledge response
- the ORUNDETECT flag is set to 1 in the CTRL/STAT Register, see *Control/Status Register, CTRL/STAT* on page 3-77.

The data transfer is one of:

- target to host, following a read request (RDATA)
- host to target, following a write request (WDATA).

Note

If the Overrun Detect bit in the CTRL/STAT Register is set to 1, then a data transfer phase is required on all responses, including WAIT and FAULT. For more information, see *Sticky overrun behavior* on page 3-48.

For details of the CTRL/STAT Register see *Control/Status Register, CTRL/STAT* on page 3-77.

3.6.1 Clocking

The SW-DP clock, **SWCLKTCK**, can be asynchronous to the **DAPCLK**. **SWCLKTCK** can be stopped when the debug port is idle.

The host must continue to clock the interface for a number of cycles after the data phase of any data transfer. This ensures that the transfer can be clocked through the SW-DP. This means that after the data phase of any transfer the host must do one of the following:

- immediately start a new SW-DP operation
- continue to clock the SW-DP serial interface until the host starts a new SW-DP operation
- after clocking out the data parity bit, continue to clock the SW-DP serial interface until it has clocked out at least 8 more clock rising edges, before stopping the clock.

3.6.2 Overview of debug interface

This section gives an overview of the physical interface used by the SW-DP.

Line interface

The SW-DP uses a serial wire for both host and target sourced signals. The host emulator drives the protocol timing - only the host emulator generates packet headers.

The SW-DP operates in synchronous mode, and requires a clock pin and a data pin.

Synchronous mode uses a clock reference signal, which can be sourced from an on-chip source and exported, or provided by the host device. This clock is then used by the host as a reference for generation and sampling of data so that the target is not required to perform any oversampling.

Both the target and host are capable of driving the bus HIGH and LOW, or tristating it. The ports must be able to tolerate short periods of contention to allow for loss of synchronization.

Line pullup

Both the host and target are able to drive the line HIGH or LOW, so it is important to ensure that contention does not occur by providing undriven time slots as part of the handover. So that the line can be assumed to be in a known state when neither is driving the line, a 100kOhm pullup is required at the target, but this can only be relied on to maintain the state of the wire. If the wire is driven LOW and released, the pullup resistor eventually brings the line to the HIGH state, but this takes many bit periods.

The pullup is intended to prevent false detection of signals when no host device is connected. It must be of a high value to reduce IDLE state current consumption from the target when the host actively pulls down the line.

Note

Whenever the line is driven LOW, this results in a small current drain from the target. If the interface is left connected for extended periods when the target has to use a low power mode, the line must be held HIGH, or reset, by the host until the interface must be activated.

Line turn-round

To avoid contention, a turnaround period is required when the device driving the wire changes.

Idle and reset

Between transfers, the host must either drive the line LOW to the IDLE state, or continue immediately with the start bit of a new transfer. The host is also free to leave the line HIGH, either driven or tristated, after a packet. This reduces the static current drain, but if this approach is used with a free running clock, a minimum of 50 clock cycles must be used, followed by a READ-ID as a new re-connection sequence.

There is no explicit reset signal for the protocol. A reset is detected by either host or target when the expected protocol is not observed. It is important that both ends of the link become reset before the protocol can be restarted with a reconnection sequence. Re-synchronization following the detection of protocol errors or after reset is achieved by providing 50 clock cycles with the line HIGH, or tristate, followed by a read ID request.

If the SW-DP detects that it has lost synchronization, for example no stop bit is seen when expected, it leaves the line undriven and waits for the host to either re-try with a new header after a minimum of one cycle with the line LOW, or signals a reset by not driving the line itself. If the SW-DP detects two bad data sequences in a row, it locks out until a reset sequence of 50 clock cycles with DBGDI high is seen.

If the host does not see an expected response from SW-DP, it must allow time for SW-DP to return a data payload. The host can then retry with a read to the SW-DP ID code register. If this is unsuccessful, the host must attempt a reset.

3.6.3 Overview of protocol operation

This section gives an overview of the bi-directional operation of the protocol. It shows each of the possible sequences of operations on the SW-DP interface data connection.

The sequences of operations illustrated here are:

- *Successful write operation (OK response)* on page 3-42
- *Successful read operation (OK response)* on page 3-42
- *WAIT response to Read or Write operation request* on page 3-43
- *FAULT response to Read or Write operation request* on page 3-43
- *Protocol error sequence* on page 3-44.

The terms used in the illustrations are described in *Key to illustrations of operations*.

——— **Note** ———

The diagrams in this section are included to show the operation of the SWD protocol. They are not timing diagrams for the protocol. Contact ARM Limited if you require more information about timings of the serial connection to the SW-DP.

———————————

Key to illustrations of operations

The illustrations of the different possible operations use the following terms:

Start	A single start bit, with value 1.
APnDP	A single bit, indicating whether the DP or the AP Access Register is to be accessed. This bit is 0 for a DPACC access, or 1 for an APACC access.
RnW	A single bit, indicating whether the access is a read or a write. This bit is 0 for an write access, or 1 for a read access.
ADDR[2:3]	Two bits, giving the ADDR[3:2] address field for the DP or AP Register Address:

- For an APACC access, the register being addressed depends on the ADDR[3:2] value and the value held in the SELECT register. For details of the addressing see *AHB-AP programmer's model* on page 3-88, if you want to access a AHB-AP register.

  For details of the SELECT register see *AP Select Register, SELECT* on page 3-81.

- For a DPACC access, the A[3:2] value determines the address of the register in the SW-DP register map, see Table 3-13 on page 3-72.

Note

The A[3:2] value is transmitted LSB-first on the wire. This is why it appears as A[2:3] on the diagrams.

Parity A single parity bit for the preceding packet. See *Parity in the SWD protocol* on page 3-41.

Stop A single stop bit. In the synchronous SWD protocol this is always 0.

Park A single bit. The host must drive the line high before tristating the line. The target reads this bit as 1.

Trn Turnaround. This is a period during which the line is not driven and the state of the line is Undefined. The length of the turnaround period is controlled by the TURNROUND field in the Wire Control Register, see *Wire Control Register, WCR (SW-DP only)* on page 3-84. The default setting is a turnaround period of one clock cycle.

Note

All the examples given in this chapter show the default turnaround period of one cycle.

ACK A three-bit target-to-host response.

Note

The ACK value is transmitted LSB-first on the wire. This is why it appears as ACK[0:2] on the diagrams.

WDATA[0:31]

32 bits of write data, from host to target.

Note

The WDATA[0:31] value is transmitted LSB-first on the wire. This is why it appears as WDATA[0:31] on the diagrams.

RDATA[0:31]

32 bits of read data, from target to host.

Note

The RDATA[0:31] value is transmitted LSB-first on the wire. This is why it appears as RDATA[0:31] on the diagrams.

Parity in the SWD protocol

In the SWD protocol, a simple parity check is applied to all packet request and data transfer phases. Even parity is used:

Packet requests

The parity check is made over the APnDP, RnW and A[2:3] bits. If, of these four bits:

- the number of bits set to 1 is odd, then the parity bit is set to 1
- the number of bits set to 1 is even, then the parity bit is set to 0.

Data transfers (WDATA and RDATA)

The parity check is made over the 32 data bits, WDATA[0:31] or RDATA[0:31]. If, of these 32 bits:

- the number of bits set to 1 is odd, then the parity bit is set to 1
- the number of bits set to 1 is even, then the parity bit is set to 0.

The packet request parity bit is shown in each of the diagrams in this section, from Figure 3-13 on page 3-42 to Figure 3-19 on page 3-49. It appears on the wire immediately after the A[2:3] bits.

The WDATA parity bit is shown in Figure 3-13 on page 3-42 and in Figure 3-19 on page 3-49. It appears on the wire immediately after the WDATA[31] bit.

The RDATA parity bit is shown in Figure 3-14 on page 3-43 and in Figure 3-18 on page 3-48. It appears on the wire immediately after the RDATA[31] bit.

——— **Note** ———

The ACK[0:2] bits are never included in the parity calculation. Debuggers must remember this when parity checking the data from a read operation, when the debugger receives a continuous stream of 36 bits, as shown in Figure 3-14 on page 3-43:

- bits 0 to 2 are ACK[0:2]
- bits 3 to 34 are RDATA[0:31]
- bit 35 is the parity bit.

The parity check must be applied to bits 3 to 34 of this block of data, and the result compared with bit 35, the parity bit.

———————

Successful write operation (OK response)

A successful write operation consists of three phases:

- an eight-bit write packet request, from the host to the target
- a three-bit OK acknowledge response, from the target to the host
- a 33-bit data write phase, from the host to the target.

By default, there are single-cycle turnaround periods between each of these phases. See the description of **Trn** in *Key to illustrations of operations* on page 3-39 for more information.

Figure 3-13 shows a successful write operation.

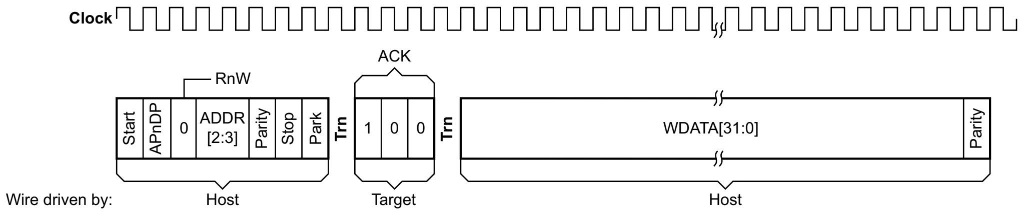


Figure 3-13 SWD successful write operation

Note

The OK response shown in Figure 3-13 only indicates that the debug port is ready to accept the write data. The debug port writes this data after the write phase has completed, and therefore the response to the debug port write itself is given on the next operation.

There is no turnaround phase after the data phase. The host is driving the line, and can start the next operation immediately.

Successful read operation (OK response)

A successful read operation consists of three phases:

- an eight-bit read packet request, from the host to the target
- a three-bit OK acknowledge response, from the target to the host
- a 33-bit data read phase, where data is transferred from the target to the host.

By default, there are single-cycle turnaround periods between the first and second of these phases, and after the third phase. See the description of **Trn** in *Key to illustrations of operations* on page 3-39 for more information. However, there is no turnaround period between the second and third phases.

Figure 3-14 shows a successful read operation.

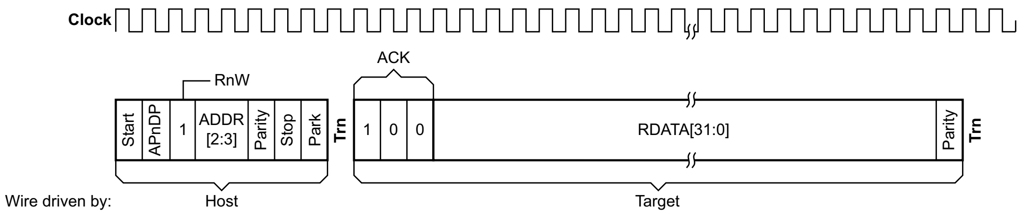


Figure 3-14 SWD successful read operation

WAIT response to Read or Write operation request

A WAIT response to a read or write packet request consists of two phases:

- an eight-bit read or write packet request, from the host to the target
- a three-bit WAIT acknowledge response, from the target to the host.

By default, there are single-cycle turnaround periods between these two phases, and after the second phase. See the description of **Trn** in *Key to illustrations of operations* on page 3-39 for more information.

Figure 3-15 shows a WAIT response to a read or write packet request.

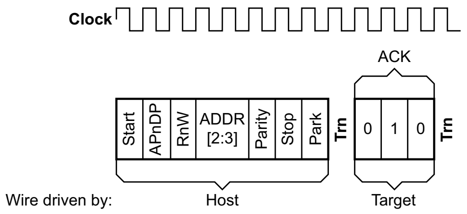


Figure 3-15 SWD WAIT response to a packet request

——— **Note** ———

If Overrun Detection is enabled then a data phase is required on a WAIT response. For more information see *Sticky overrun behavior* on page 3-48.

FAULT response to Read or Write operation request

A FAULT response to a read or write packet request consists of two phases:

- an eight-bit read or write packet request, from the host to the target
- a three-bit FAULT acknowledge response, from the target to the host.

By default, there are single-cycle turnaround periods between these two phases, and after the second phase. See the description of **Trn** in *Key to illustrations of operations* on page 3-39 for more information.

Figure 3-16 shows a FAULT response to a read or write packet request.

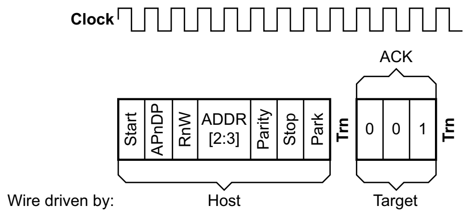


Figure 3-16 SWD FAULT response to a packet request

——— **Note** ———

If Overrun Detection is enabled then a data phase is required on a FAULT response. For more information see *Sticky overrun behavior* on page 3-48.

———————————

Protocol error sequence

A protocol error occurs when a host issues a packet request but the target fails to return any acknowledge response. This is shown in Figure 3-17.

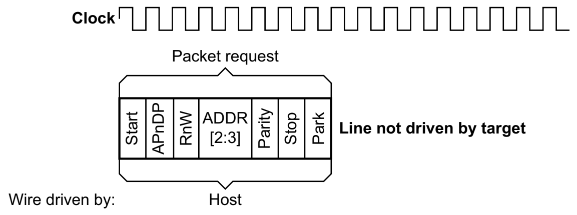


Figure 3-17 SWD protocol error after a packet request

3.6.4 Protocol description

This section provides additional information on the DAP Serial Wire Debug operations that were introduced in *Overview of protocol operation* on page 3-39.

Connection and line reset sequence

The serial interface to the SW-DP must use a connection sequence, to ensure that hot-plugging the serial connection does not result in unintentional transfers. The connection sequence ensures that the SW-DP is synchronized correctly to the header that is used to signal a connection. It consists of a sequence of 50 clock cycles with data = 1, that is, with the serial data signal asserted HIGH by the debugger.

This connection sequence is also used as a line reset sequence, see Protocol Error responses on page 5-13. The protocol requires that any run of 50 consecutive 1s on the data input is detected as a line reset, regardless of the state of the protocol.

After the host has transmitted a line request sequence to the SW-DP, it must read the IDCODE register. The SW-DP returns an OK response to this read. For more information see:

- *Identification Code Register, IDCODE* on page 3-76
- *Successful read operation (OK response)* on page 3-42.

The requirement that the host reads the IDCODE register to exit the training state gives confirmation that correct packet frame alignment has been achieved.

OK response

When it receives a packet request from the debug host, the SW-DP must respond immediately. It issues an OK response, indicated by an acknowledge phase of b001, if it is ready for the data phase of the transfer, if one is required.

——— **Note** ———

- As shown in *Overview of protocol operation* on page 3-39, there is always a turnaround between the end of the packet request from the host and the start of the acknowledgement from the SW-DP target. The default turnaround is exactly one serial clock cycle, but see the description of **Trn** in *Key to illustrations of operations* on page 3-39 for more information.

  There is a turnaround whenever there is a change in the direction of data transfer over the serial SWD connection. If an operation that is described as immediate involves a change in the data transfer direction then the operation must start immediately after the turnaround.

- All SWD transfers are made LSB-first. Therefore, the OK response of b001 appears on the wire as 1, followed by 0, followed by 0, as shown in Figure 3-13 on page 3-42 and Figure 3-14 on page 3-43.

———————

If the host requested a write access it must start the write transfer immediately after receiving the acknowledgement from the target. This behavior is the same whether the write is to the debug port or to an access port. However, the SW-DP can buffer AP writes, as described in *SW-DP write buffering* on page 3-49.

If the host requested a read access to the debug port then the SW-DP sends the read data immediately after the acknowledgement. Because there is no change in the data transfer direction between the acknowledgement and the read data there is not any turnaround between these phases. This is shown in Figure 3-14 on page 3-43.

Read accesses to the access port are *posted*. This means that the result of the access is returned on the next transfer. If the next access you have to make is not another access port read then you must insert a read of the DP RDBUFF Register to obtain the posted result, see *Read Buffer, RDBUFF* on page 3-83.

When you must make a series of access port reads, you only have to insert one read of the RDBUFF Register:

- On the first access port read access, the read data returned is Undefined. You must discard this result.
- If you immediately make another access port read access this returns the result of the previous access port read.
- You can repeat this for any number of access port reads.
- Issuing the last access port read packet request returns the last-but-one access port read result.
- You must then read the DP RDBUFF Register to obtain the last access port read result.

Operation and use of the READOK flag

The SW-DP CTRL/STAT register includes a READOK flag, bit [6]. This register is described in *Control/Status Register, CTRL/STAT* on page 3-77.

The READOK flag is updated on every access port read access, and on every RDBUFF read request. When the SW-DP initiates the access port access it clears the READOK flag to 0, and when the SW-DP target gives an OK response to the read request it sets the READOK flag to 1.

This means that if a host receives a corrupted ACK response to an access port or RDBUFF read request it can check whether the read actually completed correctly. The host can read the DP CTRL/STAT Register to find the value of the READOK flag:

- If the flag is set to 1 then the read was performed correctly. The host can use a RESEND request to obtain the read result, see *Read Resend Register, RESEND (SW-DP only)* on page 3-86.
- If the flag is set to 0 then the read was not successful. The host must retry the original access port or RDBUFF read request.

WAIT response

A WAIT response is issued by the SW-DP if it is not able to process immediately the request from the debugger. However, a WAIT response must not be issued to the following requests. SW-DP must always be able to process these three requests immediately:

- a read of the IDCODE register, see *Identification Code Register, IDCODE* on page 3-76
- a read of the CTRL/STAT register, see *Control/Status Register, CTRL/STAT* on page 3-77
- a write to the ABORT register, see *Abort Register, ABORT* on page 3-74.

With any request other than those listed, the SW-DP issues a WAIT response, with no data phase, if it cannot process the request. This happens:

- if a previous access port or debug port access is outstanding
- if the new request is an access port read request and the result of the previous AP read is not yet available.

Note

When overrun detection is enabled a WAIT response must include a data phase. See *Sticky overrun behavior* on page 3-48 for more information.

Normally, when a debugger receives a WAIT response it retries the same operation. This enables it to process data as quickly as possible. However, if several retries have been attempted, and time allowed for a slow interconnection and memory system to respond, if appropriate, the debugger might write to the ABORT register. This signals to the active access port that it must terminate the transfer that it is currently attempting. An access port implementation might be unable to terminate a transfer on its ASIC interface. However, on receiving an ABORT request the access port must free up the SWD interface.

Writing to the ABORT register after receiving a WAIT response enables the debugger to access other parts of the debug system.

FAULT response

SW-DP does not issue a FAULT response to an access to the IDCODE, CTRL/STAT or ABORT registers. For any other access, the SW-DP issues a FAULT response if any sticky flag is set in the CTRL/STAT Register, see *Control/Status Register, CTRL/STAT* on page 3-77. See *Sticky overrun behavior* for more information about the sticky overrun flag.

Use of the FAULT response enables the protocol to remain synchronized. A debugger might stream a block of data and then check the CTRL/STAT register at the end of the block.

The sticky error flags are cleared by writing bits in the ABORT register, see *Abort Register, ABORT* on page 3-74.

Sticky overrun behavior

If SW-DP receives a transaction request when the previous transaction has not completed it generates a WAIT response. If overrun detection is enabled in the CTRL/STAT Register, the STICKYORUN flag is set to 1 in that register. For more information see *Control/Status Register, CTRL/STAT* on page 3-77. Subsequent transactions generate FAULT responses, because a sticky flag is set.

When overrun detection is enabled, WAIT and FAULT responses require a data phase:

- If the transaction is a read the data in the data phase is Unpredictable. The target does not drive the line, and the host must not check the parity bit.
- If the transaction is a write the data phase is ignored.

Figure 3-18 shows the WAIT or FAULT response to a read operation when overrun detection is enabled. Figure 3-19 on page 3-49 shows the response to a write operation when overrun detection is enabled.

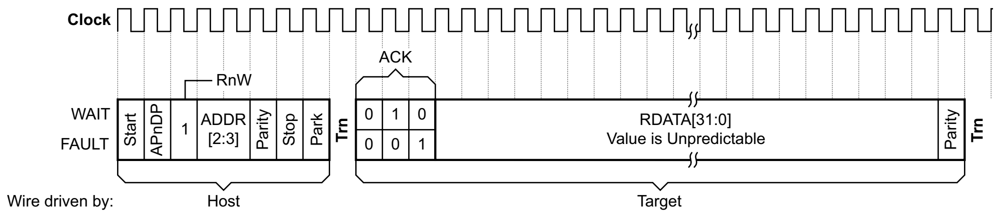


Figure 3-18 SW WAIT or FAULT response to a read operation when overrun detection is enabled

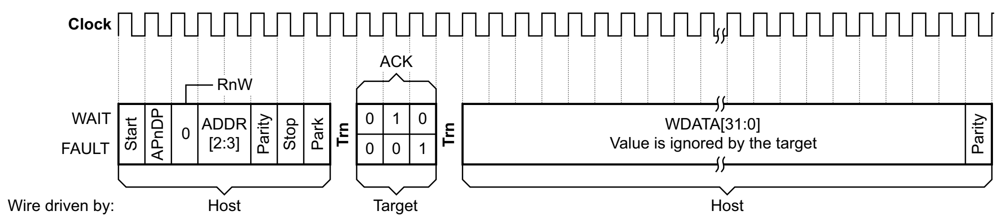


Figure 3-19 SW WAIT or FAULT response to a write operation when overrun detection is enabled

Protocol Error responses

If the SW-DP detects a parity error in the packet request it does not reply to the request.

When the host receives no reply to its request, it must back off, in case the SW-DP has lost frame synchronization for some reason. After this, it can issue a new transfer request. In this situation it must read the IDCODE register, see *Identification Code Register, IDCODE* on page 3-76. This is mandated by this specification because a successful read of the IDCODE register confirms that the target is operational.

If there is no response at the second attempt, the debugger must force a line reset to ensure frame synchronization and valid operation. This is necessary because the SW-DP is in a state where it only responds to a line reset. After the line reset the debugger must read the IDCODE register before it attempts any other operations.

If the transfer that resulted in the original protocol error response was a write you can assume that no write occurred. If the original transfer was a read it is possible that the read was issued to an access port. Although this is unlikely, you must consider this possibility because reads are pipelined and the debug port might implement a write buffer.

SW-DP write buffering

The SW-DP implements a write buffer, that enables it to accept write operations even when other transactions are still outstanding. The debug port issues an OK response to a write request if it can accept the write into its write buffer. This means that an OK response to a write request, other than a write to the DP ABORT Register, indicates only that the write has been accepted by the debug port. It does not indicate that all previous transactions have completed.

If a write is accepted into the write buffer but later abandoned then the WDATAERR flag is set in the CTRL/STAT Register, see *Control/Status Register, CTRL/STAT* on page 3-77. A buffered write is abandoned if:

- A sticky flag is set by a previous transaction.
- A debug port read of the IDCODE or CTRL/STAT Register is made. Because the debug port is not permitted to stall reads of these registers, it must:
  - perform the IDCODE or CTRL/STAT Register access immediately
  - discard any buffered writes, because otherwise they would be performed out-of-order.
- A debug port write of the ABORT Register is made. Again, this is because the debug port cannot stall an ABORT Register access.

This means that if you make a series of access port write transactions, it might not be possible to determine which transaction failed from examining the ACK responses. However it might be possible to use other enquiries to find which write failed. For example, if you are using the auto-address increment (AddrInc) feature of a Memory Access Port (AHB-AP), then you can read the Transfer Address Register to find which was the final successful write transaction. See *AHB-AP Transfer Address Register, TAR, 0x04* on page 3-91 and *AHB-AP register summary* on page 3-88 for more information.

The write buffer must be emptied before the following operations can be performed:

- any access port read operation
- any debug port operation other than a read of the IDCODE or CTRL/STAT Register, or a write of the ABORT Register.

Attempting these operations causes WAIT responses from the debug port, until the write buffer is empty.

Note

If Pushed Verify or Pushed Compare is enabled, access port write transactions are converted into AP reads. These are then treated in the same way as other access port read operations. See *Pushed compare and pushed verify operations* on page 3-61 for details of these operations.

If you have to perform a SW-DP read of the IDCODE or CTRL/STAT Register, or a SW-DP write to the ABORT Register immediately after a sequence of access port writes, you must first perform an access that the SW-DP is able to stall. In this way you can check that the write buffer is lost before performing the SW-DP register access. If this is not done, WDATAERR might be set, and the buffered writes lost.

Summary of target responses

Table 3-7 summarizes the target SW-DP response to all possible debugger debug port read operation requests.

Table 3-8 on page 3-52 summarizes the target SW-DP response to all possible debugger access port read operation requests.

Table 3-9 on page 3-52 summarizes the target SW-DP response to all possible debugger debug port write operation requests, assuming the WDATA parity check is good.

Table 3-10 on page 3-53 summarizes the target SW-DP response to all possible debugger access port write operation requests, assuming the WDATA parity check is good.

Fault conditions that are not shown in these two tables are described in *Fault conditions not included in the target response tables* on page 3-53

Table 3-7 Target response summary for DP read transaction requests

ADDR [3:2]	Sticky flag set?	AP Ready?	SW-DP (target) response	
			ACK	Action
b00	X	X	OK	Respond with IDCODE value.
b01	X	X	OK	Respond with CRTL/STAT or WCR value[a].
b10	No	Yes	OK	RESEND. Respond by resending the last read value sent to the host. This value is the result of one of: • the most recent AP read • the most recent DP RDBUF read.
b11	No	Yes	OK	Respond with RDBUF value from previous access port read, and set READOK flag in CTRL/STAT Register to 1.
b10	No	No	WAIT	No data phase, unless overrun detection is enabled[b].
b10	Yes	X	FAULT	No data phase, unless overrun detection is enabled[b].
b11	No	No	WAIT	No data phase, unless overrun detection is enabled[b]. Set READOK flag in CTRL/STAT Register to 0.
b11	Yes	X	FAULT	No data phase, unless overrun detection is enabled[b]. Set READOK flag in CTRL/STAT Register to 0.

a. Which value is returned depends on the value of the CTRLSEL bit in the SELECT Register. in the debug port. See *AP Select Register, SELECT* on page 3-81.

b. See *Sticky overrun behavior* on page 3-48 for details of data phase when overrun detection is enabled.

Table 3-8 Target response summary for AP read transaction requests

ADDR [3:2]	Sticky flag set?	AP Ready?	SW-DP (target) response	
			ACK	**Action**
bXX	No	Yes	OK	Normally[a], return value from previous access port read[b] and set READOK flag in CTRL/STAT Register. Initiate AP read of addressed register[c].
bXX	No	No	WAIT	No data phase, unless overrun detection is enabled[d]. Set READOK flag in CTRL/STAT Register to 0.
bXX	Yes	X	FAULT	No data phase, unless overrun detection is enabled[d]. Set READOK flag in CTRL/STAT Register to 0.

a. If Pushed Verify or Pushed Compare is enabled, behavior is Unpredictable.
b. On the first of a sequence of AP reads, the value returned in the data phase is Unpredictable.
c. The AP register is addressed by the value of A[3:2] together with the value of the APBANKSEL field in the SELECT Register in the DP. See *AP Select Register, SELECT* on page 3-81.
d. See *Sticky overrun behavior* on page 3-48 for details of data phase when overrun detection is enabled.

Table 3-9 Target response summary for DP write transaction requests

ADDR [3:2]	Sticky flag set?	AP Ready?	SW-DP (target) response	
			ACK	**Action**
b00	X	X	OK	Write WDATA value to ABORT Register.
Not b00	No	Yes[a]	OK	Write WDATA value to debug port register indicated by ADDR[3:2].
Not b00	No	No	WAIT	No data phase, unless overrun detection is enabled[b].
Not b00	Yes	X	FAULT	No data phase, unless overrun detection is enabled[b].

a. Writes might be accepted when other transactions are still outstanding, These writes might be abandoned subsequently. See *SW-DP write buffering* on page 3-49 for more information.
b. See *Sticky overrun behavior* on page 3-48 for details of data phase when overrun detection is enabled.

Table 3-10 Target response summary for AP write transaction requests

ADDR [3:2]	Sticky flag set?	AP Ready?	SW-DP (target) response	
			ACK	Action
bXX	No	Yes[a]	OK	Normally[b], write WDATA value to the indicated access port register[c].
bXX	No	No	WAIT	No data phase, unless overrun detection is enabled[d].
bXX	Yes	X	FAULT	No data phase, unless overrun detection is enabled[d].

a. Writes might be accepted when other transactions are still outstanding, These writes might be abandoned subsequently. See *SW-DP write buffering* on page 3-49 for more information.

b. If Pushed Verify or Pushed Compare is enabled, the write is converted to a read of the addressed AP register, and the value returned by this read is compared with the supplied WDATA value, see *Pushed compare and pushed verify operations* on page 3-61 for more information. For an outline of how AP registers are addressed see footnote [c] to this table.

c. The AP register is addressed by the value of A[3:2] together with the value of the APBANKSEL field in the SELECT Register in the DP See *AP Select Register, SELECT* on page 3-81.

d. See *Sticky overrun behavior* on page 3-48 for details of data phase when overrun detection is enabled.

Fault conditions not included in the target response tables

There are two fault conditions that are not included in possible operation requests listed in Table 3-7 on page 3-51 and Table 3-9 on page 3-52:

Protocol fault

If there is a protocol fault in the operation request then the target does not respond to the request at all. This means that when the host expects an ACK response, it finds that the line is not driven.

WDATA fails parity check (write operations only)

The ACK response of the debug port is sent before the parity check is performed, and can be found from Table 3-9 on page 3-52. When the parity check is performed and fails, the WDATAERR flag is set in the CTRL/STAT Register, see *Control/Status Register, CTRL/STAT* on page 3-77.

Summary of host responses

Every access by a debugger to a SW-DP starts with an operation request. *Summary of target responses* on page 3-51 listed all possible requests from a debugger, and summarized how the SW-DP responds to each request.

Whenever a debugger issues an operation request to a SW-DP, it expects to receive a 3-bit acknowledgement, as listed in the ACK columns of Table 3-7 on page 3-51 and Table 3-7 on page 3-51. This section summarizes how the debugger must respond to this acknowledgement, for all possible cases. This is shown in Table 3-11.

Table 3-11 Summary of host (debugger) responses to the SW-DP acknowledge

Operation requested	ACK received	Host response	
		Data phase	**Additional action**
R	OK	Capture RDATA from target and check for valid parity and protocol.	Might have to re-issue original read request or use the RESEND register if a parity or protocol fault occurs and are unable to flag data as invalid[a].
W	OK	Send WDATA.	Validity of this transfer is confirmed on next access.
X	WAIT	No data phase, unless overrun detection is enabled[b].	Normally, repeat the original operation request. See *WAIT response* on page 3-47 for more information.
X	FAULT	No data phase, unless overrun detection is enabled[b].	Can send new headers, but only an access to debug port register addresses b0X gives a valid response.
X	No ACK	Back off to allow for possible data phase.	Can attempt IDCODE Register read. Otherwise reset connection and retrain. See *Protocol Error responses* on page 3-49.
R	Invalid ACK	Back off to allow for possible data phase.	Can check CTRL/STAT Register to see if the response sent was OK.
W	Invalid ACK	Back off to ensure that target does not capture next header as WDATA.	Repeat the write access. A FAULT response is possible if the first response was sent as OK but not recognized as valid by the debugger. The subsequent write is not affected by the first, misread, response.

a. The host debugger might be able to support corrupted reads, or it might have to re-try the transfer.

b. If overrun detection is enabled, a data phase is required. On a read operation, the RDATA value is Unpredictable and the debugger must capture and discard this data. On a write operation the debugger must send a WDATA packet, that the target ignores.

3.6.5 Transfer timings

This section describes the interaction between the timing of transactions on the serial wire interface, and the DAP internal bus transfers. It shows when the target responds with a WAIT acknowledgement.

Figure 3-20 shows the effect of signalling ACK = WAIT on the length of the packet.

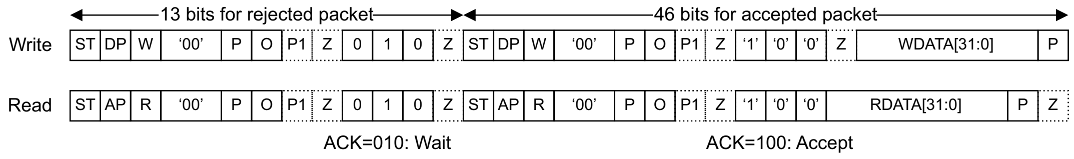

13 bits for rejected packet
46 bits for accepted packet
Write
ST DP W '00' P O P1 Z 0 1 0 Z ST DP W '00' P O P1 Z '1' '0' '0' Z WDATA[31:0] P
Read
ST AP R '00' P O P1 Z 0 1 0 Z ST AP R '00' P O P1 Z '1' '0' '0' RDATA[31:0] P Z
ACK=010: Wait
ACK=100: Accept

Figure 3-20 SW-DP acknowledgement timing

An access port access results in the generation of a transfer on the DAP internal bus. These transfers have an address phase and a data phase. The data phase can be extended by the access if it requires extra time to process the transaction, for example, if it has to perform an AHB access to the system bus to read data.

Table 3-12 shows the terms used in Figure 3-21 on page 3-56 to Figure 3-23 on page 3-57.

Table 3-12 Terms used in SW-DP timing

Term	Description
W.APACC	Write a DAP access port register.
R.APACC	Read a DAP access port register.
xxPACC	Read or write, to debug port or access port register.
WD[0]	First write packet data.
WD[-1]	Previous write packet data. A transaction that happened before the figures timeframe.
WD[1]	Second write packet data.
RD[0]	First read packet data.
RD[1]	Second read packet data.

Figure 3-21 shows a sequence of write transfers. It shows that a single new transfer, WD[1], can be accepted by the serial engine, while a previous write transfer, WD[0], is completing. Any subsequent transfer must be stalled until the first transfer completes.

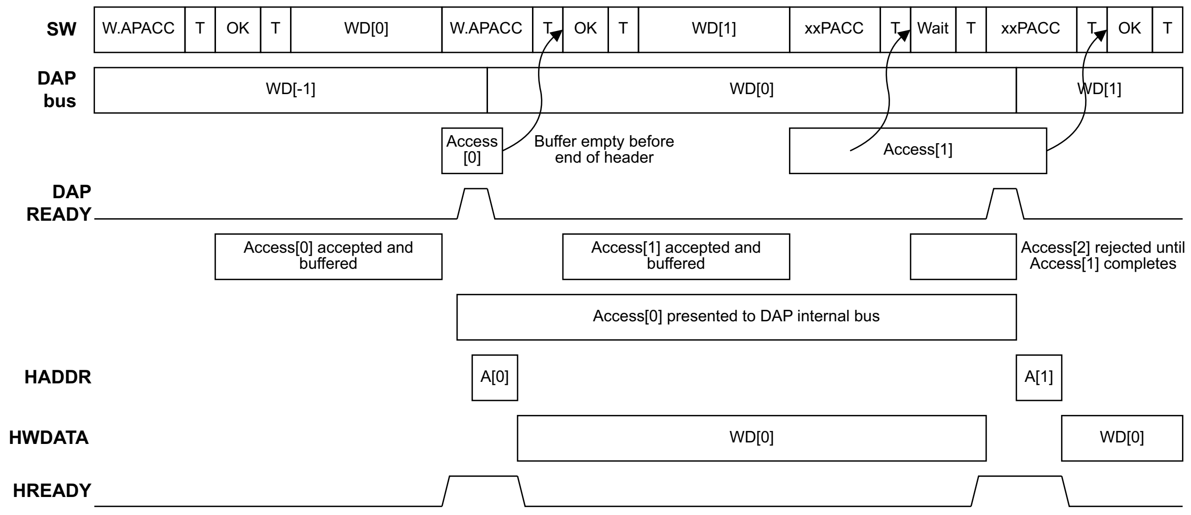

SW
W.APACC
T
OK
T
WD[0]
W.APACC
T
OK
T
WD[1]
xxPACC
T
Wait
T
xxPACC
T
OK
T
DAP bus
WD[-1]
WD[0]
WD[1]
Access [0]
Buffer empty before end of header
Access[1]
DAP READY
Access[0] accepted and buffered
Access[1] accepted and buffered
Access[2] rejected until Access[1] completes
Access[0] presented to DAP internal bus
HADDR
A[0]
A[1]
HWDATA
WD[0]
WD[0]
HREADY

Figure 3-21 SW-DP to DAP bus timing for writes

Figure 3-22 shows a sequence of read transfers. It shows that the payload for an access port read transfer provides the data for the previous read request. A read transfer only stalls if the previous transfer has not completed, therefore the first read transfer returns undefined data. It is still necessary to return data to ensure that the protocol timing remains predictable.

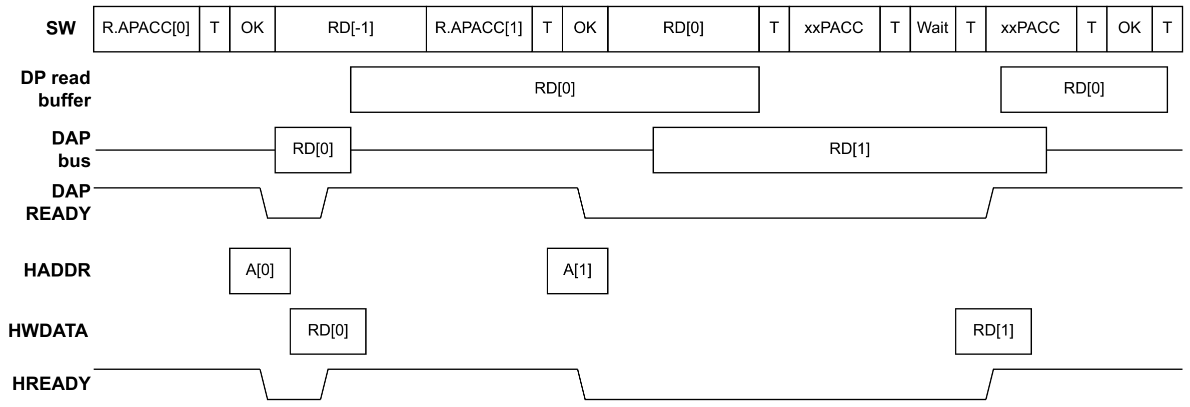


Figure 3-22 SW-DP to DAP bus timing for reads

Figure 3-23 shows a sequence of transfers separated by IDLE periods. It shows that the wire is always handed back to the host after any transfer.

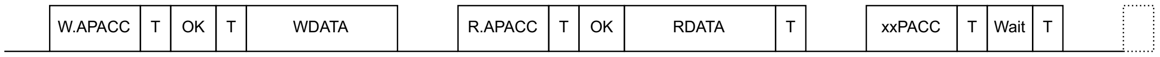


Figure 3-23 SW-DP idle timing

After the last bit in a packet, the line can be LOW, or idle, for any period longer than a single bit, to enable the Start bit to be detected for back-to-back transactions.

3.7 Common Debug Port features

This section describes features that are implemented by the SW-DP and JTAG-DP as part of the SWJ-DP. These common features affect the way that a debugger is able to perform transactions with the DAP. It contains the following:

- *Sticky flags and debug port error responses*.

3.7.1 Sticky flags and debug port error responses

In the SW-DP and JTAG-DP, sticky flags are used to indicate error conditions and to report the result of pushed compare and pushed verify operations. The different sticky flags are described in the following sections:

- *Read and write errors* on page 3-59
- *Overrun detection* on page 3-59
- *Protocol errors, SW-DP only* on page 3-60
- *Pushed compare and pushed verify operations* on page 3-61.

Note

When set to 1, a sticky flag remains set until it is explicitly cleared to 0. Even if the condition that caused the flag to be set no longer applies, the flag remains set until the debugger clears it. The method for clearing sticky flags is different for the SW-DP and JTAG-DP. See *Control/Status Register, CTRL/STAT* on page 3-77 for information about how these flags are cleared.

Errors can be returned by the DAP itself, or might come from a debug resource, for example, from a memory access made by a MEM-AP to a debug register file of a processor that is powered down.

In the debug port, errors are flagged by sticky flags in the DP Control/Status Register (CTRL/STAT). When an error is flagged the current transaction is completed and further APACC (AP Access) transactions are discarded until the sticky flag is cleared.

The debug port response to an error condition might be:

- To signal an error response immediately. This happens with the SW-DP.
- To immediately discard all transactions as complete. This happens with the JTAG-DP.

This means that a debugger must check the Control/Status Register after performing a series of APACC transactions, to check if an error occurred. If a sticky flag is set to 1, the debugger clears the flag to 0 and then, if necessary, initiates more APACC transactions to find the cause of the sticky flag condition. Because the flags are sticky the debugger does not have to check the flags after every transaction, it only has to check the Control/Status Register periodically. This reduces the overhead of checking for errors.

3.7.2 Read and write errors

A read or write error might occur in the debug port, or come from the system being debugged as the result of a (AHB-AP) access in response to an access port request. In either case, when the error is detected the Sticky Error flag, STICKYERR, in the Control/Status Register is set to b1.

A read/write error is also generated if the debugger makes an access port transaction request while the debug power domain is powered down.

3.7.3 Overrun detection

Debug ports support an overrun detection mode. This mode enables an emulator on a high latency, high throughput connection to be sent blocks of commands. These should be sent with sufficient in-line delays to make overrun errors unlikely. However, if an overrun error occurs, the debug port detects and flags the overrun errors, by setting a flag in the Control/Status Register. In overrun detection mode the debugger must check for overrun errors after each sequence of APACC transactions, by checking the Sticky Overrun flag in the Control/Status Register. It is not necessary for the emulator to react immediately to the overrun condition.

Overrun detection mode is enabled by setting the Overrun Detect bit, ORUNDETECT, in the DP Control/Status Register. When this bit is set, the only allowed response to any transaction is:

- OK/FAULT on the JTAG-DP
- OK on the SW-DP.

In overrun detection mode, any other response, at any point, is treated as an error and causes the Sticky Overrun flag STICKYORUN in the DP Control/Status Register to be set to b1. The Sticky Error flag, STICKYERR, is not set.

The debugger must clear STICKYORUN to 0 to enable transactions to resume.

See *Control/Status Register, CTRL/STAT* on page 3-77 for more information.

Note

The method of clearing the STICKYORUN flag to 0 is different for a JTAG-DP and a SW-DP:

- This bit is read-only for SW-DP.
- On a JTAG-DP, this bit can be read normally, and writing 1 to this bit clears the bit to 0

The behavior of the debug port when the Sticky Overrun flag is set is debug port defined.

If a new transaction is attempted, and results in an overrun error, before an earlier transaction has completed, the first transaction still completes normally. Other sticky flags might be set on completion of the first transaction.

If the overrun detection mode is disabled, by clearing the ORUNDETECT flag, while STICKYORUN is set, the subsequent value of STICKYORUN is Unpredictable. To leave overrun detection mode a debugger must:

- check the value of the STICKYORUN bit in the Control/Status register
- clear the STICKYORUN bit, if it is set
- clear the ORUNDETECT bit, to stop overrun detection mode.

3.7.4 Protocol errors, SW-DP only

Note

Although these errors can only be detected with the SW-DP, they are described in this chapter because they are part of the sticky flags error handling mechanism.

On the SWD interface, protocol errors can only occur, for example because of wire-level errors. These errors might be detected by the parity checks on the data.

If the SW-DP detects a parity error in a message header, the debug port does not respond to the message. The debugger must be aware of this possibility. If it does not receive a response to a message, the debugger must back off. It must then request a read of the IDCODE register, to ensure the debug port is responsive, before retrying the original access. For details of the IDCODE register see *Identification Code Register, IDCODE* on page 3-76.

If the SW-DP detects a parity error in the data phase of a write transaction, it sets the Sticky Write Data Error flag, WDATAERR, in the Control/Status (CTRL/STAT) Register. Subsequent accesses from the debugger, other than IDCODE, CTRL/STAT or ABORT, results in a FAULT response. For details of the CTRL/STAT register see *Control/Status Register, CTRL/STAT* on page 3-77.

On receiving a FAULT response from the SW-DP a debugger must read the CTRL/STAT register and check the sticky flag values. The WDATAERR flag is cleared by writing b1 to the WDERRCLR field of the Abort Register, see *Abort Register, ABORT* on page 3-74.

3.7.5 Pushed compare and pushed verify operations

The SW-DP and JTAG-DP debug ports support pushed operations, where the value written as an access port transaction is used at the debug port level to compare against a target read:

- the debugger writes a value as an access port transaction
- the debug port performs a read from the access port
- the debug port compares the two values and updates the Sticky Compare flag, STICKYCMP, in the DP Control/Status register, based on the result of the comparison:
  - pushed compare sets STICKYCMP to b1 if the values match
  - pushed verify sets STICKYCMP to b1 if the values do not match.

  Whenever the STICKYCMP bit is set, on detection of a valid comparison, any outstanding transaction repeats are cancelled.

  For more information see *Control/Status Register, CTRL/STAT* on page 3-77.

The debug port includes a byte lane mask, so that the compare can be restricted to particular bytes in the word. This mask is set using the MASKLANE bits in the Control/Status register. For more information about this masking see *MASKLANE and the bit masking of the pushed compare and pushed verify operations* on page 3-80.

Figure 3-24 gives an overview of the pushed operations.

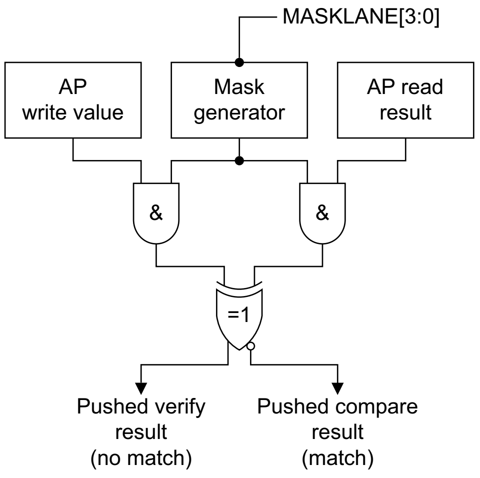

MASKLANE[3:0]
AP write value
Mask generator
AP read result
&
&
=1
Pushed verify result (no match)
Pushed compare result (match)

Figure 3-24 Pushed operations overview

Pushed operations improve performance where writes might be faster than reads. They are used as part of in-line tests, for example Flash ROM programming and monitor communication. Pushed operations are enabled using the Transaction Mode bits, TRNMODE, in the DP Control/Status Register, see *Control/Status Register, CTRL/STAT* on page 3-77.

Considering pushed operations on a specific access port makes it easier to understand how these operations are implemented. On an AHB-AP, if you perform an access port write transaction to the Data Read/Write (DRW) Register, or to one of the Banked Data (BD0 to BD3) Registers, with either pushed compare or pushed verify active:

- The debug port holds the data value from the access port write transaction in the pushed compare logic, see Figure 3-24 on page 3-61.
- The access port reads from the address indicated by the AP *Transfer Address Register* (TAR), see *AHB-AP Transfer Address Register, TAR, 0x04* on page 3-91.
- The value returned by this read is compared with the value held in the pushed compare logic, and the STICKYCMP bit is set depending on the result. The comparison is masked as required by the MASKLANE bits. For more information see *Control/Status Register, CTRL/STAT* on page 3-77.

As described, whenever an access port *write* transaction is performed with pushed compare or pushed verify active, the actual access port access that results is a *read* operation, not a write.

——— **Note** ———

Performing an access port read transaction with pushed compare or pushed verify active causes Unpredictable behavior.

On a SW-DP, performing an access port read transaction with pushed compare or pushed verify active returns a value. This means the wire-level protocol remains coherent. However, the value returned is Unpredictable, and the read has Unpredictable side-effects.

———————————

Example use of pushed verify operation on a AHB-AP

You can use pushed verify to verify the contents of system memory.

- Make sure that the AHB-AP Control/Status Word (CSW) is set up to increment the Transfer Address Register (TAR) after each access. See *Control/Status Register, CTRL/STAT* on page 3-77.
- Write to the Transfer Address Register (TAR) to indicate the start address of the Debug Register region that is to be verified, see *AHB-AP Transfer Address Register, TAR, 0x04* on page 3-91.

- Write a series of expected values as access port transactions. On each write transaction, the debug port issues an access port read access, compares the result against the value supplied in the access port write transaction, and sets the STICKYCMP bit in the CRL/STAT Register if the values do not match. See *Control/Status Register, CTRL/STAT* on page 3-77.

  The TAR is incremented on each transaction.

In this way, the series of values supplied is compared against the contents of the access port locations, and STICKYCMP set if they do not match.

Example use of pushed find operation on a AHB-AP

You can use pushed find to search system memory for a particular word. If you use pushed find with byte lane masking you can search for one or more bytes.

- Make sure that the AHB-AP *Control/Status Word* (CSW) is set up to increment the TAR after each access. See *Control/Status Register, CTRL/STAT* on page 3-77.
- Write to the *Transfer Address Register* (TAR) to indicate the start address of the Debug Register region that is to be searched. See *AHB-AP Transfer Address Register, TAR, 0x04* on page 3-91.
- Write the value to be searched for as an AP write transaction. The debug port repeatedly reads the location indicated by the TAR. On each debug port read:
  - The value returned is compared with the value supplied in the access port write transaction. If they match, the STICKYCMP flag is set.
  - The TAR is incremented.

  This continues until STICKYCMP is set, or ABORT is used to terminate the search.

You could also use pushed find without address incrementing to poll a single location, for example to test for a flag being set on completion of an operation.

3.8 Transaction counter

The debug port includes an access port Transaction Counter, TRNCNT. The Transaction Counter enables a debugger to make a single access port transaction request that generates a sequence of access port transactions. With a AHB-AP or APB-AP access, the access port transaction sequence might generate a sequence of accesses to the connected memory system.

Examples of the use of the Transaction Counter are:

- For a code download or memory fill operation. For memory fill, the Transaction Counter can be used to repeatedly write a single data value supplied in the initial access port transaction request. The AHB-AP includes a mechanism that auto-increments the access address after each access port access. This means that a series of access port accesses under the control of the transaction counter write the supplied data value to a sequence of memory addresses. For more information see *Packed transfers* on page 3-95.
- With pushed compare or pushed verify operation enabled, the Transaction Counter can be used when reading from the Data Read/Write Register, to perform a fast search or verify of an area of memory. This use is mentioned in *Pushed compare and pushed verify operations* on page 3-61, and is described in more detail in *Example use of pushed find operation on a AHB-AP* on page 3-63.

A field in the DP Control/Status Register, bits [23:12], maps onto the Transaction Counter. Writing a value other than zero to this field generates multiple access port transactions, see *Control/Status Register, CTRL/STAT* on page 3-77. For example, writing `0x001` to this field generates two AP transactions, and writing `0x002` generates three transactions,.

The Transaction Counter does not auto-reload when it reaches zero.

If the Transaction Counter is not zero, it is decremented after each successful transaction. The transaction counter is not decremented and the transaction is not repeated if one of the following is true:
- the Transaction Counter is zero
- the Sticky Error flag, in the Control/Status Register, is set to 1
- the Sticky Compare flag, in the Control/Status Register, is set to 1.

If a sequence of operations is terminated because the Sticky Error or Sticky Compare flag was set to 1, the Transaction Counter remains at the value from the last successful transaction. This means that software can recover the location of the error, or determine where the compare or verify operation terminated.

3.9 System and debug power and debug reset control

The debug port provides:

- Four control bits for system and debug power control. The use of these is described in:
  - — *DAP power domains model*
  - — *Power control requirements and operation* on page 3-66
  - — *Emulation of power-down* on page 3-68
  - — *Emulation of power control* on page 3-69.
- Two control bits for debug reset control, see *Debug reset control* on page 3-70.

These control bits are programmable by the debugger, and drive signals into the connected system. These signals are intended as hints or stimuli into existing power and reset controllers.

The Debug Interface does not replace the system power and reset controllers, and the Debug Interface specification does not place any requirements on the operation of the system power and reset controllers. However, this section provides a model of how these signals might be used.

3.9.1 DAP power domains model

The DAP model supports multiple power domains. These provide support for debug components that can be powered off.

Three power domains are modelled:

Always-on power domain

This must be powered on for the debugger to connect to the device.

System power domain

This includes all the system components.

Debug power domain

This includes all of the debug subsystem.

The last two domains could be subdivided if necessary. However, to define a simple debug interface, the device must be partitioned into System and Debug power domains at its top level. Any finer-grained control is outside the scope of this model.

The debug port registers reside in the Always-on power domain, on the external interface side of the debug port. Therefore, they can always be driven, enabling power-up requests to be made to a system power controller. The power and reset control bits are part of the DP Control/Status register, see *Control/Status Register, CTRL/STAT* on page 3-77. See *Debug reset control* on page 3-70 for more information about the reset control bits in this register. Four bits of the register provide power control signals:

Bit [28], CDBGPWRUPREQ

CDBGPWRUPREQ is the signal from the debug interface to the power controller, used to request the system power controller to fully power-up and enable clocks in the debug power domain.

Bit [29], CDBGPWRUPACK

CDBGPWRUPACK is the signal from the power controller to the debug interface. When **CDBGPWRUPREQ** is asserted, the power controller powers-up the debug power domain and then asserts **CDBGPWRUPACK** to acknowledge that it has responded to the request.

Bit [30], CSYSPWRUPREQ

CSYSPWRUPREQ is the signal from the debug interface to the power controller, used to request the system power controller to fully power-up and enable clocks in the system power domain.

Bit [31], CSYSPWRUPACK

CSYSPWRUPACK is the signal from the power controller to the debug interface. When **CSYSPWRUPREQ** is asserted, the power controller powers-up the system power domain and then asserts **CSYSPWRUPACK** to acknowledge that it has responded to the request.

In most situations, debuggers power-up the complete SoC. However, if a debugger is being used to investigate an energy management issue, it might want to power-up only the debug domain. To enable this possibility, you can map the power controller into a bus segment that the DAP can access when only the debug power domain is powered on.

When using an ARM Debug Interface, for the debug process to work correctly, systems must not remove power from the debug port during a debug session. If power is removed, the DAP controller state is lost. However, the DAP is designed to permit the rest of the DAP and the core to be powered down and debugged while maintaining power to the debug port.

3.9.2 Power control requirements and operation

The following description applies to both:

- system domain power up, using the **CSYSPWRUPREQ** and **CSYSPWRUPACK** signals
- debug domain power up, using the **CDBGPWRUPREQ** and **CDBGPWRUPACK** signals.

Therefore, in the description:

- **CxxxPWRUPREQ** refers to either **CSYSPWRUPREQ** or **CDBGPWRUPREQ**
- **CxxxPWRUPACK** refers to either **CSYSPWRUPACK** or **CDBGPWRUPACK**.

The rules for the operation of power-up requests and acknowledgements are:

- To initiate power-on, **CxxxPWRUPREQ** must be asserted HIGH by the debug port.
  - — If the corresponding power domain is powered down or in a low-power retention state, the power controller must power up and restore clocks to the domain when it detects **CxxxPWRUPREQ** asserted HIGH. When the domain is powered up, the controller must assert **CxxxPWRUPACK** HIGH.
  - — If the corresponding power domain is already powered up and being clocked when the power controller detects **CxxxPWRUPREQ** asserted HIGH, the controller must still respond with **CxxxPWRUPACK** HIGH. However, there is no effect on the powered-up domain.
- Tools can only initiate a DAP transfer when both **CxxxPWRUPREQ** and **CxxxPWRUPACK** are asserted HIGH for one of the pairs of power control signals. When both **CxxxPWRUPREQ** and **CxxxPWRUPACK** are asserted HIGH, the corresponding power domain is powered on.
- The removal of power to a domain is requested by the debug port deasserting **CxxxPWRUPREQ**, that is, by the debug port taking **CxxxPWRUPREQ** LOW.

  The power controller deasserts **CxxxPWRUPACK** when it has accepted the request to power-down the domain.

  CxxxPWRUPACK being taken LOW by the power controller does not indicate that the domain has been powered down, it only indicates that the power controller has recognized the request to remove power. In effect, **CxxxPWRUPACK** acts as a logical AND of the corresponding **CxxxPWRUPREQ** and **CACTIVE** signals.
- **CxxxPWRUPACK** must default to the LOW state, and only go HIGH on receipt of a **CxxxPWRUPREQ** request.
- On detecting the deassertion of **CxxxPWRUPREQ**, indicated by the signal going LOW, the power controller must gracefully power-down the domain, unless removal of power from the domain would affect system operation. For example, the power controller might maintain power to the domain if it has other requests to maintain power.

- After power-down has been requested, by the deassertion of **CxxxPWRUPREQ**, tools must wait until **CxxxPWRUPACK** is LOW before making a new request for power-up. **CxxxPWRUPACK** going LOW indicates that the power controller has recognized the original power-down request.

  This requirement ensures that the power control handshaking mechanism is not violated.

——— **Note** ———

The AMBA v3 AXI low-power clock control signals **CSYSREQ** and **CSYSACK** have similar behavior to the *ARM Debug Interface* (ADI) **CSYSPWRUPREQ** and **CSYSPWRUPACK** signals. However:

- the AXI signals request entry to a low-power state, and acknowledge the request
- the ADI signals request full-power state, and acknowledge the request.

————————

Figure 3-25 shows the timing of the power control signals.

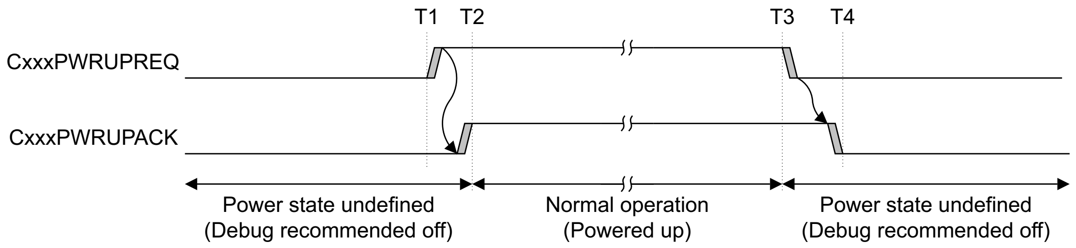


Figure 3-25 Power-up request and acknowledgement timing

——— **Note** ———

All AP transactions must be initiated between times T2 and T3 for **CDBGPWRUPREQ** and **CDBGPWRUPACK**, as shown in Figure 3-25.

————————

3.9.3 Emulation of power-down

When **CxxxPRWUPREQ** from the ADI is asserted HIGH for a domain, if the power controller receives a conflicting request for the domain from another source it must emulate the power-down request for the domain. This applies if the power controller receives, from the other source, either of:

- a power-down request
- a request to enter a low-power retention mode, with clocks disabled.

This requirement makes it possible to debug a system where one domain powers up and down dynamically. An example of a system that requires this is an IEM-enabled ARM core.

During emulation of the power-down request, the power controller carries out all of the expected power control handshaking, but does not actually remove power from the domain.

Emulation of power-down is particularly relevant to application debugging, when the application developer does not care whether the core domain actually powers up and down because this is controlled at the OS level.

3.9.4 Emulation of power control

Where the system to which an ADI is connected does not support the ADI power control model, the required signals must be emulated or generated from other signals. Three possible cases are described here.

System without power controller support for the ADI control scheme

If the connected system can not support the ADI power-up and power-down control scheme, then **CxxxPRWUPACK** must be connected to **CxxxPRWUPREQ**. With these connections, whenever the ADI requests power-up, or removes a power-up request, it receives the appropriate acknowledge immediately. This power control emulation is shown in Figure 3-26.

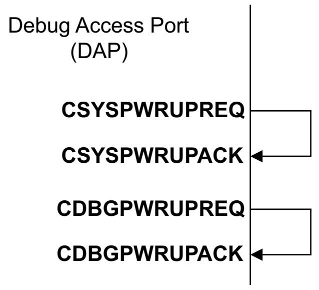


Figure 3-26 Emulation of power-up control

System power controller does acknowledge power-up requests

If the system power controller does not provide ACK responses to power-up requests, but supplies an **ACTIVE** signal when a domain is powered-up, the **CxxxPRWUPACK** signals must be generated. This is shown in Figure 3-27 on page 3-70.

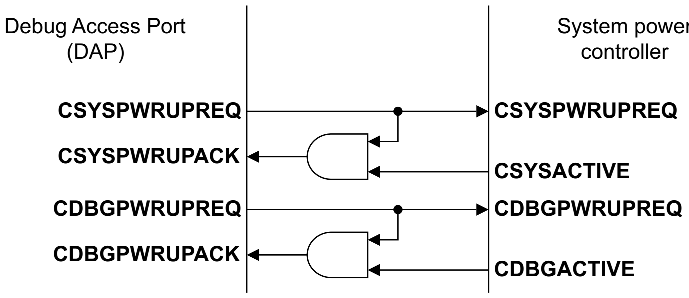


Figure 3-27 Generation of ACK signals from REQ and ACTIVE signals

System power controller does not support separate power domains

If the system power controller does not support separate debug and system power domains, then the two **CxxxPRWUPREQ** power-up request signals must be ORed together to provide a single power-up request to the controller. However, the **CxxxPRWUPACK** signals must be generated, so that the DAP sees the correct response to asserting a **CxxxPRWUPREQ** signal. This is shown in Figure 3-28.

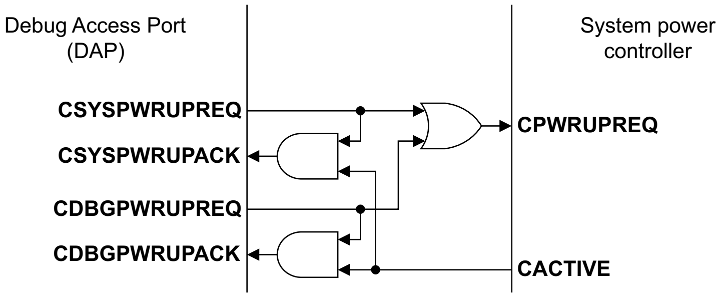


Figure 3-28 Signal generation for a single power domain

3.9.5 Debug reset control

The Control/Status Register provides two bits, bits [27:26], for reset control of the debug domain, see *Control/Status Register, CTRL/STAT* on page 3-77. The debug domain controlled by these signals covers the internal DAP and the connection between the DAP and the debug components, for example the debug bus. The two bits provide a debug reset request, **CDBGRSTREQ**, and a reset acknowledge, **CDBGRSTACK**. The associated signals provide a connection to a system reset controller.

The debug port registers are in the always-on power domain on the external interface side of the debug port. Therefore, the registers can be driven at any time, to generate a reset request to the system reset controller.

Figure 3-29 shows the reset request and acknowledge timing.

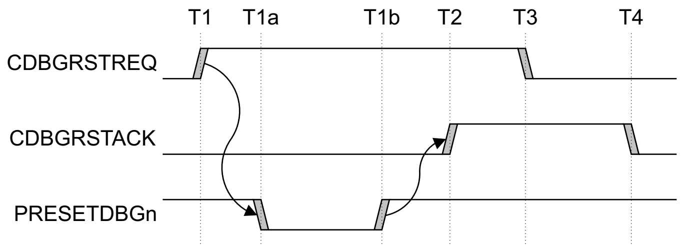


Figure 3-29 Reset request and acknowledge timing

In Figure 3-29:

1. At time T1, the debugger writes to the CDBGRSTREQ bit, bit [26] of the Control/Status Register. This initiates the reset request.

   The debug domain is reset between times T1a and T1b, and the reset is complete by time T2. This operation resets the access port registers and other access port state.

   Note

   There is no reset of the debug port registers and debug port state. These are only reset by a power-on reset.

2. At time T2, the system reset controller acknowledges that the reset of the debug domain has completed. The **CDBGRSTACK** signal sets the corresponding bit, bit [27], in the Control/Status Register.
3. At time T3, the debugger checks the Control/Status Register and finds that the reset has completed. Therefore, it writes to the Control/Status register, to clear the CDBGRSTREQ bit to 0. This removes the reset request signal.
4. At time T4, the system reset controller recognizes that **CDBGRSTREQ** is no longer asserted, and deasserts **CDBGRSTACK**.

Caution

If **CDBGRSTREQ** is removed before the reset controller asserts **CDBGRSTACK**. the behavior is unpredictable.

The access port debug components are also reset on power-up of the debug power domain.

A debug reset request has no effect on devices that are powered down when the request is issued.

3.10 Debug port programmer's model

The CoreSight DK includes a SWJ-DP for selection of SWD or JTAG as the debug access mechanism. The SWJ-DP uses the programmer's models of the JTAG-DP and SW-DP.

This section contains:

- *JTAG-DP registers*. This contains a summary of the JTAG-DP registers.
- *SW-DP registers* on page 3-73. This contains a summary of the SW-DP registers.
- *Debug port register descriptions* on page 3-74. This contains details of the DP registers, and describes implementation differences between SW-DP and JTAG-DP registers.

3.10.1 JTAG-DP registers

The JTAG-DP register accessed depends on both:

- the Instruction Register (IR) value for the DAP access
- the address field of the DAP access.

For more information, see *Accessing the JTAG-DP registers* on page 3-73.

Table 3-13 shows the JTAG-DP register map.

Table 3-13 JTAG-DP register map

IR contents	Description	Address	Access	Reference	Notes
IDCODE	ID Code Register	-[a]	RO	*Identification Code Register, IDCODE* on page 3-76	-
DPACC	-	`0x0`	RAZ/WI	-	Reserved. Read-as-zero, Writes ignored.
DPACC	DP Control/Status Register	`0x4`	R/W	*Control/Status Register, CTRL/STAT* on page 3-77	-
DPACC	Select Register	`0x8`	R/W	*AP Select Register, SELECT* on page 3-81	-
DPACC	Read Buffer	`0xC`	RAZ/WI	*Read Buffer, RDBUFF* on page 3-83	-
ABORT	DAP Abort Register	`0x0`	WO[b]	*Abort Register, ABORT* on page 3-74	-
ABORT	-	`0x4` - `0xC`	-	-	- b

a. There is no address associated with IDCODE accesses. See *Accessing the JTAG-DP registers* on page 3-73.

b. The value read on the ABORT scan chain is Unpredictable. The result of accessing the ABORT scan chain with the address field not set to `0x0` is Unpredictable

Accessing the JTAG-DP registers

The JTAG-DP registers are only accessed when the *Instruction Register* (IR) for the DAP access contains the IDCODE, DPACC, or ABORT instruction. In detail, the register accesses for each instruction are:

IDCODE The IDCODE scan chain has no address field, and accesses the IDCODE register.

DPACC The DPACC scan chain accesses registers at addresses `0x0` to `0xC`.

ABORT For a write access with address `0x0`, the ABORT scan chain accesses the ABORT register.

For a read access with address `0x0`, and for any access with address `0x4` to `0xC`, the behavior of the ABORT scan chain is Unpredictable.

3.10.2 SW-DP registers

For most register addresses on the SW-DP, different registers are addressed on read and write accesses. In addition, the CTRLSEL bit in the Select Register changes which register is accessed at address 0b01.

Table 3-14 shows the SW-DP register map.

Table 3-14 SW-DP register map

Address	CTRLSEL[a]	Description	Access[b]	Reference
b00	X	ID Code Register	R	*Identification Code Register, IDCODE* on page 3-76
		Abort Register	W	*Abort Register, ABORT* on page 3-74
b01	b0	Control/Status Register	R/W	*Control/Status Register, CTRL/STAT* on page 3-77
	b1	Wire Control Register	R/W	*Wire Control Register, WCR (SW-DP only)* on page 3-84
b10	X	Read Resend Register	R	*Read Resend Register, RESEND (SW-DP only)* on page 3-86
		Select Register	W	*AP Select Register, SELECT* on page 3-81
b11	X	Read Buffer	R	*Read Buffer, RDBUFF* on page 3-83
		-	W	-

a. CTRLSEL bit in the SELECT register, see *AP Select Register, SELECT* on page 3-81.
b. Entries in the Access column refer to whether the SWD protocol makes a read or a write access to the given address.

3.10.3 Debug port register descriptions

This section gives a detailed description of each of the debug port registers. Each description states whether the register is implemented for the JTAG-DP and for the SW-DP, and any differences in the implementation.

Abort Register, ABORT

The Abort Register is always present on all debug port implementations. Its main purpose is to force a DAP abort, and on a SW-DP it is also used to clear error and sticky flag conditions.

JTAG-DP It is at address `0x0` when the *Instruction Register* (IR) contains ABORT.

SW-DP It is at address 0b00 on write operations when the APnDP bit =1, see *Key to illustrations of operations* on page 3-39. Access to the Abort Register is not affected by the value of the CTRLSEL bit in the Select Register.

It is:

- A write-only register.
- Always accessible, and returns an OK response if a valid transaction is received. Abort Register accesses always complete on the first attempt.

Figure 3-30 shows the Abort Register bit assignments.

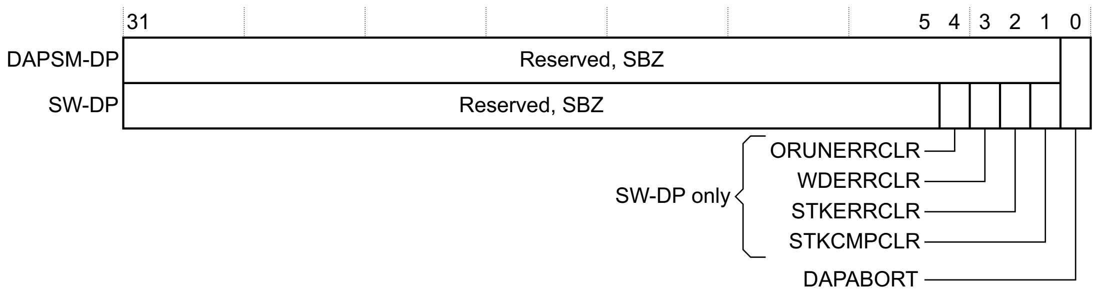


Figure 3-30 Abort Register bit assignments

Table 3-15 shows the Abort Register bit assignments.

Table 3-15 Abort Register bit assignments

Bits	Function	Description
[31:5]	-	Reserved, SBZ.
[4][a]	ORUNERRCLR[a]	Write b1 to this bit to clear the STICKYORUN overrun error flag[b].
[3][a]	WDERRCLR[a]	Write b1 to this bit to clear the WDATAERR write data error flag[b].
[2][a]	STKERRCLR[a]	Write b1 to this bit to clear the STICKYERR sticky error flag[b].
[1][a]	STKCMPCLR[a]	Write b1 to this bit to clear the STICKYCMP sticky compare flag[b].
[0]	DAPABORT	Write b1 to this bit to generate a DAP abort. This aborts the current access port transaction. This should only be done if the debugger has received WAIT responses over an extended period.

a. Implemented on SW-DP only. On a JTAG-DP this bit is Reserved, SBZ.
b. In the Control/Status register.

DP Aborts

Writing b1 to bit [0] of the Abort Register generates a debug port abort, causing the current AP transaction to abort. This also terminates the Transaction Counter, if it was active.

From a software perspective, this is a fatal operation. It discards any outstanding and pending transactions, and leaves the access port in an unknown state. However, on a SW-DP, the sticky error bits are not cleared.

You should use this function only in extreme cases, where debug host software has observed stalled target hardware for an extended period. Stalled target hardware is indicated by WAIT responses.

After a debug port abort is requested, new transactions can be accepted by the debug port. However, an access port access to the access port that was aborted can result in more WAIT responses. Other access ports can be accessed, however, the state of the system might make it impossible to continue with debug.

Caution

On a JTAG-DP, for the Abort Register:

- bit [0], DAPABORT, is the only bit that is defined
- the effect of writing any value other than `0x00000001` is Unpredictable.

Clearing error and sticky compare flags, SW-DP only

When a debugger, connected to a SW-DP, checks the Control/Status register and finds that an error flag is set, or that the sticky compare flag is set, it must write to the Abort register to clear the error or sticky compare flag. Table 3-15 on page 3-75 listed the flags that might be set in the Control/Status register, and shows which bit of the Abort register is used to clear each of the flags. You can use a single write of the Abort register to clear multiple flags, if this is necessary.

After clearing the flag, you might have to access the debug port and access port registers to find what caused the flag to be set. Typically:

- For the STICKYCMP or STICKYERR flag, you must find which location was accessed to cause the flag to be set.
- For the WDATAERR flag, after clearing the flag you resend the data that was corrupted.
- For the STICKYORUN flag, you must find which debug port or access port transaction caused the overflow. You then have to repeat your transactions from that point.

Identification Code Register, IDCODE

The Identification Code Register is always present on all debug port implementations. It provides identification information about the ARM Debug Interface.

JTAG-DP It is accessed using its own scan chain.

SW-DP It is at address 0b00 on read operations when the APnDP bit =1. Access to the Identification Code Register is not affected by the value of the CTRLSEL bit in the Select Register.

It is:

- a read-only register
- always accessible.

Figure 3-31 shows the Identification Code Register bit assignments.

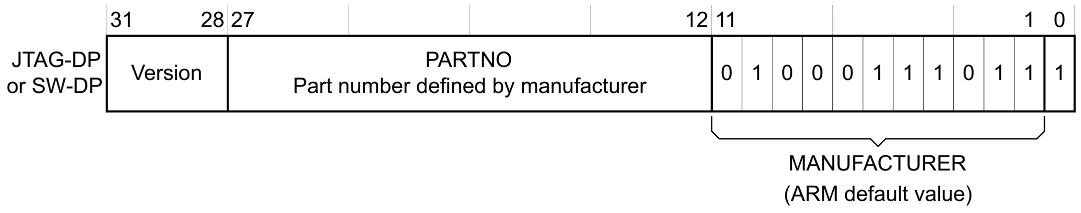


Figure 3-31 Identification Code Register bit assignments

Table 3-16 shows the Identification Code Register bit assignments.

Table 3-16 Identification Code Register bit assignments

Bits	Function	Description
[31:28]	Version	Version code: **JTAG-DP** `0x3` **SW-DP** `0x2`
[27:12]	PARTNO	Part Number for the debug port. This value is provided by the designer of the Debug Port and *must not* be changed. Current ARM-designed debug ports have the following PARTNO values: **JTAG-DP** `0xBA00` **SW-DP** `0xBA10`
[11:1]	MANUFACTURER	JEDEC Manufacturer ID, an 11-bit JEDEC code that identifies the manufacturer of the device. See *JEDEC Manufacturer ID*. The ARM default value for this field, shown in Figure 3-31 on page 3-76, is `0x43B`. Designers can change the value of this field. If the DAP is also used for boundary scan then this field *must* be set to the JEDEC Manufacturer ID assigned to the implementor.
[0]	-	Always 0b1.

JEDEC Manufacturer ID

This code is also described as the JEP-106 manufacturer identification code, and can be subdivided into two fields, as shown in Table 3-17.

Table 3-17 JEDEC JEP-106 manufacturer ID code, with ARM Limited values

JEP-106 field	Bits[a]	ARM Limited registered value
Continuation code	4 bits, [11:8]	b0100, `0x4`
Identity code	7 bits, [7:1]	b0111011, `0x3B`

a. Field width, in bits, and the corresponding bits in the Identification Code Register.

JDEC codes are assigned by the JEDEC Solid State Technology Association, see *JEP106M, Standard Manufacture's Identification Code*.

Control/Status Register, CTRL/STAT

The Control/Status Register is always present on all debug port implementations. It provides control of the debug port, and status information about the debug port.

JTAG-DP It is at address `0x4` when the *Instruction Register* (IR) contains DPACC.

SW-DP It is at address 0b01 on read and write operations when the APnDP bit =1 and the CTRLSEL bit in the Select Register is set to b0. For information about the CTRLSEL bit see *AP Select Register, SELECT* on page 3-81.

It is a read-write register, in which some bits have different access rights. It is Implementation-defined whether some fields in the register are supported, and Table 3-18 shows which fields are required in all implementations.

Figure 3-32 shows the Control/Status Register bit assignments.

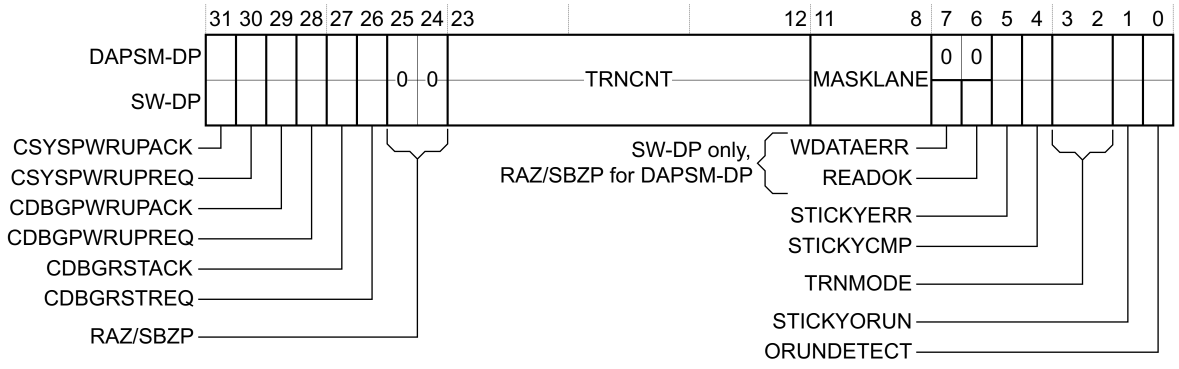


Figure 3-32 Control/Status Register bit assignments

Table 3-18 shows the Control/Status Register bit assignments.

Table 3-18 Control/Status Register bit assignments

Bits	Access	Function	Description	Required?
[31]	RO	CSYSPWRUPACK	System power-up acknowledge.	No
[30]	R/W	CSYSPWRUPREQ	System power-up request. After a reset this bit is LOW (0).	No
[29]	RO	CDBGPWRUPACK	Debug power-up acknowledge.	No
[28]	R/W	CDBGPWRUPREQ	Debug power-up request. After a reset this bit is LOW (0).	No
[27]	RO	CDBGRSTACK	Debug reset acknowledge.	Yes
[26]	R/W	CDBGRSTREQ	Debug reset request. After a reset this bit is LOW (0).	Yes
[25:24]	-	-	Reserved, RAZ/SBZP	

Table 3-18 Control/Status Register bit assignments (continued)

Bits	Access	Function	Description	Required?
[21:12]	R/W	TRNCNT	Transaction counter. After a reset the value of this field is Unpredictable.	Yes
[11:8]	R/W	MASKLANE	Indicates the bytes to be masked in pushed compare and pushed verify operations. See *MASKLANE and the bit masking of the pushed compare and pushed verify operations* on page 3-80. After a reset the value of this field is Unpredictable.	Yes
[7]	RO[a]	WDATAERR[a]	This bit is set to 1 if a Write Data Error occurs. It is set if: • there is a a parity or framing error on the data phase of a write • a write that has been accepted by the debug port is then discarded without being submitted to the access port. This bit can only be cleared by writing b1 to the WDERRCLR field of the Abort Register, see *Abort Register, ABORT* on page 3-74. After a power-on reset this bit is LOW (0).	Yes[a]
[6]	RO[a]	READOK[a]	This bit is set to 1 if the response to a previous access port or RDBUFF was OK. It is cleared to 0 if the response was not OK. This flag always indicates the response to the last access port read access. After a power-on reset this bit is LOW (0).	Yes[a]
[5]	RO[b]	STICKYERR	This bit is set to 1 if an error is returned by an access port transaction. To clear this bit: **On a JTAG-DP** Write b1 to this bit of this register. **On a SW-DP** Write b1 to the STKERRCLR field of the Abort Register, see *Abort Register, ABORT* on page 3-74. After a power-on reset this bit is LOW (0).	Yes
[4]	RO[b]	STICKYCMP	This bit is set to 1 when a match occurs on a pushed compare or a pushed verify operation. To clear this bit: **On a JTAG-DP** Write b1 to this bit of this register. **On a SW-DP** Write b1 to the STKCMPCLR field of the Abort Register, see *Abort Register, ABORT* on page 3-74. After a power-on reset this bit is LOW (0).	Yes

Table 3-18 Control/Status Register bit assignments (continued)

Bits	Access	Function	Description	Required?
[3:2]	R/W	TRNMODE	This field sets the transfer mode for access port operations, see *Transfer mode (TRNMODE), bits [3:2]* on page 3-81. After a power-on reset the value of this field is Unpredictable.	Yes
[1]	RO[b]	STICKYORUN	If overrun detection is enabled (see bit [0] of this register), this bit is set to 1 when an overrun occurs. To clear this bit: **On a JTAG-DP** Write b1 to this bit of this register. **On a SW-DP** Write b1 to the ORUNERRCLR field of the Abort Register, see *Abort Register, ABORT* on page 3-74. After a power-on reset this bit is LOW (0).	Yes
[0]	R/W	ORUNDETECT	This bit is set to b1 to enable overrun detection. After a reset this bit is Low (0).	Yes

a. Implemented on SW-DP only. On a JTAG-DP this bit is Reserved, RAZ/SBZP.
b. RO on SW-DP. On a JTAG-DP, this bit can be read normally, and writing b1 to this bit clears the bit to b0.

MASKLANE and the bit masking of the pushed compare and pushed verify operations

The MASKLANE field, bits [11:8] of the CTRL/STAT Register, is only relevant if the Transfer Mode is set to pushed verify or pushed compare operation, see *Transfer mode (TRNMODE), bits [3:2]* on page 3-81.

In the pushed operations, the word supplied in an access port write transaction is compared with the current value of the target access port address. The MASKLANE field lets you specify that the comparison is made using only certain bytes of the values. Each bit of the MASKLANE field corresponds to one byte of the access port values. Therefore, each bit is said to control one byte lane of the compare operation.

Table 3-19 shows how the bits of MASKLANE control the comparison masking.

Table 3-19 Control of pushed operation comparisons by MASKLANE

MASKLANE[a]	Meaning	Mask used for comparisons[b]
b1XXX	Include byte lane 3 in comparisons.	`0xFF------`
bX1XX	Include byte lane 2 in comparisons.	`0x--FF----`
bXX1X	Include byte lane 1 in comparisons.	`0x----FF--`
bXXX1	Include byte lane 0 in comparisons.	`0x------FF`

a. Bits [11:8] of the CTRL/STAT Register.
b. Bytes of the mask shown as -- are determined by the other bits of MASKLANE.

Transfer mode (TRNMODE), bits [3:2]

This field sets the transfer mode for access port operations. Table 3-20 lists the allowed values of this field, and their meanings.

Table 3-20 Transfer Mode bit definitions

TRNMODE[a]	AP Transfer mode
b00	Normal operation.
b01	Pushed verify operation.
b10	Pushed compare operation.
b11	Reserved.

a. Bits [3:2] of the CTRL/STAT Register.

In normal operation, access port transactions are passed to the access port for processing.

In pushed verify and pushed compare operations, the debug port compares the value supplied in the access port transaction with the value held in the target access port address.

AP Select Register, SELECT

The AP Select Register is always present on all debug port implementations. Its main purpose is to select the current Access Port (AP) and the active four-word register window in that access port. On a SW-DP, it also selects the Debug Port address bank.

JTAG-DP . It is at address `0x8` when the *Instruction Register* (IR) contains DPACC, and is a read/write register.

SW-DP . It is at address 0b10 on write operations when the APnDP bit =1, and is a write-only register. Access to the AP Select Register is not affected by the value of the CTRLSEL bit.

Figure 3-33 on page 3-82 shows the AP Select Register bit assignments.

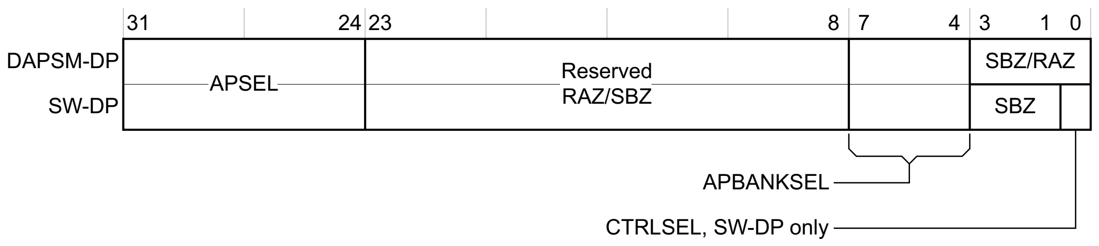


Figure 3-33 AP Select Register bit assignments

Table 3-21 shows the AP Select Register bit assignments.

Table 3-21 AP Select Register bit assignments

Bits	Function	Description
[31:24]	APSEL	Selects the current access port. The reset value of this field is Unpredictable.[a]
[23:8]	-	Reserved. SBZ/RAZ[a].
[7:4]	APBANKSEL	Selects the active four-word register window on the current access port. The reset value of this field is Unpredictable.[a]
[3:1]	-	Reserved. SBZ/RAZ[a].
[0][b]	CTRLSEL[b]	SW-DP Debug Port address bank select, see *CTRLSEL, SW-DP only* on page 3-83. After a reset this field is b0. However the register is WO and you cannot read this value.

a. On a SW-DP the register is write-only and therefore you cannot read the field value.
b. Implemented on SW-DP only. On a JTAG-DP this bit is Reserved, SBZ/RAZ.

If APSEL is set to a non-existent access port, all access port transactions return zero on reads and are ignored on writes.

——— **Note** ———

Every ARM Debug Interface implementation must include at least one access port.

CTRLSEL, SW-DP only

The CTRLSEL field, bit [0], controls which debug port register is selected at address b01 on a SW-DP. Table 3-22 shows the meaning of the different values of CTRLSEL.

Table 3-22 CTRLSEL field bit definitions

CTRLSEL[a]	**DP Register at address b01**
0	CTRL/STAT, see *Control/Status Register, CTRL/STAT* on page 3-77.
1	WCR, see *Wire Control Register, WCR (SW-DP only)* on page 3-84.

a. Bit [0] of the SELECT Register.

Read Buffer, RDBUFF

The 32-bit Read Buffer is always present on all debug port implementations. However, there are significant differences in its implementation on JTAG and SW Debug Ports.

JTAG-DP It is at address `0xC` when the *Instruction Register* (IR) contains DPACC, and is a Read-as-zero, Writes ignored (RAZ/WI) register.

SW-DP It is at address 0b11 on read operations when the APnDP bit =1, and is a read-only register. Access to the Read Buffer is not affected by the value of the CTRLSEL bit in the SELECT Register.

Read Buffer implementation and use on a JTAG-DP

On a JTAG-DP, the Read Buffer always reads as zero, and writes to the Read Buffer address are ignored.

The Read Buffer is architecturally defined to provide a debug port read operation that does not have any side effects. This means that a debugger can insert a debug port read of the Read Buffer at the end of a sequence of operations, to return the final Read Result and ACK values.

Read Buffer implementation and use on a SW-DP

On a SW-DP, performing a read of the Read Buffer captures data from the access port, presented as the result of a previous read, without initiating a new access port transaction. This means that reading the Read Buffer returns the result of the last access port read access, without generating a new AP access.

After you have read the Read Buffer, its contents are no longer valid. The result of a second read of the Read Buffer is Unpredictable.

If you require the value from an access port register read, that read must be followed by one of:

- A second access port register read. You can read the *Control/Status Register* (CSW) if you want to ensure that this second read has no side effects.
- A read of the DP Read Buffer.

This second access, to the access port or the debug port depending on which option you used, stalls until the result of the original access port read is available.

Wire Control Register, WCR (SW-DP only)

The Wire Control Register is always present on any SW-DP implementation. Its purpose is to select the operating mode of the physical serial port connection to the SW-DP.

It is a read/write register at address 0b01 on read and write operations when the CTRLSEL bit in the Select Register is set to b1. For information about the CTRLSEL bit see *AP Select Register, SELECT* on page 3-81.

Note

When the CTRLSEL bit is set to b1, to enable access to the WCR, the DP Control/Status Register is not accessible.

Many features of the Wire Control Register are implementation-defined.

Figure 3-34 shows the Wire Control Register bit assignments.

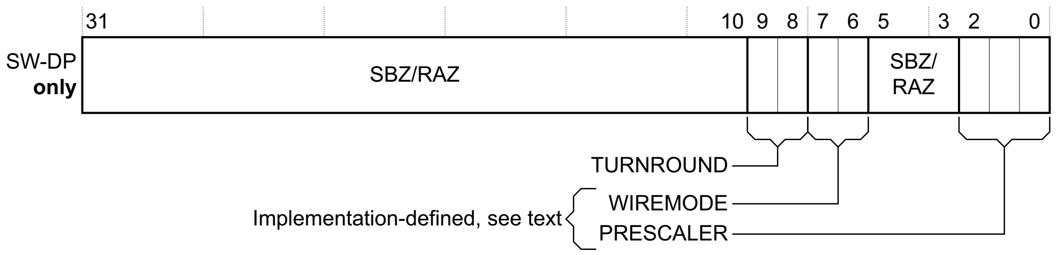


Figure 3-34 Wire Control Register bit assignments

Table 3-23 shows the Wire Control Register bit assignments.

Table 3-23 Wire Control Register bit assignments

Bits	Function	Description
[31:10]	-	Reserved. SBZ/RAZ.
[9:8]	TURNROUND	Turnaround tristate period, see *Turnaround tristate period, TURNROUND, bits [9:8]*. After a reset this field is b00.
[7:6]	WIREMODE	Identifies the operating mode for the wire connection to the debug port, see *Wire operating mode, WIREMODE, bits [7:6]*. After a reset this field is b01.
[5:3]	-	Reserved. SBZ/RAZ.
[2:0]	PRESCALER	Reserved. SBZ/RAZ.

Turnaround tristate period, TURNROUND, bits [9:8]

This field defines the turnaround tristate period. This turnaround period allows for pad delays when using a high sample clock frequency. Table 3-24 lists the allowed values of this field, and their meanings.

Table 3-24 Turnaround tristate period field bit definitions

TURNROUND[a]	Turnaround tri-state period
b00	1 sample period
b01	2 sample periods
b10	3 sample periods
b11	4 sample periods

a. Bits [9:8] of the WCR Register.

Wire operating mode, WIREMODE, bits [7:6]

This field identifies SW-DP as operating in Synchronous mode only.

This field is required, and Table 3-25 lists the allowed values of the field, and their meanings.

Table 3-25 Wire operating mode bit definitions

WIREMODE[a]	Wire operating mode
b00	Reserved
b01	Synchronous (no oversampling)
b1X	Reserved

a. Bits [7:6] of the WCR Register.

Read Resend Register, RESEND (SW-DP only)

The Read Resend Register is always present on any SW-DP implementation. Its purpose is to enable the read data to be recovered from a corrupted debugger transfer, without repeating the original AP transfer.

It is a 32-bit read-only register at address 0b10 on read operations. Access to the Read Resend Register is not affected by the value of the CTRLSEL bit in the SELECT Register.

Performing a read to the RESEND register does not capture new data from the access port. It returns the value that was returned by the last AP read or DP RDBUFF read.

Reading the RESEND register enables the read data to be recovered from a corrupted transfer without having to re-issue the original read request or generate a new DAP or system level access.

The RESEND register can be accessed multiple times. It always returns the same value until a new access is made to the DP RDBUFF register or to an access port register.

3.11 AHB-AP

This section describes the *AHB Access Port* (AHB-AP), for access to a system AHB bus through an AHB-Lite master. It acts as a slave to the DAP internal bus, driven by only a single debug port, JTAG-DP, at any one time. Figure 3-35 shows the internal structure of the AHB-AP.

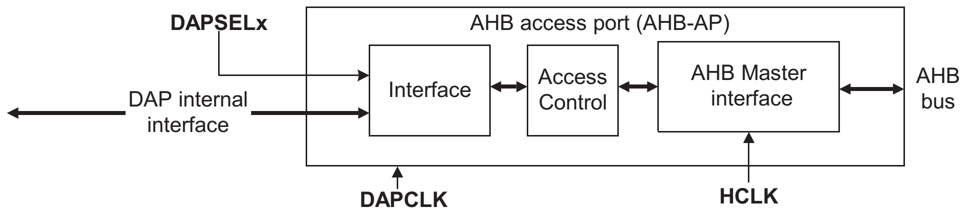


Figure 3-35 AHB access port internal structure.

The AHB-AP has two interfaces:

- An internal DAP bus interface
- AHB-Lite Master Port for access to system components.

3.11.1 AHB-Lite master ports

The AHB-Lite master port supports the following:

- AHB in AMBA v2.0
- ARM11 AMBA extensions
- TrustZone extensions.

The AHB-Lite master port does not support the following:

- BURST and SEQ
- Exclusive accesses
- Unaligned transfers.

Other signals

Table 3-26 shows the other AHB-AP ports.

Table 3-26 Other AHB-AP ports

Name	Type	Description
DBGEN	Input	Enables AHB-AP transfers if HIGH. Access to the AHB-AP registers is still permitted if **DBGEN** is LOW, but no AHB transfers are initiated. If a transfer is attempted when **DBGEN** is LOW, then the DAP bus returns **DAPSLVERR** HIGH.
SPIDEN	Input	Permits secure transfers to take place on the AHB-AP. If **SPIDEN** is HIGH, then **HPROT[6]** can be asserted as programmed into the SProt bit in the Control/Status Word Register. See *AHB-AP Control/Status Word Register, CSW, 0x00* on page 3-89.
nCDBGPWRDN	Input	Indicates that the debug infrastructure is powered down and enables clamping of signals driven onto the system AHB interface, and clamping of inputs to the debug domain.
nCSOCPWRDN	Input	Indicates that the system AHB interface is powered down. Ensures no transfers can be initiated and forces an error response to be generated in the access port. Also clamps inputs to the system domain.

3.11.2 AHB-AP programmer's model

This section describes the registers used to program the AHB-AP:

- *AHB-AP register summary*
- *AHB access port register descriptions* on page 3-89.

AHB-AP register summary

Table 3-27 shows the AHB access port registers.

Table 3-27 AHB access port registers

Offset	Type	Width	Reset value	Name
`0x00`	R/W	32	`0x40000002`	Control/Status Word, CSW
`0x04`	R/W	32	`0x00000000`	Transfer Address, TAR
`0x08`	-	-	-	Reserved SBZ
`0x0C`	R/W	32	-	Data Read/Write, DRW
`0x10`	R/W	32	-	Banked Data 0, BD0
`0x14`	R/W	32	-	Banked Data 1, BD1

Table 3-27 AHB access port registers (continued)

Offset	Type	Width	Reset value	Name
`0x18`	R/W	32	-	Banked Data 2, BD2
`0x1C`	R/W	32	-	Banked Data 3, BD3
`0x20-0xF7`	-	-	-	Reserved SBZ
`0xF8`	RO	32	Implementation defined	Debug ROM table
`0xFC`	RO	32	`0x24770001`	Identification Register, IDR. Required by all access ports

AHB access port register descriptions

The section describes the AHB access port registers:

- *AHB-AP Control/Status Word Register, CSW, 0x00*
- *AHB-AP Transfer Address Register, TAR, 0x04* on page 3-91
- *AHB-AP Data Read/Write Register, DRW, 0x0C* on page 3-92
- *AHB-AP Banked Data Registers, BD0-BD03, 0x10-Ox1C* on page 3-92
- *ROM Address Register, ROM, 0xF8* on page 3-92
- *AHB-AP Identification Register, IDR, 0xFC* on page 3-93.

AHB-AP Control/Status Word Register, CSW, 0x00

This is the control word used to configure and control transfers through the AHB interface.

Figure 3-36 shows the Control/Status Word Register bit assignments.

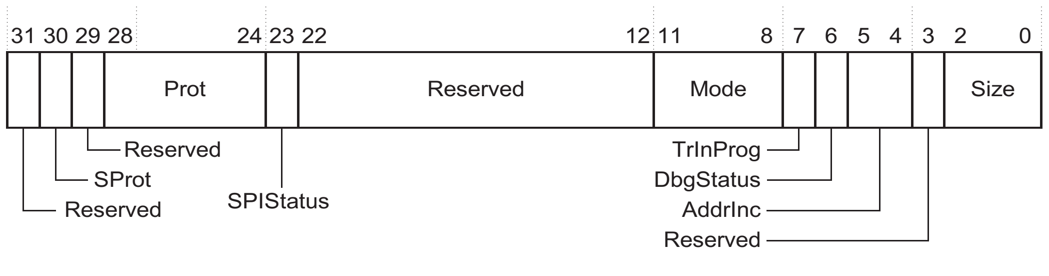


Figure 3-36 AHB-AP Control/Status Word Register bit assignments

Table 3-28 shows the bit assignments.

Table 3-28 AHB-AP Control/Status Word Register bit assignments

Bits	Type	Name	Function
[31]	-	-	Reserved SBZ.
[30]	R/W	SProt	Specifies that a secure transfer is requested. SProt HIGH indicates a non-secure transfer. SProt LOW indicates a secure transfer. • If this bit is LOW, and **SPIDEN** is HIGH, **HPROT[6]** is asserted LOW on an AHB transfer. • If this bit is LOW, and **SPIDEN** is LOW, **HPROT[6]** is asserted HIGH and the AHB transfer is not initiated. • If this bit is HIGH, the state of **SPIDEN** is ignored. **HPROT[6]** is HIGH. Reset value = b1. Non-secure.
[29]	-	-	Reserved SBZ.
[28:24]	R/W	Prot	Specifies the protection signal encoding to be output on **HPROT[4:0]**. Reset value: non-secure, non-exclusive, noncacheable, non-bufferable, data access, privileged = b00011.
[23]	RO	SPIStatus	Indicates the status of the SPIDEN port. If SPIStatus is LOW, no secure AHB transfers are carried out.
[22:12]	-	-	Reserved SBZ.

[11:8]	R/W	Mode	Specifies the mode of operation.
			b0000 = Normal download/upload model
			b0001-b1111 = Reserved SBZ.
			Reset value = b0000.

[7]	RO	TrInProg	Transfer in progress. This field indicates if a transfer is currently in progress on the AHB master port
[6]	RO	DbgStatus	Indicates the status of the DBGEN port. If DbgStatus is LOW, no AHB transfers are carried out. 1 = AHB transfers permitted. 0 = AHB transfers not permitted.

Table 3-28 AHB-AP Control/Status Word Register bit assignments (continued)

Bits	Type	Name	Function
[5:4]	R/W	AddrInc	Auto address increment and packing mode on Read or Write data access. Only increments if the current transaction completes without an Error Response. Does not increment if the transaction completes with an Error Response or the transaction is aborted. Auto address incrementing and packed transfers are not performed on access to Banked Data registers `0x10-0x1C`. The status of these bits is ignored in these cases. Increments and wraps within a 1KB address boundary, for example, for word incrementing from `0x1400-0x17FC`. If the start is at `0x14A0`, then the counter increments to `0x17FC`, wraps to `0x1400`, then continues incrementing to `0x149C`. b00 = Auto increment OFF. b01 = Increment, single. Single transfer from corresponding byte lane. b10 = Increment, packed Word = Same effect as single increment. Byte/Halfword: Packs four 8-bit transfers or two 16-bit transfers into a 32-bit DAP transfer. Multiple transactions are carried out on the AHB interface. b11 = Reserved SBZ, no transfer. Size of address increment is defined by the Size field, bits [2:0]. Reset value = b00.
[3]	-	-	Reserved SBZ, R/W = b0

[2:0]	R/W	Size	Size of the data access to perform:
			b000 = 8 bits
			b001 = 16 bits
			b010 = 32 bits
			b011-b111 = Reserved SBZ.
			Reset value = b010.

AHB-AP Transfer Address Register, TAR, 0x04

Table 3-29 shows the AHB-AP Transfer Address Register bit assignments.

Table 3-29 AHB-AP Transfer Address Register bit assignments

Bits	Type	Name	Function
[31:0]	R/W	Address	Address of the current transfer. Reset value is `0x00000000`.

AHB-AP Data Read/Write Register, DRW, 0x0C

Table 3-30 shows the AHB-AP Data Read/Write Register bit assignments.

Table 3-30 AHB-AP Data Read/Write Register bit assignments

Bits	Type	Name	Function
[31:0]	R/W	Data	Write mode: Data value to write for the current transfer. Read mode: Data value read from the current transfer.

AHB-AP Banked Data Registers, BD0-BD03, 0x10-Ox1C

BD0-BD3 provide a mechanism for directly mapping through DAP accesses to AHB transfers without having to rewrite the *Transfer Address Register* (TAR) within a four-location boundary. BD0 reads/writes from TA. BD1 reads/writes from TA+4. Table 3-31 shows the AHB-AP Banked Data Register bit assignments.

Table 3-31 Banked Data Register bit assignments

Bits	Type	Name	Function
[31:0]	R/W	Data	If **DAPADDR[7:4]** = `0x0001`, so accessing AHB-AP registers in the range `0x10-0x1C`, the derived **HADDR[31:0]** is: • Write mode: Data value to write for the current transfer to external address TAR[31:4] + **DAPADDR[3:2]** + 2'b00. • Read mode: Data value read from the current transfer from external address TAR[31:4] + **DAPADDR[3:2]** + 2'b00. Auto address incrementing is not performed on DAP accesses to BD0-BD3. Banked transfers are only supported for word transfers. Non-word banked transfers are reserved and unpredictable. Transfer size is currently ignored for banked transfers.

ROM Address Register, ROM, 0xF8

Table 3-32 shows the ROM Address Register bit assignments.

Table 3-32 ROM Address Register bit assignments

Bits	Type	Name	Function
[31:0]	RO	Debug AHB ROM Address	Base address of a ROM table. The ROM provides a look-up table for system components. Set to `0xFFFFFFFF` in the AHB-AP in the initial release.

AHB-AP Identification Register, IDR, 0xFC

Figure 3-37 shows the AHB-AP Identification Register bit assignments.

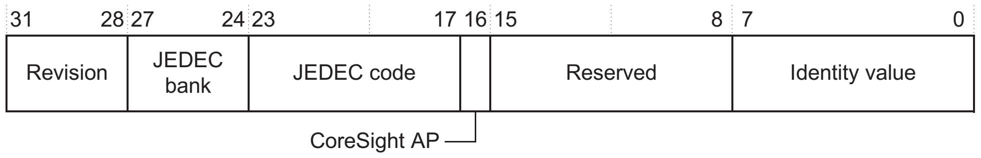


Figure 3-37 AHB-AP Identification Register bit assignments

Table 3-33 shows the AHB-AP Identification Register bit assignments.

Table 3-33 AHB-AP Identification Register bit assignments

Bits	Type	Name
[31:28]	RO	Revision. Reset value is `0x1` for AHB-AP.
[27:24]	RO	JEDEC bank[a].
[23:17]	RO	JEDEC code.
[16]	RO	CoreSight AP.
[15:8]	-	Reserved SBZ.
[7:0]	RO	Identity value. Reset value is `0x01` for AHB-AP.

a. Using JEDEC bank `0x0` with a JEDEC code of `0x00` is reserved for use by ARM Limited.

3.11.3 AHB-AP clocks and resets

The AHB-AP has two clock domains:

DAPCLK	Drives the DAP bus interface and access control for register read and writes. **DAPCLK** must be driven by a constant clock. When started, it must not be stopped or altered while the DAP is in use.
HCLK	AHB clock domain driving AHB interface.

DAPCLK and **HCLK** are asynchronous domains. Synchronizers are used between the Access Control block and the AHB master interface.

DAPRESETn initializes the state of all registers in the DAPCLK domain. **DAPRESETn** enables initialization of the DAP without affecting the normal operation of the SoC in which the DAP is integrated, and must be driven by the tools on external connection by accessing the appropriate location in the debug port.

HRESETn is used to reset the AHB interface and does not reset any state in the DAP interface.

3.11.4 Supported AHB protocol features

The AHB-Lite master port supports the following features:

- AHB in AMBA v2.0
- ARM11 AMBA extensions.

HPROT encodings

HPROT[6:0] is provided as an external port and is programmed from the Prot field in the CSW register with the following conditions:

- **HPROT[4:0]** programming is supported.
- **HPROT[5]** is not programmable and always set LOW. Exclusive access is not supported, and so **HRESP[2]** is not supported.
- **HPROT[6]** programming is supported. **HPROT[6]** HIGH denotes a nonsecure transfer. **HPROT[6]** LOW denotes a secure transfer. **HPROT[6]** can be asserted LOW by writing to the SProt field in the CSW Register. A secure transfer can only be initiated if **SPIDEN** is HIGH. If SProt is set LOW in the CSW Register to perform a secure transfer, but **SPIDEN** is LOW, then no AHB transfer takes place.

See *AHB-AP Control/Status Word Register, CSW, 0x00* on page 3-89 for values of the Prot field.

HRESP

HRESP[0] is the only RESPONSE signal required by the AHB-AP:

- AHB-Lite devices do not support SPLIT and RETRY and so **HRESP[1]** is not required. It is still provided as an input, and if not present on any slave it must be tied LOW. Any response that is not OKAY, that is, not 2'b00, is treated as an ERROR response.
- **HRESP[2]** is not required because exclusive accesses are unsupported in the AHB-AP.

HBSTRB support

HBSTRB[3:0] signals are automatically generated based on the transfer size **HSIZE[2:0]** and **HADDR[1:0]**. Byte, halfword and word transfers are supported. It is not possible for the user to directly control **HBSTRB[3:0]**.

Unaligned transfers are not supported.

Table 3-34 shows an example of the generated **HBSTRB[3:0]** signals for different-sized transfers.

Table 3-34 Example generation of byte lane strobes

Transfer description	HADDR[1:0]	HSIZE[2:0]	HBSTRB[3:0]
8-bit access to `0x1000`	b00	b000	b0001
8-bit access to `0x1003`	b11	b000	b1000
16-bit access to `0x1002`	b10	b001	b1100
32-bit access to `0x1004`	b00	b010	b1111

AHB-AP transfer types and bursts

The AHB-AP cannot initiate a new AHB transfer every clock cycle (unpacked) because of the additional cycles required to serial scan in the new address or data value through a debug port. The AHB-AP supports two **HTRANS** transfer types, IDLE and NONSEQ.

- When a transfer is in progress, it is of type NONSEQ.
- When no transfer is in progress and the AHB-AP is still granted the bus then the transfer is of type IDLE.

The only unpacked **HBURST** encoding supported is SINGLE. Packed 8-bit transfers or 16-bit transfers are treated as individual NONSEQ, SINGLE transfers at the AHB-Lite interface. This ensures that there are no issues with boundary wrapping, to avoid additional AHB-AP complexity.

A full AHB master interface can be created by adding an AHB-Lite to AHB wrapper to the output of the AHB-AP, as provided in the AMBA Design Kit.

3.11.5 Packed transfers

The DAP internal interface is a 32-bit data bus. 8-bit or 16-bit transfers can be formed on AHB according to the Size field in the Control/Status Word Register at `0x00`. The AddrInc field in the Control/Status Word Register enables optimized use of the DAP internal bus to reduce the number of accesses from the tools to the DAP. It indicates if the entire data word is to be used to pack more than one transfer. Address incrementing is automatically enabled if packet transfers are initiated so that multiple transfers are carried out at the sequential addresses. The size of the address increment is based on the size of the transfer.

See *AHB-AP Control/Status Word Register, CSW, 0x00* on page 3-89 for values of the AddrInc field and *AHB-AP Data Read/Write Register, DRW, 0x0C* on page 3-92 for Data Read/Write Register bit values.

An example of the transactions are:

- For an unpacked 16-bit write to an address base of `0x2` (CSW[2:0]=b001, CSW[5:4]=b01), **HWDATA[31:16]** is written from bits [31:16] in the Data Read/Write Register.
- For an unpacked 8-bit read to an address base of `0x1`, (CSW[2:0]=b000, CSW[5:4]=b01), **HRDATA[31:16]** and **HRDATA[7:0]** are zeroed, and **HRDATA[15:8]** contains read data.
- For a packed byte write at a base address `0x2`, (CSW[2:0]=b000, CSW[5:4]=b10), four write transfers are initiated, the order of data being sent is:
  - **HWDATA[23:16]**, from DRW[23:16], to **HADDR[31:0]**=`0x2`
  - **HWDATA[31:24]**, from DRW[31:24], to **HADDR[31:0]**=`0x3`
  - **HWDATA[7:0]**, from DRW[7:0], to **HADDR[31:0]**=`0x4`
  - **HWDATA[15:8]**, from DRW[15:8], to **HADDR[31:0]**=`0x5`
- For a packed halfword reading at a base address of `0x2`, (CSW[2:0]=b001, CSW[5:4]=b10), two read transfers are initiated:
  - **HRDATA[31:16]** is stored into DRW[31:16] from **HADDR[31:0]**=`0x2`
  - **HRDATA[15:0]** is stored into DRW[15:0] from **HADDR[31:0]**=`0x4`

The AHB-AP only asserts **DAPREADY** HIGH when all packed transfers from the AHB interface have completed.

If the current transfer is aborted or the current transfer receives an ERROR response, the AHB-AP does not complete the following packed transfers and returns **DAPREADY** HIGH immediately after the current packed transfer.

3.11.6 Version support

If the additional functionality provided by the ARM11 AMBA extensions or TrustZone extensions are not required then the output signals to support the extra functionality must be left unconnected.

3.12 DAP transfers

This section describes:

- *DAP transfer aborts*
- *Error response generation*.

3.12.1 DAP transfer aborts

The AHB-AP does not cancel the system-facing operation and returns **DAPREADY** HIGH one cycle after **DAPABORT** has been asserted by the driving debug port. The externally driving AHB master port does not violate the AHB protocol. After a transfer has been aborted, the Control/Status Word Register can be read to determine the state of the transfer in progress bit, TrInProg. When TrInProg returns to zero, either because the external transfer completes, or because of a reset, the AHB-AP returns to normal operation. All other writes to the AHB-AP are ignored until this bit is returned LOW after a Transfer Abort.

See *DAP transfer aborting* on page 3-132 for a detailed description of Transfer Abort.

3.12.2 Error response generation

This section describes:

- *System initiated error response*
- *Access port initiated error response*.

System initiated error response

An error response received on the system driving master propagates onto the DAP bus when the transfer is completed. This response is received by the debug ports.

Access port initiated error response

Access port initiated error responses are:

- *AHB-AP reads after an abort*
- *AHB-AP writes after an abort* on page 3-98.

AHB-AP reads after an abort

After a **DAPABORT** operation has been carried out, and an external transfer is still pending, that is, the TrInProg bit is still HIGH, reads of all registers return a normal response except for reads of the Data Read/Write Register and banked registers which cannot initiate a new system read transfer. Reads of the Data Read/Write Register and banked registers return an error response until the TrInProg bit in the Control/Status Word Register is cleared because of the system transfer completing, or because of a reset.

AHB-AP writes after an abort

After a **DAPABORT** operation has been carried out, and an external transfer is still pending, that is, the transfer in progress bit is still HIGH, all writes to the access port return an error response, because they are ignored until the TrInProg bit is cleared.

3.12.3 Differentiation between system and access port initiated error responses

If **DAPSLVERR** is HIGH and TrInProg is LOW in the Control/Status Word Register, the error is from either:

- a system error response if **DBGEN** and **SPIDEN** permit the transfer to be initiated
- an AHB-AP error response if **DBGEN** and **SPIDEN** do not permit the transfer to be initiated.

Table 3-35 shows the options.

Table 3-35 Error responses with DAPSLVERR HIGH and TrInProg LOW

SProt	SPIDEN	DBGEN	Error response from	Reason
x	x	0	AHB-AP	No transfers permitted
0	0	1	AHB-AP	Secure transfers not permitted
0	1	1	System	Secure transfer produced an error response
1	x	1	System	Non secure transfer produced an error response

If **DAPSLVERR** is HIGH and TrInProg is HIGH, then the error is from an access port error response. The transfer has not been accepted by the access port. This case can only occur after an abort has been initiated and the system transfer has not completed.

3.12.4 Effects of resets

The **HRESETn** signal can be asserted LOW during any time. The DAP interface must return **DAPSLVERR** for the current transaction.

The **DAPRESETn** signal shall only be asserted LOW when there is no pending transaction on the AHB interface.

3.12.5 Effects of power down signals

The **nCSOCPWRDN** signal can be asserted LOW at any time. The DAP interface must return **DAPSLVERR** for the current transaction. The **nCDBGPWRDN** signal must only be asserted LOW when there is no pending transaction on the AHB interface.

3.13 AHB-AP dual power domain support

If the AHB-AP is split across multiple power domains, with the **DAPCLK** driven side in the Debug domain, and the AHB master port in the SoC power domain, clamping logic must be instantiated on the outputs of signals crossing each power down domain. See the *CoreSight Implementation and Integration Manual* for more information.

3.13.1 RTL hierarchy

Figure 3-38 shows the RTL structure to support power domain separation.

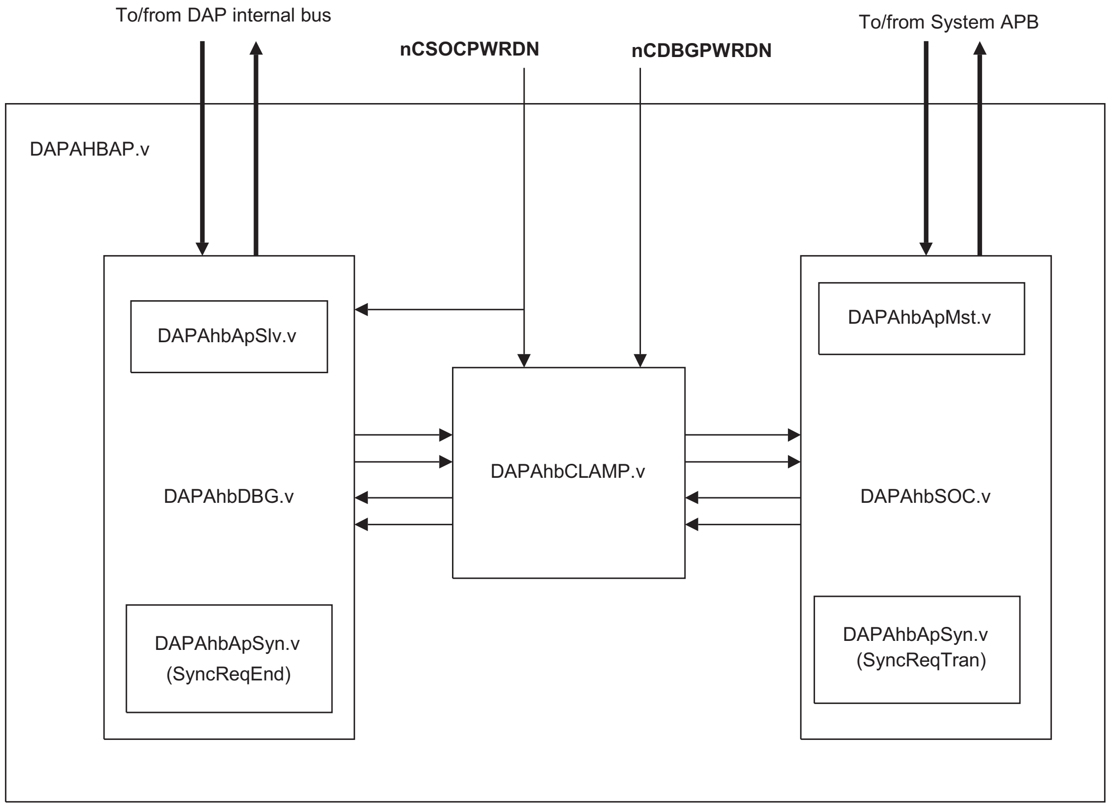


Figure 3-38 AHB-AP signal clamping

3.14 APB-AP

The *APB Access Port* (APB-AP) provides an APB master supporting the APB AMBA v3.0 protocol for access to an APB bus. Figure 3-39 shows the functional blocks of the APB-AP.

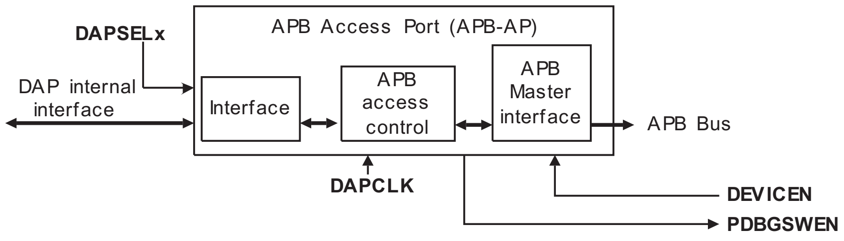


Figure 3-39 APB-AP functional blocks

3.14.1 APB-AP port definitions

The internal DAP signals are described in *Internal DAP bus signals* on page 3-11.

The APB master port supports the APB in AMBAv3.0 extensions for transfer extension using **PREADY** and slave error response detection using **PSLVERR**.

Table 3-36 shows the other APB-AP ports.

Table 3-36 APB-AP other ports

Name	Type	Description
PDBGSWEN	Output	Enable software access to the Debug APB at the APB multiplexor
DEVICEEN	Input	Disable device when LOW

3.14.2 APB-AP programmer's model

Table 3-37 shows the APB-AP registers.

Table 3-37 APB-AP registers

Offset	Type	Width	Reset value	Name
`0x00`	R/W	32	`0x00000002`	Control/Status Word, CSW
`0x04`	R/W	32	`0x00000000`	Transfer Address, TAR
`0x08`	-	-	-	Reserved SBZ

Table 3-37 APB-AP registers (continued)

Offset	Type	Width	Reset value	Name
`0x0C`	R/W	32	-	Data Read/Write, DRW
`0x10`	R/W	32	-	Banked Data 0, BD0
`0x14`	R/W	32	-	Banked Data 1, BD1
`0x18`	R/W	32	-	Banked Data 2, BD2
`0x1C`	R/W	32	-	Banked Data 3, BD3
`0x20-0xF4`	-	-	-	Reserved SBZ
`0xF8`	RO	32	Implementation-defined.	Debug ROM Address, ROM
`0xFC`	RO	32	`0x04770002`	Identification Register, IDR

The APB-AP registers are described in:

- *APB-AP Control/Status Word Register, CSW, 0x00*
- *APB-AP Transfer Address Register, TAR, 0x04* on page 3-103
- *APB-AP Data Read/Write Register, DRW, 0x0C* on page 3-104
- *APB-AP Banked Data Registers, BD0-BD3, 0x10-0x1C* on page 3-104
- *Debug APB ROM Address, ROM, 0xF8* on page 3-104
- *APB-AP Identification Register* on page 3-105.

3.14.3 APB-AP Control/Status Word Register, CSW, 0x00

The APB-AP control word is used to configure and control transfers through the APB interface. Figure 3-40 shows the bit assignments.

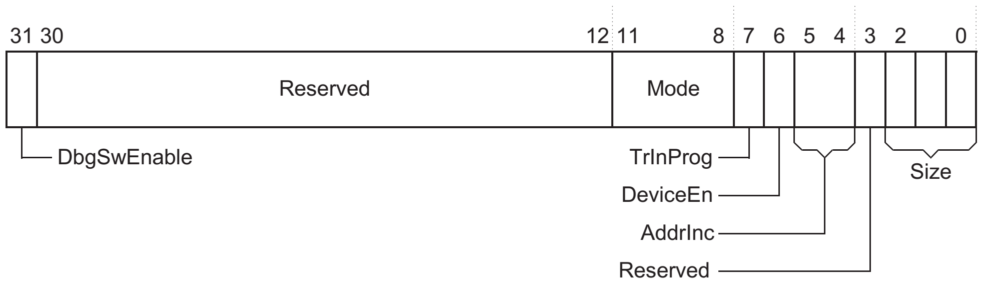


Figure 3-40 APB-AP Control/Status Word Register bit assignments

Table 3-38 shows the bit assignments.

Table 3-38 APB Control/Status Word Register bit assignments

Bits	Type	Name	Function
[31]	R/W	DbgSwEnable	Software access enable. Drives **PDBGSWEN** to enable or disable software access to the Debug APB bus in the APB multiplexor. b1 = Enable software access b0 = Disable software access. Reset value = b0. On exit from reset, defaults to b1 to enable software access.
[30:12]	-	-	Reserved SBZ.

[11:8]	R/W	Mode	Specifies the mode of operation.
			b0000 = Normal download/upload model
			b0001-b1111 = Reserved SBZ.
			Reset value = b0000.

[7]	RO	TrInProg	Transfer in progress. This field indicates if a transfer is currently in progress on the APB master port.
[6]	RO	DeviceEn	Indicates the status of the **DEVICEEN** input. • If APB-AP is connected to the Debug APB, that is, a bus connected only to debug and trace components, it must be permanently enabled by tying **DEVICEEN** HIGH. This ensures that trace components can still be programmed when **DBGEN** is LOW. In practice, it is expected that the APB-AP is almost always used in this way. • If APB-AP is connected to a system APB dedicated to the non-secure world, **DEVICEEN** must be connected to **DBGEN**. • If APB-AP is connected to a system APB dedicated to the secure world, **DEVICEEN** must be connected to **SPIDEN**.
[5:4]	R/W	AddrInc	Auto address increment and packing mode on Read or Write data access. Does not increment if the transaction completes with an error response or the transaction is aborted. Auto address incrementing is not performed on access to banked data registers `0x10-0x1C`. The status of these bits is ignored in these cases. b11 = Reserved b10 = Reserved b01 = Increment b00 = Auto increment OFF. Increment occurs in word steps. Reset value = b00.

Table 3-38 APB Control/Status Word Register bit assignments (continued)

Bits	Type	Name	Function
[3]	-	-	Reserved SBZ.
[2:0]	RO	Size	Size of the access to perform. Fixed at b010 = 32 bits. Reset value = b010.

3.14.4 APB-AP Transfer Address Register, TAR, 0x04

The Transfer Address Register holds the address of the current transfer. Figure 3-41 shows the bit assignments.

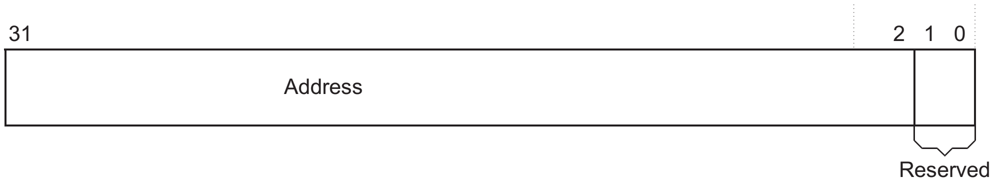


Figure 3-41 APB-AP Transfer Address Register bit assignments

Writes to the Transfer Address Register from the DAP interface write to bits [31:2] only. Bits [1:0] of **DAPWDATA** are ignored on writes to the Transfer Address Register. Table 3-39 shows the bit assignments.

Table 3-39 APB-AP Transfer Address Register bit assignments

Bits	Type	Name	Function
[31:2]	R/W	Address[31:2]	Address[31:2] of the current transfer. **PADDR[31:2]**=TAR[31:2] for accesses from Data Read/Write Register at `0x0C`. **PADDR[31:2]**=TAR[31:4]+**DAPADDR[3:2]** for accesses from Banked Data Registers at `0x10`-`0x1C` and `0x0C`.
[1:0]	-	Reserved SBZ	Set to 2'b00. SBZ/RAZ.

3.14.5 APB-AP Data Read/Write Register, DRW, 0x0C

Table 3-40 shows the bit assignments of the APB-AP Data Read/Write Register.

Table 3-40 ABP-AP Data Read/Write Register bit assignments

Bits	Type	Name	Function
[31:0]	R/W	Data	Write mode: Data value to write for the current transfer. Read mode: Data value read from the current transfer.

3.14.6 APB-AP Banked Data Registers, BD0-BD3, 0x10-0x1C

BD0-BD3 provide a mechanism for directly mapping through DAP accesses to APB transfers without having to rewrite the Transfer Address Register within a four word boundary, that is, BD0 reads/write from TAR, BD1 from TAR+4, and so on.

Table 3-41 shows the bit assignments.

Table 3-41 APB-AP Banked Data Registers bit assignments

Bits	Type	Name	Function
[31:0]	R/W	Data	If **DAPADDR[7:4]** = `0x0001`, so accessing APB-AP registers in the range `0x10-0x1C`, then the derived **PADDR[31:0]** is: • Write mode: Data value to write for the current transfer to external address TAR[31:4] + **DAPADDR[3:2]** + 2'b00. • Read mode: Data value read from the current transfer from external address TAR[31:4] + **DAPADDR[3:2]** + 2'b00. Auto address incrementing is not performed on DAP accesses to BD0-BD3. Reset value = `0x00000000`

3.14.7 Debug APB ROM Address, ROM, 0xF8

A ROM table must be present in all CoreSight systems. See *ROM table registers* on page 2-23 for more information. Figure 3-42 shows the bit assignments.

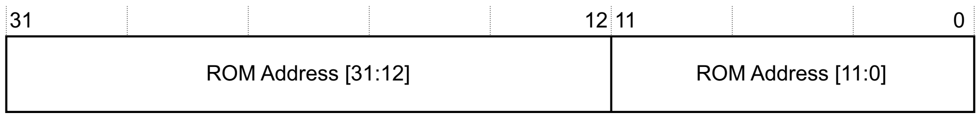


Figure 3-42 Debug APB ROM Address Register bit assignments

Table 3-42 shows the bit assignments.

Table 3-42 Debug APB ROM Address Register bit assignments

Bits	Type	Name	Function
[31:12]	RO	ROM Address [31:12]	Base address of the ROM table. The ROM provides a look-up table of all CoreSight Debug APB components. Read only. Set to `0xFFFFF` if no ROM is present. In the initial CoreSight release this must be set to `0x80000`.
[11:0]	RO	ROM Address [11:0]	Set to `0x000` if ROM is present. Set to `0xFFF` if ROM table is not present. In the initial CoreSight release this must be set to `0x000`.

3.14.8 APB-AP Identification Register

Figure 3-43 shows the bit assignments.

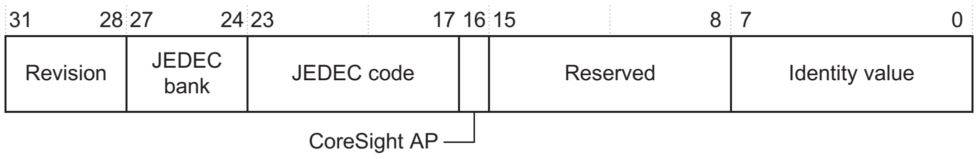


Figure 3-43 APB-AP Identification Register bit assignments

Table 3-43 shows the bit assignments.

Table 3-43 APB-AP Identification Register bit assignments

Bits	Type	Name
[31:28]	RO	Revision. Reset value is `0x0` for APB-AP.
[27:24]	RO	JEDEC bank[a]. `0x4` indicates ARM Limited.
[23:17]	RO	JEDEC code. `0x38` indicates ARM Limited.
[16]	RO	Memory AP. `0x0` indicates a standard register map is used.
[15:8]	-	Reserved SBZ.
[7:0]	RO	Identity value. Reset value is `0x02` for APB-AP.

a. Using JEDEC bank `0x0` with a JEDEC code of `0x00` is reserved for use by ARM Limited.

3.14.9 APB-AP clocks and resets

The APB-AP supports a synchronous APB interface. The internal DAP interface and the APB interface operate from **DAPCLK**.

The APB-AP has one clock domain, **DAPCLK**. It drives the complete APB-AP. This must be connected to **PCLKDBG** for the APB interface.

DAPRESETn resets the internal DAP interface and the APB interface.

3.14.10 APB-AP transfers

The APB-AP is fully compliant with the APB AMBAv3.0 protocol supporting:

- extended slave transfers using **PREADY**
- transfer responses using **PSLVERR**.

3.14.11 Effects of DAPABORT

The APB-AP does not cancel the system-facing operation and returns **DAPREADY** HIGH one cycle after **DAPABORT** has been asserted by the debug port. The externally driving APB master port does not violate the APB protocol. After a transfer has been aborted, the Control and Status Register can be read to determine the state of the transfer in progress bit, TrInProg. When TrInProg returns to zero, either because of the external transfer completing or a reset, the APB-AP returns to normal operation. All other writes to the APB-AP are ignored until this bit is returned LOW after a Transfer Abort.

See *DAP transfer aborting* on page 3-132 for a detailed description of Transfer Abort.

3.14.12 APB-AP error response generation

This section describes error response generation in the following sections:

- *System initiated error response*
- *AP-initiated error response* on page 3-107
- *Differentiation between System-initiated and AP-initiated error responses* on page 3-107.

System initiated error response

An error response received on the APB master interface propagates onto the DAP bus as the transfer is completed. This is received by the debug ports.

AP-initiated error response

- APB-AP reads after an abort:

  After a Transfer Abort operation has been carried out, and an external transfer is still pending, that is, the TrInProg bit in the Control/Status Word Register is still HIGH, reads of all registers return a normal response except for reads of the data registers Data Read/Write Register and banked registers which cannot initiate a new system read transfer. Reads of the Data Read/Write Register and banked registers return an error response until the TrInProg bit is cleared because of the system transfer completing, or a reset.

- APB-AP writes after an abort:

  After a Transfer Abort operation has been carried out, and an external transfer is still pending, that is, the transfer in progress bit is still HIGH, all writes to the access port return an error response, because they are ignored until the TrInProg bit has cleared.

Differentiation between System-initiated and AP-initiated error responses

If **DAPSLVERR** is HIGH and TrInProg is LOW in the Control/Status Word Register then the error is from a system error response.

If **DAPSLVERR** is HIGH and TrInProg is HIGH, then the error is from an access port error response. The transfer has not been accepted by the access port. This case can only occur after an abort has been initiated and the system transfer has not completed.

3.15 APB-Mux

The *APB Multiplexor* (APB-Mux) for the DAP enables external tools and system access to the CoreSight Debug APB. The APB-Mux encapsulates the multiple interfaces into a single deliverable component, enabling multi-master access to the Debug APB.

Figure 3-44 shows the APB-Mux. External tool connection to the APB-Mux uses the *APB Access Port* (APB-AP) to provide an APB master interface. System access requires an APB bridge to provide the APB master interface.

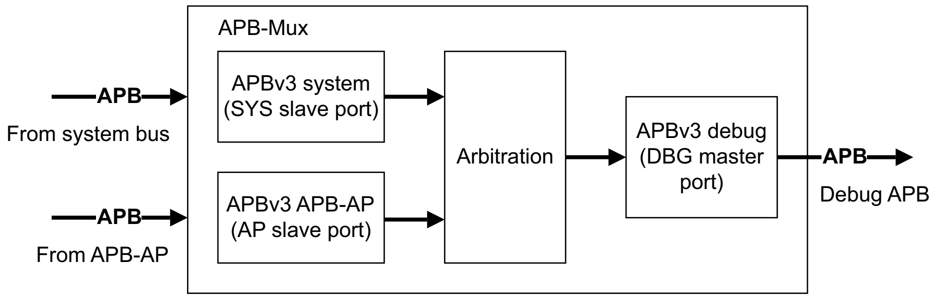

APB-Mux
APB
From system bus
APB
From APB-AP
APBv3 system (SYS slave port)
APBv3 APB-AP (AP slave port)
Arbitration
APBv3 debug (DBG master port)
APB
Debug APB

Figure 3-44 APB-Mux block diagram

Figure 3-45 on page 3-109 shows the APB-Mux integrated into the DAP. The APB-AP Slave port is connected to the APB-AP and the System Slave port to the system bus. The system bus requires an APB bridge to connect to the APB-Mux. APB-AP and system connections must be made in the order shown in Figure 3-44 to support distinct debug and system power domains.

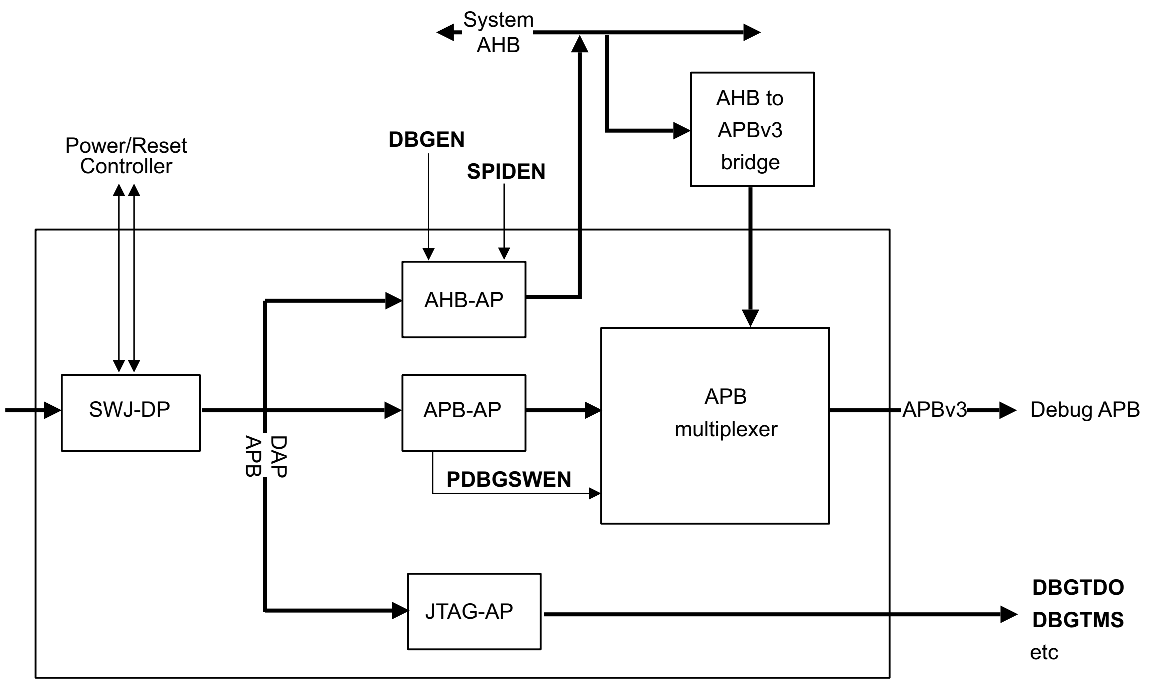


Figure 3-45 APB-Mux integrated into the DAP

3.15.1 APB-Mux port definitions

The APB-Mux has the following ports:

- an APB-AP slave port, signals suffixed with AP
- a system slave port, signals suffixed with SYS
- a Debug APB master port, signals suffixed with DBG.

Additional APB-Mux signals are described in *APB-Mux miscellaneous signals* on page 3-110.

3.15.2 APB-Mux miscellaneous signals

Table 3-44 shows the APB-Mux miscellaneous signals.

Table 3-44 APB-Mux miscellaneous signals

Name	Type	Description
nCDBGPWRDN	Input	Indicates that the debug infrastructure is powered down. Any system accesses to the debug APB return an error response. Also enables clamping of signals driven onto the system interface, and clamping of inputs to the debug domain.
nCSOCPWRDN	Input	Indicates that the system APB slave interface is powered down and enables clamping of signals driven into the APB Mux. Also clamps inputs to the system domain.
PDBGSWEN	Input	Enables software access to the Debug APB.

3.15.3 APB-Mux arbitration and APB Mux connectivity

This section describes APB-Mux arbitration and connectivity:

- *APB-Mux arbitration*
- *APB-Mux connectivity*.

APB-Mux arbitration

The APB-Mux uses a fixed arbitration scheme to support the two slave interfaces. The arbitration logic ensures that only one APB bus master, either the APB-AP, or system bus master has access to the CoreSight Debug APB at any one time.

The APB-AP Slave port always has priority over the System Slave port. If two transfers are initiated at the same time, the transfer from the APB-AP Slave port is always propagated to the master port output for Debug APB access. The System Slave port is only granted access if the APB-AP Slave port is not requesting an access, that is, **PSELAP** is LOW. It is therefore possible for the APB-AP Slave port to maintain back-to-back transfers on the Debug APB without allowing access to the System Slave port. When a transaction is in progress on one slave port and the other port initiates a transaction, **PREADY** is held LOW for the newly requested transaction until the APB-Mux arbiter grants access to the other slave port.

The APB-Mux provides no address decoding, nor default response generation. This must be implemented by an external address decoder incorporating a default slave.

APB-Mux connectivity

The connection rules are:

- AP-AP Slave port connects to the APB-AP
- System Slave port connects to the system bus.

3.15.4 APB-Mux Software Access Enable

The APB-Mux receives **PDBGSWEN** from the APB-AP to enable software access from the System Slave Port to the Debug APB.

PDBGSWEN LOW

System access to the Debug APB is not permitted. The System Slave port must return **PSLVERRSYS** HIGH on any access. No APB-Mux master transfer is initiated.

PDBGSWEN HIGH

System access to the Debug APB is permitted.

3.15.5 APB-Mux clocks, power, and resets

The APB-Mux has two clocks:

PCLKDBG	Drives all logic, except for the System Slave port interface.
PCLKSYS	Drives the System Slave port interface.

The APB-Mux has an asynchronous interface between the System Slave port and the rest of the APB-Mux, as shown in Figure 3-46. The asynchronous interface defines a common boundary between:

- debug and system clocks domains, **PCLKDBG** and **PCLKSYS**
- debug and system reset domains, **PRESETDBGn** and **PRESETSYSn**
- debug and system power domains.

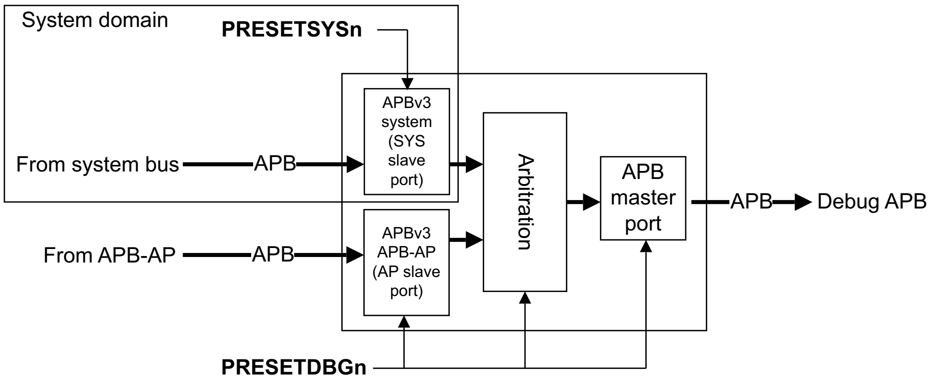


Figure 3-46 APB-Mux domains

Effects of power down

The debug and system power domains can be independently powered up and down. Accesses from tools to the debug APB through the APB-Mux can only be performed when the Debug APB is powered up, therefore this access mechanism does not cross asynchronous domains. **PSLVERRAP** is only returned HIGH if a Debug APB component has driven this signal HIGH. A system access to the Debug APB, when the Debug APB is powered down, must return **PSLVERRSYS** HIGH to indicate that a transaction is unsuccessful. **nCDBGPWRDN** enables the APB Mux to detect if the debug domain is powered down.

Effects of resets

The debug and system domains can be independently reset. A reset initiated from either domain must not cause a protocol violation in the other domain.

- If the Debug APB is reset during a system level write access to the debug infrastructure, the System Slave port must return **PSLVERRSYS** HIGH. The write operation is not performed.
- If the Debug APB is reset during a system level read access to the debug infrastructure, the System Slave port must return **PSLVERRSYS** HIGH. The read data is undefined.
- If the System APB is reset during a Debug APB access from the APB-AP, this must not invalidate the existing transfer already in progress from the APB-AP Slave port on the APB-Mux.
- If the System APB is reset during a system level access to the debug infrastructure then the APB-Mux must hold the existing transfer values to complete the Debug APB transaction without violating the APB protocol. A system write transfer to Debug APB, if already initiated, must complete. A system read transfer to Debug APB, if already initiated, must complete up to the APB-Mux master interface.

Output clamping

If the APB-Mux is split across multiple power domains, with the **PCLKDBG** driven side in the Debug domain, and the system slave port in the SoC power domain, clamping logic must be instantiated on the outputs of signals crossing each power down domain.

Figure 3-47 on page 3-113 shows the RTL structure to support power domain separation.

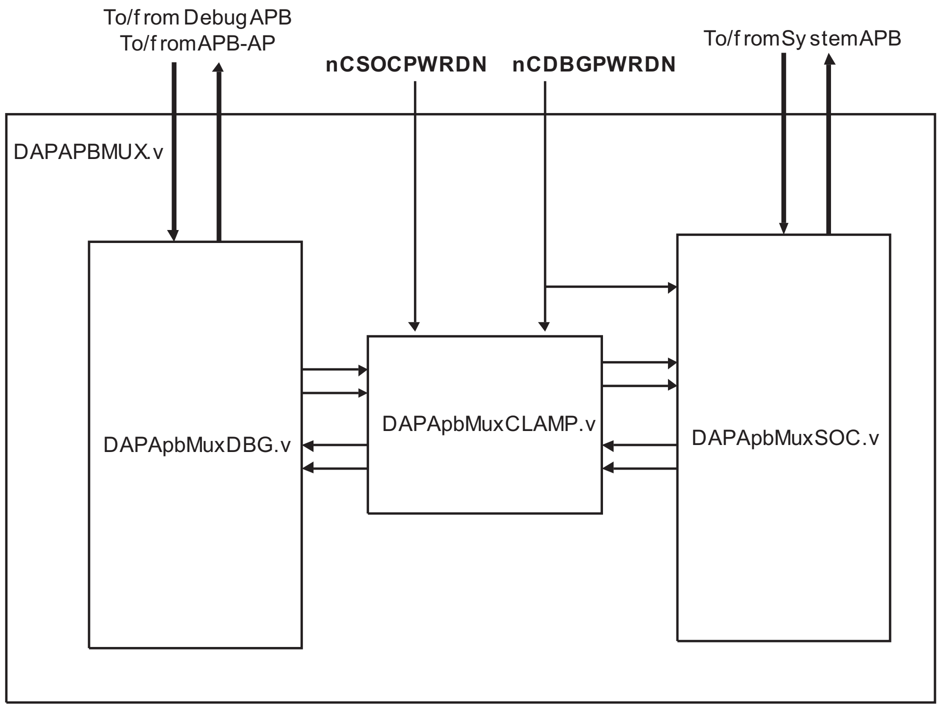


Figure 3-47 APB-Mux power domain separation

3.16 JTAG-AP

The *JTAG Access Port* (JTAG-AP) provides JTAG access to on-chip components, operating as a JTAG master port to drive JTAG chains throughout a SoC. The JTAG command protocol is byte-orientated, with a word wrapper on the read and write ports so as to yield acceptable performance from the DAP internal bus which has a 32-bit data path only. Daisy chaining is avoided by using a port multiplexor. In this way, slower cores do not impede faster cores.

The JTAG-AP is described in the following sections

- *About JTAG-AP*
- *Internal DAP bus interface signals* on page 3-115
- *JTAG to slave device signals* on page 3-116
- *JTAG-AP programmer's model* on page 3-116
- *Global port and RTCK* on page 3-120
- *RTCK wrapper* on page 3-121
- *JTAG-AP byte command protocol* on page 3-122
- *Transfer aborts* on page 3-124
- *BFIFOn interface examples* on page 3-124
- *JTAG-AP clock domains* on page 3-126
- *ROM table registers* on page 3-127
- *ROM table entries* on page 3-128.

3.16.1 About JTAG-AP

Figure 3-48 shows that JTAG-AP is a bridge between an on-chip DAP internal bus and a number of JTAG devices and JTAG chains.

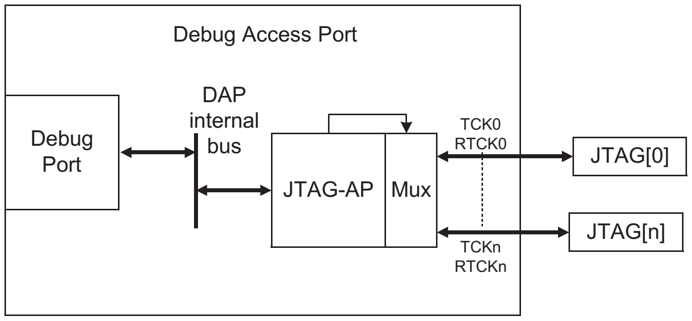


Figure 3-48 JTAG-AP with multiplexor

JTAG-AP supports multiple cores through a multiplexor wrapper. It is recommended that only one core is connected to each multiplexor port. A global interface override is specified for use with the JTAG Port multiplexor where **TCK** is timed against all active RTCK ports of the multiplexor. Also, **TCK**, **TMS**, **TDI**, **nTRST**, and **nSRST** are driven to all ports as shown in Figure 3-49. This facilitates synchronized stop/step/start.

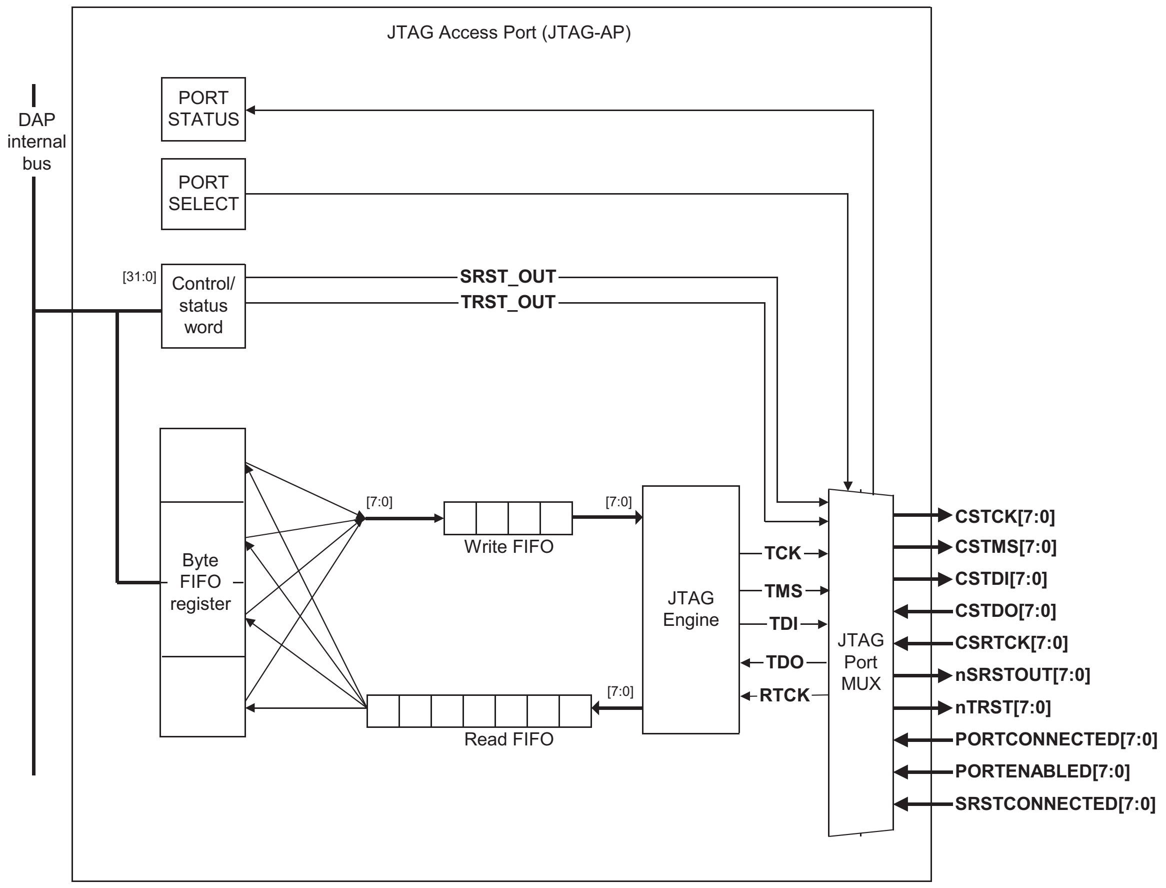


Figure 3-49 JTAG-AP block diagram

3.16.2 Internal DAP bus interface signals

These signals are described in *Internal DAP bus signals* on page 3-11.

3.16.3 JTAG to slave device signals

Table 3-45 shows the JTAG to slave device signals.

Table 3-45 JTAG to slave device signals

Name	Type	Description
nSRSTOUT[7:0]	Output	Subsystem Reset Out
SRSTCONNECTED[7:0]	Input	Subsystem Reset is present/wired
nCSTRST[7:0]	Output	JTAG Test Reset
CSTCK[7:0]	Output	JTAG Test Clock
CSTMS[7:0]	Output	JTAG Test Mode Select
CSTDI[7:0]	Output	JTAG Test Data In, to external TAP
CSTDO[7:0]	Input	JTAG Test Data Out, from external TAP
CSRTCK[7:0]	Input	Return Test Clock, target pacing signal
PORTCONNECTED[7:0]	Input	JTAG Port is connected, status signal
PORTENABLED[7:0]	Input	JTAG Port is enabled, for example, it might be deasserted by a core powering down

3.16.4 JTAG-AP programmer's model

Table 3-46 shows the JTAG-AP registers.

Table 3-46 JTAG-AP register summary

Offset	Type	Width	Reset value	Name
`0x00`	R/W	32	`0x00000000`	Control/Status Word, CSW
`0x04`	R/W	8	`0x00`	Port Select, PORTSEL
`0x08`	R/W	8	`0x00`	Port Status, PSTA
`0x0C`	-	-	-	Reserved
`0x10`	R/W	8	Undefined	Byte FIFO 1 Entry, BFIFO1
`0x14`	R/W	16	Undefined	Byte FIFO 2 Entry, BFIFO2
`0x18`	R/W	24	Undefined	Byte FIFO 3 Entry, BFIFO3

Table 3-46 JTAG-AP register summary (continued)

Offset	Type	Width	Reset value	Name
`0x1C`	R/W	32	Undefined	Byte FIFO 4 Entry, BFIFO4
`0x20-0xF8`	-	-	-	Reserved SBZ
`0xFC`	RO	32	`0x14760010`	Identification Register, IDR. Required by all access ports

JTAG-AP Control/Status Word Register, CSW, 0x00

The JTAG-AP control word is used to configure and control transfers through the JTAG interface. Figure 3-50 shows the JTAG-AP Control/Status Word Register bit assignments.

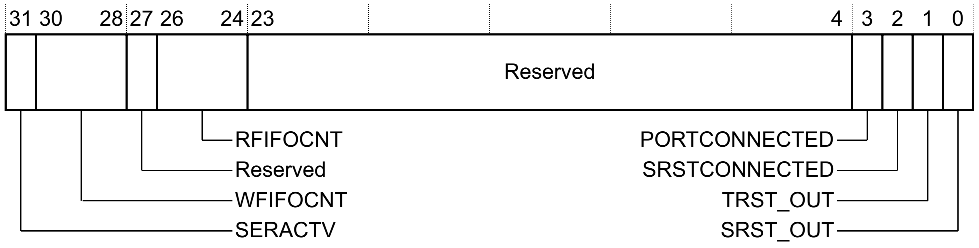


Figure 3-50 JTAG-AP Control/Status Word Register bit assignments

Table 3-47 shows the JTAG-AP Control/Status Word Register bit assignments. The register must not be modified while there are outstanding commands in the Write FIFO.

Table 3-47 JTAG-AP Control/Status Word Register bit assignments

Bits	Type	Name	Function
[31]	RO	SERACTV	JTAG Serializer active. Reset value = b0.
[30:28]	RO	WFIFOCNT	Outstanding Write FIFO Byte Count. Reset value = b000.
[27]	-	-	Reserved SBZ
[26:24]	RO	RFIFOCNT	Outstanding Read FIFO Byte Count. Reset value = b000
[23:4]	-	-	Reserved SBZ `0x00000`
[3]	RO	PORTCONNECTED	PORT Connected. AND of PORTCONNECTED inputs of currently selected ports.Reset value = b0.

Table 3-47 JTAG-AP Control/Status Word Register bit assignments (continued)

Bits	Type	Name	Function
[2]	RO	SRSTCONNECTED[a]	SRST Connected. AND of SRSTCONNECTED inputs of currently selected ports. If multiple ports are selected, it is the AND of all the SRSTCONNECTED inputs from the selected ports. Reset value = b0.
[1]	R/W	TRST_OUT	TRST Assert, not self clearing. JTAG TAP controller reset. Reset value = b0.
[0]	R/W	SRST_OUT	SRST Assert, not self clearing. Core Reset. Reset value = b0.

a. SRSTCONNECTED is a strap pin on the multiplexor inputs. It is set to 1 to indicate that the target JTAG device supports individual SRST controls.

JTAG-AP Port Select Register, PORTSEL, 0x04

The Port Select Register enables ports if connected and the slave port is currently enabled. The Port Select Register must be written when the TCK engine is idle, SERACTV=0, and WFIFO, WFIFOCNT=0, is empty. Writing at other times can generate unpredictable results.

Figure 3-51 shows the shows the JTAG-AP Port Select Register bit assignments.

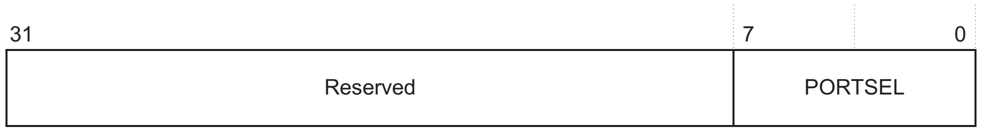


Figure 3-51 JTAG-AP Port Select Register bit assignments

Table 3-48 shows the JTAG-AP Port Select Register bit assignments.

Table 3-48 JTAG-AP Port Select Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved SBZ.
[7:0]	R/W	PORTSEL	Port Select. Reset value = b00000000.

JTAG-AP Port Status Register, PSTA, 0x08

The Port Status Register is a sticky register that captures the state of a connected and selected port on every clock cycle. If a connected and selected port is disabled or powered down, even transiently, the corresponding bit in the Port Status Register is set. It remains set until cleared by writing a one to the corresponding bit.

Figure 3-52 shows the JTAG-AP Port Status Register bit assignments.

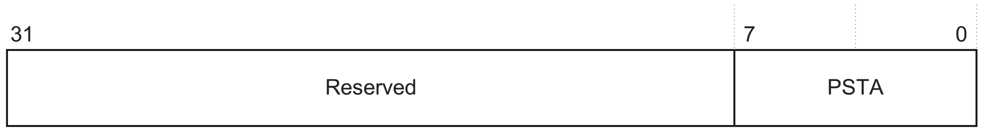


Figure 3-52 JTAG-AP Port Status Register bit assignments

Table 3-49 shows the JTAG-AP Port Status Register bit assignments.

Table 3-49 JTAG-AP Port Status Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved SBZ.
[7:0]	R/W	PSTA	Port Status. Reset value = b00000000.

JTAG-AP Byte FIFO registers, BFIFOn, 0x10-0x1C

The Byte FIFO registers are a word interface to one, two, three, or four parallel byte entries in the Byte Command FIFO, LSB first. The DAP Internal Bus is a 32-bit interface with no SIZE field. So, an address decoding is used to designate size, because the JTAG-AP Engine JTAG protocol is byte encoded. See *JTAG-AP byte command protocol* on page 3-122 for protocol details. Writes to the BFIFOx larger than the current write FIFO depth stall on **DAPREADY** in Normal mode. Reads to the BFIFOx larger than the current read FIFO depth stall on **DAPREADY** in Normal mode. For reads less than the full 32-bits, the upper bits are zero. For example, for a 24-bit read, **DAPRDATA[31:24]** is `0x00`.

JTAG-AP Identification Register

Figure 3-53 shows the JTAG-AP Identification Register bit assignments.

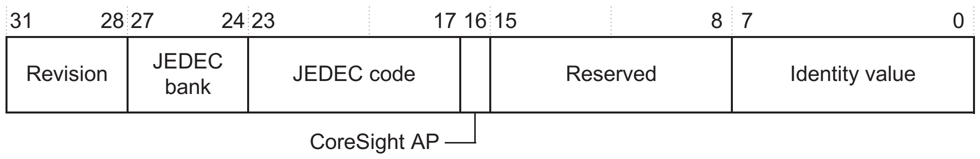


Figure 3-53 JTAG-AP Identification Register bit assignments

Table 3-50 shows the JTAG-AP Identification Register bit assignments.

Table 3-50 JTAG-AP Identification Register bit assignments

Bits	Type	Name
[31:28]	RO	Revision. Reset value is `0x1` for JTAG-AP.
[27:24]	RO	JEDEC bank[a].
[23:17]	RO	JEDEC code.
[16]	RO	CoreSight AP.
[15:8]	-	Reserved SBZ.
[7:0]	RO	Identity value. Reset value is `0x10` for JTAG-AP.

a. Using JEDEC bank `0x0` with a JEDEC code of `0x00` is reserved for use by ARM Limited.

3.16.5 Global port and RTCK

When more than one bit of PORTSEL[7:0] is set, all active JTAG-AP multiplexor port **RTCK**s are combinatorially joined, so that:

- If **TCK**=0 then select OR of active **RTCK**s.
- If **TCK**=1 then select AND of active **RTCK**s.

An active **RTCK** is generated by an active port which is defined as a port which:

- is selected, when its PORTSEL[7:0] bit is set
- is connected, when its PORTCONNECTED[7:0] bit is set
- has not been disabled or powered down in this session, when its PSTA bit is 0.

If no ports are active, **RTCK** is connected directly to **TCK**. This means that disabling or powering down a JTAG slave cannot lock up the **RTCK** interface.

3.16.6 RTCK wrapper

——— **Note** ———

This section applies only to synchronous TAP controllers.

————————

Where devices do not have a return clock, a **RTCK** wrapper must be used to register **TCK** against the core clock. See the applicable *CoreSight Design Kit Implementation and Integration Manual* for details on how to implement an **RTCK** wrapper.

3.17 JTAG-AP byte command protocol

JTAG commands are encoded as a byte protocol to provide high command packing. Table 3-51 shows the JTAG opcodes. x represents a don't care value.

- *TMS command bit assignments* describes the bit assignments for the TMS command.
- *TDI_TDO command bit assignments* on page 3-123 describes the bit assignments for the TDI_TDO command.

All JTAG commands, including **TMS** and **TDI** data, are sent to the BFIFOn write interfaces. Read data (**TDO**) is read from the BFIFOn read interfaces.

Table 3-51 JTAG byte command protocol

Opcode	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMS	0	x	x	x	x	x	x	x
TDI_TDO	1	0	0	x	x	x	x	x
Reserved	1	0	1	x	x	x	x	x
Reserved	1	1	0	x	x	x	x	x
Reserved	1	1	1	x	x	x	x	x

3.17.1 TMS command bit assignments

Table 3-52 shows the JTAG TMS bit assignments. S indicates the required TMS sequence, LSB first.

Table 3-52 JTAG TMS bit assignments

TCK cycles	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
5	0	TDI value	1	S_4	S3	S2	S1	S0
4	0	TDI value	0	1	S3	S2	S1	S0
3	0	TDI value	0	0	1	S2	S1	S0
2	0	TDI value	0	0	0	1	S1	S0
1	0	TDI value	0	0	0	0	1	S0

For example, to send four **TCK**s with TDI = '1' and TMS = [1 then 1 then 0 then 0] use b01010011. To send two **TCK**s with TDI = '0' and TMS = [1 then 0] use b00000101.

3.17.2 TDI_TDO command bit assignments

Table 3-53 shows the TDI_TDO command bit assignments. The key is as follows:

RBZ	Reserved but zero.
LastTMS	TMS is set to the value of lastTMS when the final TDI data is being output.
CaptureTDO	Set this bit to cause the capture of TDO into the read buffer. If a non-multiple of 8 bits is shifted out then the user must ignore the remaining bits in the last read byte.
TDIValue	Value of TDI for all cycles when UseTDI is set.
UseTDI	Set this bit to force TDI to always be the value of TDIValue.
LLLLLLL	7-bit binary count indicating the number of TDI bits to be sent in subsequent bytes, from 000000, length of 1, to 11111111, length of 128.
D	TDI value, least significant bit of the packet is sent first.

Table 3-53 TDI_TDO bit assignments

Byte name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TDI_TDO	1	0	0	RBZ	LastTMS	CaptureTDO	TDI value	UseTDI
TDI_Header	0	L_6	L_5	L_4	L_3	L_2	L_1	L_0
6-bit TDI packet	1	1	D_5	D_4	D_3	D_2	D_1	D_0
5-bit TDI packet	1	0	1	D_4	D_3	D_2	D_1	D_0
4-bit TDI packet	1	0	0	1	D_3	D_2	D_1	D_0
3-bit TDI packet	1	0	0	0	1	D_2	D_1	D_0
2-bit TDI packet	1	0	0	0	0	1	D_1	D_0
1-bit TDI packet	1	0	0	0	0	0	1	D_0

TDI_TDO sequences are as follows

Packed	TDI_TDO followed by packed data, a minimum of two bytes.
Nonpacked	TDI_TDO followed by TDI_Header, followed by 1-16 TDI data bytes, a minimum of three bytes up to a maximum of 18 bytes.

TDI values are sent in the bytes following a TDI_Header byte. The values are full bytes of TDI bit sequences, sent LSB first.

Note

If the TDI value in the TDI_TDO byte is set (UTDI=1), the value of any TDI specified in TDI_Packet or after TDI_Header is ignored, and TDI_Value is sent.

3.17.3 Transfer aborts

The DAP internal bus can become locked if tools incorrectly operate with the JTAG-AP read and write FIFOs. A write can stall indefinitely when the read FIFO is full. Reads can stall indefinitely if a read is performed when there are no more bytes of TDI to shift out, that is, the write FIFO is empty.

When tools have caused JTAG-AP to lock up and indefinitely hold **DAPREADY** LOW, the Debug Port can cause a transfer abort by asserting **DAPABORT**. In this case the JTAG-AP clears the read and write FIFOs and returns the Control Status Word Register and the Port Select and Port Status registers to reset conditions. See *DAP transfer aborting* on page 3-132 for more information.

Host software must also assume that the interface and the JTAG devices connected to JTAG-AP must be re-initialized.

3.17.4 BFIFOn interface examples

This section describes examples for:

- *Write FIFO*
- *Read FIFO* on page 3-125.

Write FIFO

The write FIFO is four bytes in length.

A write to BFIFO1 adds one byte to the write buffer, a write to BFIFO2 adds a halfword, a write to BFIFO3 adds three bytes, and a write to BFIFO4 adds one word. When a write to a sub-word register is performed, BFIFO1-BFIFO3, the higher order bits are ignored. See Example 3-1 on page 3-125.

Example 3-1 Write FIFO example sequences

To enter the bytes {0x12, 0x34, 0x56, 0x78} into the Write FIFO, where 0x12 is entered first and 0x78 last, any of the sequences below can be used.

In the format WFIFO[**DAPADDR**][**DAPWDATA**],WFIFO is a write to the BFIFO interface.

- WFIFO[0x10][7:0] = 0x12; WFIFO[0x10][7:0] = 0x34; WFIFO[0x10][7:0] = 0x56; WFIFO[0x10][7:0] = 0x78;
- WFIFO[0x1C][31:0] = 0x78563412;
- WFIFO[0x14][15:0] = 0x3412; WFIFO[0x14][15:0] = 0x7856;
- WFIFO[0x18][23:0] = 0x563412; WFIFO[0x10][7:0] = 0x78;
- WFIFO[0x10][7:0] = 0x12; WFIFO[0x14][15:0] = 0x5634; WFIFO[0x10][7:0] = 0x78;

Read FIFO

The read FIFO is seven bytes in length.

If a read is performed when there are no more bytes of TDI to shift out, that is, the read FIFO is empty, reads stall indefinitely.

If four bytes of TDI data is sent with the TDI_TDO byte initializing to capture TDO, we can read four bytes immediately following. A problem arises in that if only one, two, or three bytes that correspond to read data, the data rate drops accordingly. Seven bytes (4 + 1..3) is the minimum read FIFO size to pack this data. See Example 3-2.

Example 3-2 Read FIFO example sequences

To read the bytes {0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC} from the Read FIFO, where 0x12 was entered first and 0xBC last, the sequences below can be used.

In the format RFIFO[**DAPADDR**][**DAPRDATA**], RFIFO is a read from the BFIFO interface.

1. 0x78563412 = RFIFO[0x1C][31:0]; 0xBC9A = RFIFO[0x14][15:0];2. 0x563412 = RFIFO[0x18][23:0]; 0xBC9A78 = RFIFO[0x18][23:0];3. 0x12 = RFIFO[0x10][7:0]; 0x785634 = RFIFO[0x18][23:0]; 0xBC9A = RFIFO[0x14][15:0];

3.18 JTAG-AP clock domains

DAPCLK, the single JTAG-AP clock domain:

- drives the DAP bus interface
- provides access control for register reads and writes.

CSTCK is a control signal that acts as a clock or handshake signal to the target JTAG device. **CSTCK** is generated from a DAPCLK state machine, though ultimately its timing is against handshaking with **CSRTCK**, with the simple rule, do not change **CSTCK** while **CSRTCK** != **CSTCK**.

nSRSTOUT drives the subsystem reset. This is normally wired to the core reset of the target.

nCSTRST drives the JTAG slave **nTRST** port.

3.19 ROM table

The DAP provides an internal ROM table connected to the master Debug APB port of the APB-Mux. The Debug ROM table is loaded at address `0x00000000` and `0x80000000` of this bus and is accessible from both APB-AP and the system APB input. Bit [31] of the address bus is not connected to the ROM Table, ensuring that both views read the same value. The ROM table stores the locations of the components on the Debug APB. See the *CoreSight Architecture Specification* for more information.

The ROM table has a standard APB interface except for the exclusion of **PWRITEDBG** and **PWDATADBG**. All transfers are assumed to be reads. The ROM table is a read-only device and writes are ignored.

3.19.1 ROM table registers

Table 3-54 shows the ROM table registers. The values of the table entries depend on the CoreSight subsystem that is implemented.

Table 3-54 ROM table registers

Offset	Type	Bits	Name	Function
`0xFDC`	-	[7:0]	Peripheral ID7	Reserved SBZ. Read as `0x00`.
`0xFD8`	-	[7:0]	Peripheral ID6	Reserved SBZ. Read as `0x00`.
`0xFD4`	-	[7:0]	Peripheral ID5	Reserved SBZ. Read as `0x00`.
`0xFD0`	RO	[7:4]	Peripheral ID4	4KB count, set to `0x0`.
		[3:0]		JEP106 continuation code, implementation defined.
`0xFEC`	RO	[7:4]	Peripheral ID3	RevAnd, at top level, implementation defined.
		[3:0]		Customer Modified, implementation defined.
`0xFE8`	RO	[7:4]	Peripheral ID2	Revision number of Peripheral, implementation defined.
		[3]		1 = indicates that a JEDEC assigned value is used. 0 = indicates that a JEDEC assigned value is not used.
		[2:0]		JEP106 Identity Code [6:4], implementation defined.
`0xFE4`	RO	[7:4]	Peripheral ID1	JEP106 Identity Code [3:0], implementation defined.
		[3:0]		PartNumber1, implementation defined.
`0xFE0`	RO	[7:0]	Peripheral ID0	PartNumber0, implementation defined.

Table 3-54 ROM table registers (continued)

Offset	Type	Bits	Name	Function
`0xFF0`	RO	[7:0]	Component ID0	Preamble. Set to `0x0D`.
`0xFF4`	RO	[7:0]	Component ID1	Preamble. Set to `0x10`.
`0xFF8`	RO	[7:0]	Component ID2	Preamble. Set to `0x05`.
`0xFFC`	RO	[7:0]	Component ID3	Preamble. Set to `0xB1`.

The ROM table has a specific PrimeCell class. In all registers `0xFD0-0xFFC`, bits [31:8] are reserved and should be read as zero. Locations `0xF00-0xFCC` are reserved and should be read as zero.

3.19.2 ROM table entries

Table 3-55 shows the ROM table entries bit assignments for each entry in the `0x000-0xEFC` region.

Table 3-55 ROM table entries bit assignments

Bits	Name	Description
[31:12]	Address offset	Base address of the component, relative to the ROM address. Negative values are permitted using two's complement. ComponentAddress = ROMAddress + (AddressOffset SHL 12).
[11:2]	-	Reserved SBZ.
[1]	Format	1 = 32-bit format. In the DAP Debug ROM this is set to 1. 0 = 8-bit format.
[0]	Entry present	Set HIGH to indicate an entry is present.

The last entry in the ROM table has the value `0x00000000`, which is reserved.If the CoreSight component occupies several consecutive 4KB blocks, the base address of the lowest block in memory is given. The locations of components are stored in sequential locations with the ROM table. The entry following the last component in the table must read `0x00000000`, and subsequent locations are assumed to read as zero.

3.20 Connections to CoreSight components and system interfaces

Figure 3-54 shows how CoreSight components and a system bus can be accessed through the DAP using the AHB-AP. An AHB to APB bridge is used to access CoreSight component configuration registers. CoreSight components can be accessed by either software, or the DAP, or both, depending on the configuration of the AHB bus matrix.

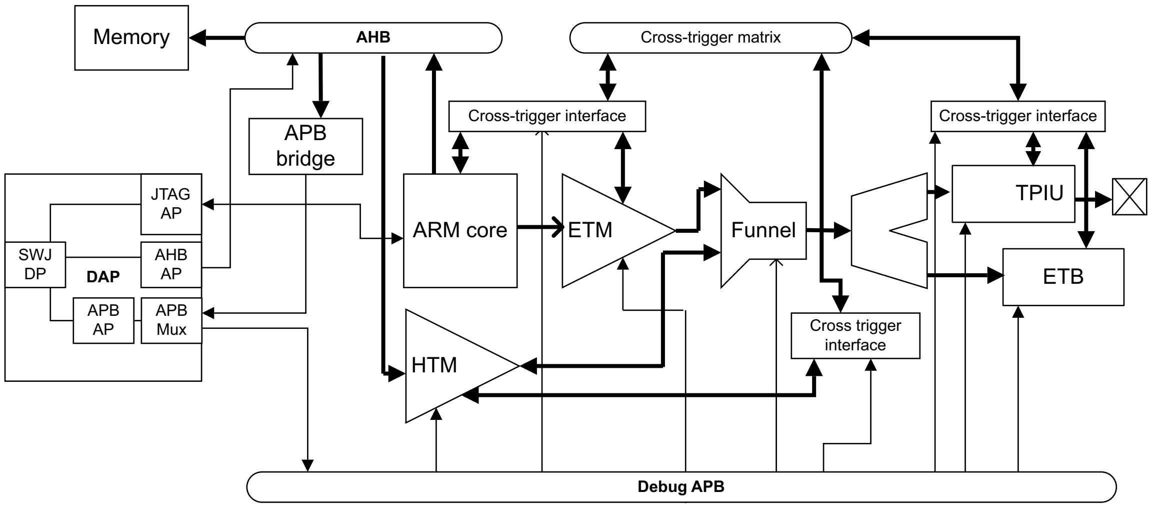


Figure 3-54 CoreSight trace with a single core

3.21 Authentication requirements for Debug Access Port

This section describes the functionality that must be available in the debug and trace components to permit authentication using the signals, and describes how they must be connected. If you do not require the system to support this level of control, you can simplify the system design.

The full authentication requirements are defined in the *CoreSight Architecture Specification*.

APB-AP has one authentication signal, called **DEVICEEN**:

- If the APB-AP is connected to a debug bus, this signal must be tied HIGH. In this release of the CoreSight Design Kit, the APB-AP must be connected to a debug bus, and **DEVICEEN** must be tied HIGH.
- If the APB-AP is connected to a system bus dedicated to the secure world, this signal must be connected to **SPIDEN**.
- If the APB-AP is connected to a system bus dedicated to the non-secure world, this signal must be connected to **DBGEN**.

For more information about **SPIDEN** and **DBGEN**, see the *CoreSight Architecture Specification*.

3.22 Clocks and resets

DAPCLK must be driven by a constant clock. It must not be stopped or altered while the DAP is in use. DAPCLKEN can be used as a clock gating term to reduce the effective clock speed from **DAPCLK**.

DAPRESETn initializes the state of all registers in the **DAPCLK** domain. **DAPRESETn** enables initialization of the DAP without affecting the normal operation of the SoC in which the DAP is integrated, and should be driven by the tools on external connection to the debug port. The reset can be initiated by writing to the control register of the JTAG-DP. This resets all the registers in the Debug clock domain, that is, Debug APB and DAP domains.

——— **Note** ———

- In a CoreSight system **DAPCLK** must be the same as **PCLKDBG**. APB-AP operates within one clock domain and requires **DAPCLK** to be the same as **PCLK**. The APB-Mux is designed to expect the Debug APB output to operate with the same clock and power domain as the APB-AP facing interface.
- For this release of the DAP, **DAPCLK** is not presented at the top level port list and is internally connected to **PCLKDBG**. **DAPCLKEN** is connected to **PCLKENDBG** and **DAPRESETn** is connected to **PRESETDBGn**.

————————————

3.23 DAP transfer aborting

Caution

The Transfer Abort operation is not intended for normal operation. It is not a recommended solution for tools to use routinely. Transfer Abort provides a safety mechanism to ensure that if a component connected to an access port fails to respond, then control of the DAP can be recovered.

The signal **DAPABORT** is driven HIGH by the debug port to initiate a DAP Transfer Abort as shown in Figure 3-55.

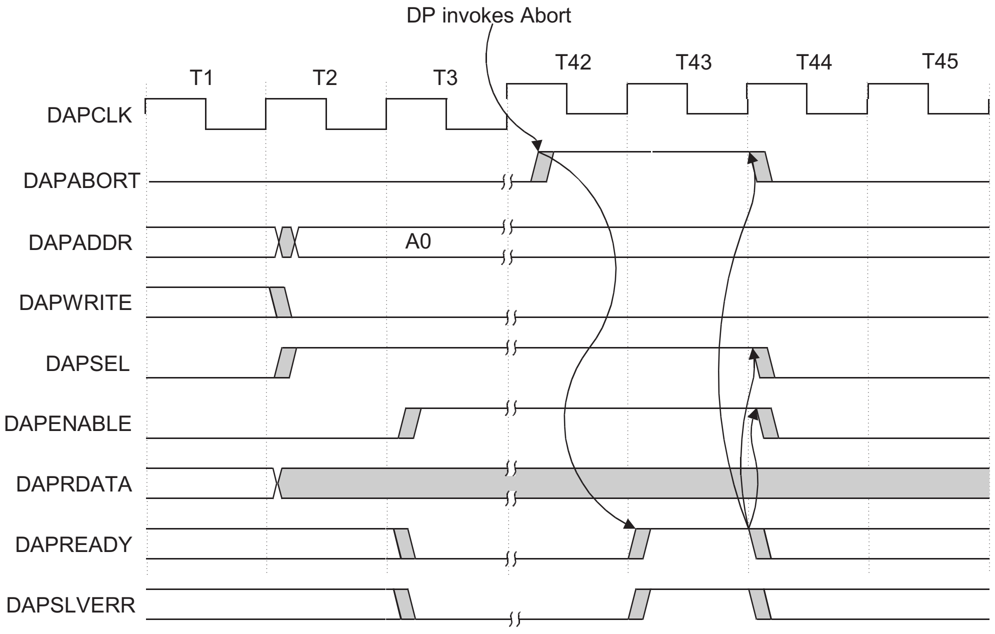


Figure 3-55 DAP read transfer with Transfer Abort

The selected access port must return **DAPREADY** HIGH within 32 DAP clock cycles to clear the current DAP internal transaction in progress before the tools can initiate a new transfer. A number of cycles are permitted to support future access ports that might be able to cancel system transactions. If the selected access port returns **DAPREADY** HIGH in the same cycle as the Transfer Abort is initiated, the Transfer Abort is ignored by the access port, and cancelled by the debug port. When a debug port initiated **DAPABORT** operation is invoked, the access port must return **DAPSLVERR** LOW when **DAPREADY** is asserted HIGH.

Any access port that can drive **DAPREADY** LOW must support the DAP Transfer Abort procedure to ensure that access can be regained to the DAP internal bus.

The effects of a DAP Transfer Abort operation on the selected access port to the system operation are implementation-defined. The access port can choose to abort the system transfer if the protocol the access port is driving supports it, or stall until the access completes.

- The JTAG-AP cancels the system transfer.
- The AHB-AP and APB-AP do not cancel the system operation and return **DAPREADY** HIGH one cycle after **DAPABORT** has gone HIGH as shown in Figure 3-55 on page 3-132. This is to ensure that the external AHB or APB protocol is not violated. Both AHB-AP and APB-AP only permit read access to the Control and Status Register to be performed while a system-facing transfer is stalled. This enables the transfer in progress bit to be read to determine if the system-facing transfer has completed. If the system transfer completes, the access port returns to a normal operational state.

An example use of **DAPABORT** is for the situation where the AHB-AP is stalled because a processor *Bus Interface Unit* (BIU) is waiting for a transfer to complete. The DAP transfer can be aborted and the APB-AP used to send a cancel BIU transaction operation to the processor using the debug interface. The AHB-AP transfer then completes and the AHB-AP can be accessed to determine the address being accessed when the deadlock occurred.

The initiation of a Transfer Abort operation results in any returned data on **DAPRDATA** being ignored, irrespective of the state of **DAPREADY**. If **DAPREADY** is returned HIGH the cycle the **DAPABORT** goes HIGH, then **DAPRDATA** is still ignored. An abort operation cannot be initiated at the start of a transfer, that is, when **DAPSEL** is HIGH and **DAPENABLE** is LOW, because this operation has just been downloaded from the tools to the debug port, and multiple cycles are required to download the next instruction, which could be the Abort.

Chapter 4
CoreSight Trace Sources

This chapter describes the CoreSight trace sources, HTMs and ETMs. It contains the following sections:

4.1 AMBA AHB Trace Macrocell

The *AHB Trace Macrocell* (HTM) gives visibility of bus information such as:

- an understanding of multi-layer bus utilization
- software debug such as visibility of access to memory areas and time correlation with CPU program flow and data accesses
- bus event detection for trace trigger or filters, and for bus profiling.

Figure 4-1 shows an HTM used in a multi-layer bus configuration.

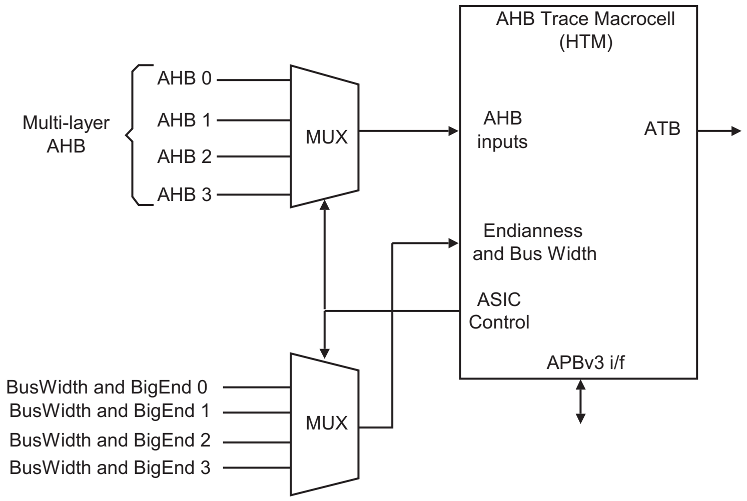


Figure 4-1 HTM in a multi-layer bus configuration

4.1.1 HTM ports

HTM ports are:

- AHB ports
- APB programming interface
- ATB interface for generated trace data
- Support for connection of CTIs, or triggers.

See the *AMBA AHB Trace Macrocell (HTM) Technical Reference Manual* for additional information on the HTM.

4.2 Embedded Trace Macrocells

CoreSight-compliant *Embedded Trace Macrocells* (ETMs) provide processor-driven trace through an ATB-compliant trace port. Configuration is supported through the CoreSight APB programming interface.

4.2.1 Ports

ETM ports are:

- Core interface
- APB programming interface
- ATB interface for generation of trace data
- Support for connection of CTIs, or triggers.

Further reading on page xxvii lists publications that provide additional information on ETMs.

4.2.2 Authentication requirements for CoreSight ETMs

This section describes the functionality that must be available in CoreSight ETMs to permit authentication using the authentication signals described in Chapter 2 *CoreSight Programmer's Model* and describes how they must be connected. If a system does not want to support this level of control, simplifications of the system design can be made.

4.2.3 Authentication signals for CoreSight ETMs

See the *ETM Architecture Specification* and the Technical Reference Manual for the applicable ARM processor for information.

Chapter 5
Embedded Cross Trigger

This chapter describes the *Embedded Cross Trigger* (ECT), *Cross Trigger Interface* (CTI), and the *Cross Trigger Matrix* (CTM). It contains the following sections:

5.1 About the Embedded Cross Trigger

The ECT for CoreSight consists of a number of CTIs and a CTM connected together. You can operate a single CTI without the requirement for a CTM.

5.1.1 How ECT works

The ECT provides an interface to the debug system as shown in Figure 5-1 on page 5-3. This enables ARM/ETM subsystems to interact, that is cross trigger, with each other. The debug system enables debug support for multiple cores, together with cross triggering between the cores and their respective ETMs.

The main function of the ECT (CTI and CTM) is to pass debug events from one processor to another. For example, the ECT can communicate debug state information from one core to another, so that program execution on both processors can be stopped at the same time if required.

***Cross Trigger Interface* (CTI)**

This block controls the *Trigger Interface* (TI). The CTI combines and maps the trigger requests, and broadcasts them to all other interfaces on the ECT as channel events. When the CTI receives a channel event it maps this onto a trigger output. This enables subsystems to cross trigger with each other. The receiving and transmitting of triggers is performed through the TI.

***Cross Trigger Matrix* (CTM)**

This block controls the distribution of channel events. It provides *Channel Interfaces* (CIs) for connection to either CTIs or CTMs. This enables multiple CTIs to be linked together.

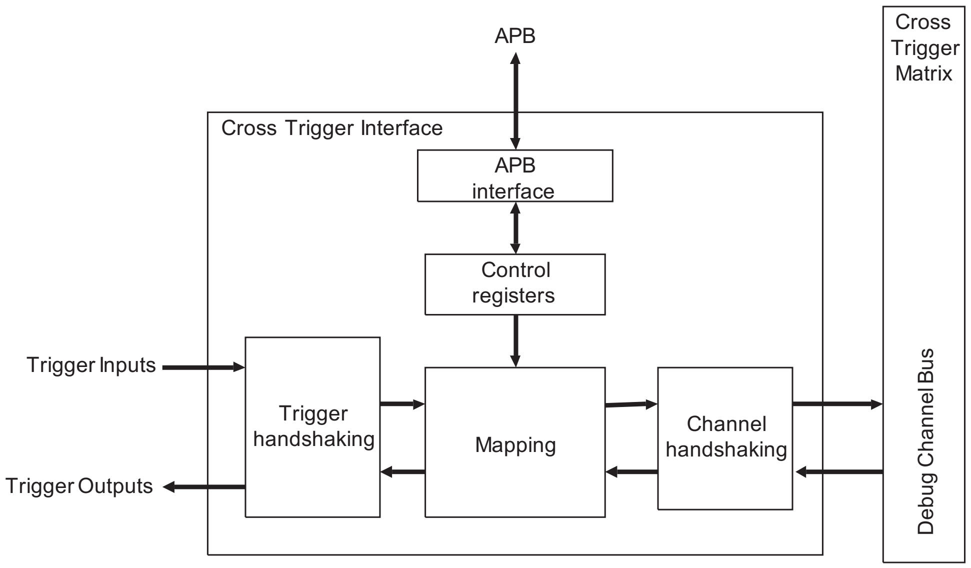


Figure 5-1 CoreSight CTI and CTM block diagram

5.1.2 CTI handshaking, synchronization, and clocks

This section describes handshaking, synchronization, and clocks in the following sections:

- *Interfaces handshake protocol*
- *Synchronization* on page 5-4
- *Clock information* on page 5-6.

Interfaces handshake protocol

The CTI does not interpret the signals on the CI or TI. If handshaking is enabled it is assumed the events are edge-triggered. As a result, closely occurring events can be unreliably recognized. An event signal is transmitted as a level. If you require edge detection or single pulse output you must implement the necessary shaping logic in the external wrapper.

To avoid any incompatibility, the following protocol has been defined:

- Only logic 1 is interpreted as an event.
- If the handshake is enabled, an output must stay active until an acknowledgement by hardware or optionally acknowledged by software for the TI is received, even if the acknowledge signal driver is deactivated.

If the handshaking is enabled, the CTI can only handle one-shot events. If events are close to one another, or multi-shot, and mapped to the same channel they are possibly merged into one event on an output trigger. For debug events such as breakpoint, trace start, and trace stop this does not cause a problem because input events mapped to the same triggers are required to do the same thing.

Events arising from different interfaces but mapped to the same channel might also be merged. This is acceptable because the mapping logic would have been programmed to allow this. Events can be merged because blocks handshake between asynchronous clock domains or channels events are mapped onto the same output trigger.

If the events broadcast on the CTI emanate from the same clock domain then you can bypass the handshaking logic. The output does not receive an acknowledgement, that is, No ack trigger class. In such cases you can use the CTI to transmit multiple shot events. The output has to remain active for at least one clock cycle to ensure it is captured at the destination interface.

Synchronization

Systems where events broadcast into the CTI can emanate from asynchronous clock domains must observe the following rules:

1. *Register input events*. Any signal that is an input of the CTI must be glitch-free to avoid any false event.

2. *Synchronize output events* on page 5-5. The outputs of the CTI must be synchronized to the local clock domain before being used by the subsystem, if such synchronization is not already present in the subsystem. The wrapper performs synchronization when necessary.

Register input events

The concern here is that a glitch on an output of the subsystem is propagated in the CTI and can be interpreted as an event. Also, the synthesis tools do not provide the necessary mechanism to constrain signals so that they are glitch free.

As an example, in an ARM9E design, **DBGACK** is the output of a combinatorial circuit, that is, a three-input OR gate. There is no way to ensure that a change of state on the inputs does not result in a glitch on the **DBGACK** signal, which can be fetched by an asynchronous circuit.

A single register can be used in this case or you can enable the synchronization logic in the CTI for that specific trigger input adding an extra clock cycle to the latency.

Synchronize output events

Figure 5-2 shows that a standard two-register synchronization is used.

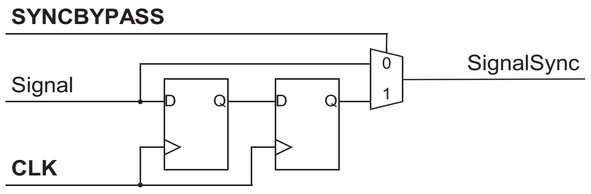


Figure 5-2 Standard synchronization

——— **Note** ———

It is important that you consult the design rules on the target process and the standard cell libraries to see if there are any restrictions or recommendations in relation to synchronizers. For example, the library rules might require the use of specially designed synchronization registers, or require that registers forming a synchronization chain must be placed in close proximity to each other.

———————————

Latency

The latency corresponds to the number of cycles necessary from the time one signal enters the ECT system until it is propagated to another processor. The path in the CTI and CTM is combinatorial, so that the latency is only because of the handshaking circuits. This means that, in the worst case, the latency is the sum of:

- one clock cycle of the CTI sending the event to the matrix, or removal of the glitch
- the combinatorial delay caused by the propagation of an **CTITRIGIN** event to an **CTITRIGOUT** output
- two clock cycles of the CTI receiving the event from the matrix, or synchronization to the clock of the receiving device.

In certain cases, it might be possible to reduce this latency:

- If the signal sent by the processor is the output of a flip-flop, you do not have to register the signal because it is glitch-free.
- If the core being interfaced to already features synchronization logic internally, for example, the processor has been specified to receive an asynchronous signal.
- If all the subsystems are synchronous, the TI bypass signals, **TISBYPASSACK** and **TISBYPASSIN**, can be activated. However, this path must respect the layout and synthesis timing constraints.

- If the CTM and CTI clocks are synchronous, the CI bypass signals, **CISBYPASSACK** and **CISBYPASSIN** can be activated.

Clock information

When a processor clock is stopped, for example, waiting for an interrupt, the corresponding CTI can receive an event from the CTM. When the CTI clock is the same as the subsystem clock and the handshaking is not bypassed, the CTM keeps the signal active until an acknowledgement is received, which only occurs when the clock is started again. In this case, out-of-date events can happen on the core. This does not inhibit the channel being used by other processors.

However, if the CTI clock differs from the local processor clock, for example, it is gated differently, it is possible for the CTI to raise an event to the core using **CTITRIGOUT**, while the processor clock is off. If raising an event to the core must be avoided, the processor must disable its CTI before stopping the clock.

It is assumed that in most systems the **CTICLK** is connected to the same clock as its local processor and the **CTMCLK** is connected to the fastest processor clock in the system so reducing the trigger latency and the requirement for clock enable ports.

If a CTI is going to be disabled while the processor enters a clock stopped mode, ARM Limited recommends the following:

- The CTI turns off the event-to-channel mapping, so that unwanted events are not generated to the CTM.
- The channel-to-event mapping circuit is turned off or disabled for a few clock cycles after the clock is on.
- This version of the CTI must not be connected to clocks that might be removed, that is stopped. This version of the CTI must not be placed within a power domain that can be turned off. In these scenarios it is recommended that **CTICLK** is the same as **CTMCLK**.

Linking CTIs and CTMs

Where the clock used on a CTI and a connected CTM, or CTM to CTM, or CTI to CTI, is asynchronous, then both the handshaking logic and synchronization registers must be left enabled, that is, **CIHSBYPASS** and **CISBYPASS** must be tied LOW.

If both devices have synchronous clocks then synchronization can be bypassed, **CISBYPASS** tied HIGH, to reduce latency.

If both clocks are the same, that is, **CTMCLK = CTICLK**, and the channel is required to send multi-shot events, the handshaking can be bypassed by tying CIHSBYPASS HIGH.

5.2 ECT programmer's model

The base addresses of the CTIs are not fixed, and can be different for any particular system implementation. However, the offset of any particular register from the base address is fixed. Each CTI has a 4KB programmer's model. Each CTI must be programmed separately. All unused memory space is reserved.

The following applies to all registers:

- Reserved or unused bits of registers must be written as 0, and ignored on a read unless otherwise stated in the text.
- All register bits are reset to 0 unless otherwise stated in the text.
- All registers must be accessed as words and so are compatible with big-endian and little-endian systems.

Note

The CTM does not have a programmer's model itself because it is a link between two or more CTIs, or more CTIs, or additional CTMs.

5.3 Summary of CTI registers

Table 5-1 shows the CTI programmable registers.

Table 5-1 CTI register summary

Offset	Name	Type	Width	Reset value	Description
`0x000`	CTICONTROL	R/W	1	`0x0`	See *CTI Control Register, CTICONTROL, 0x000* on page 5-11
`0x010`	CTIINTACK	WO	8	-	See *CTI Interrupt Acknowledge Register, CTIINTACK, 0x010* on page 5-11
`0x014`	CTIAPPSET	R/W	4	`0x0`	See *CTI Application Trigger Set Register, CTIAPPSET, 0x014* on page 5-12
`0x018`	CTIAPPCLEAR	WO	4	`0x0`	See *CTI Application Trigger Clear Register, CTIAPPCLEAR, 0x018* on page 5-13
`0x01C`	CTIAPPPULSE	WO	4	`0x0`	See *CTI Application Pulse Register, CTIAPPPULSE, 0x01C* on page 5-13
`0x020-0x03C`	CTIINEN	R/W	4	`0x00`	See *CTI Trigger to Channel Enable Registers, CTIINEN0-7, 0x020-0x03C* on page 5-14
`0x0A0-0x0BC`	CTIOUTEN	R/W	4	`0x00`	See *CTI Channel to Trigger Enable Registers, CTIOUTEN0-7, 0x0A0-0x0BC* on page 5-15
`0x130`	CTITRIGINSTATUS	RO	8	-	See *CTI Trigger In Status Register, CTITRIGINSTATUS, 0x130* on page 5-16
`0x134`	CTITRIGOUTSTATUS	RO	8	`0x00`	See *CTI Trigger Out Status Register, CTITRIGOUTSTATUS, 0x134* on page 5-16
`0x138`	CTICHINSTATUS	RO	4	-	See *CTI Channel In Status Register, CTICHINSTATUS, 0x138* on page 5-17
`0x13C`	CTICHOUTSTATUS	RO	4	`0x0`	See *CTI Channel Out Status Register, CTICHOUTSTATUS, 0x13C* on page 5-18
`0x140`	CTIGATE	R/W	4	`0xF`	See *Enable CTI Channel Gate Register, CTIGATE, 0x140* on page 5-18
`0x144`	ASICCTL	R/W	8	`0x00`	See *External Multiplexor Control Register, ASICCTL, 0x144* on page 5-20
`0xEDC`	ITCHINACK	WO	4	`0x0`	See *ITCHINACK Register, 0xEDC* on page 5-21

Table 5-1 CTI register summary (continued)

Offset	Name	Type	Width	Reset value	Description
`0xEE0`	ITTRIGINACK	WO	8	`0x00`	See *ITTRIGINACK Register, 0xEE0* on page 5-22
`0xEE4`	ITCHOUT	WO	4	`0x0`	See *ITCHOUT Register, 0xEE4* on page 5-22
`0xEE8`	ITTRIGOUT	WO	8	`0x00`	See *ITTRIGOUT Register, 0xEE8* on page 5-23
`0xEEC`	ITCHOUTACK	RO	4	`0x0`	See *ITCHOUTACK Register, 0xEEC* on page 5-23
`0xEF0`	ITTRIGOUTACK	RO	8	`0x00`	See *ITTRIGOUTACK Register, 0xEF0* on page 5-24
`0xEF4`	ITCHIN	RO	4	`0x0`	See *ITCHIN Register, 0xEF4* on page 5-24
`0xEF8`	ITTRIGIN	RO	8	`0x00`	See *ITTRIGIN Register, 0xEF8* on page 5-25
`0xEFC-0xF7C`	-	-	-	-	Reserved
`0xF00`	ITCTRL	R/W	1	`0x0`	See *CoreSight Management Registers summary* on page 2-12
`0xFA0`	Claim Tag Set	R/W	4	`0xF`	
`0xFA4`	Claim Tag Clear	R/W	4	`0x0`	
`0xFB0`	Lock Access Register	WO	32	-	
`0xFB4`	Lock Status Register	RO	2	`0x3`	
`0xFB8`	Authentication Status	RO	4	`0xA`	See *ECT CoreSight defined registers* on page 5-26
`0xFC0-0xFC4`	-	-	-	-	Reserved
`0xFC8`	Device ID	RO	20	`0x40800`	See *ECT CoreSight defined registers* on page 5-26
`0xFCC`	Device Type Identifier	RO	8	`0x14`	See *ECT CoreSight defined registers* on page 5-26
`0xFD0`	PeripheralID4	RO	8	`0x04`	See *CoreSight Management Registers summary* on page 2-12
`0xFD4`	PeripheralID5	-	-	-	Reserved

Table 5-1 CTI register summary (continued)

Offset	Name	Type	Width	Reset value	Description
`0xFD8`	PeripheralID6	-	-	-	Reserved
`0xFDC`	PeripheralID7	-	-	-	Reserved
`0xFE0`	PeripheralID0	RO	8	`0x06`	See *CoreSight Management Registers summary* on page 2-12
`0xFE4`	PeripheralID1	RO	8	`0xB9`	
`0xFE8`	PeripheralID2	RO	8	`0x0B`	
`0xFEC`	PeripheralID3	RO	8	`0x00`	
`0xFF0`	Component ID0	RO	8	`0x0D`	
`0xFF4`	Component ID1	RO	8	`0x90`	
`0xFF8`	Component ID2	RO	8	`0x05`	
`0xFFC`	Component ID3	RO	8	`0xB1`	

5.4 CTI register descriptions

This section describes the ECT CTI registers.

5.4.1 CTI Control Register, CTICONTROL, 0x000

The CTI Control Register enables the CTI.

Figure 5-3 shows the bit assignments.

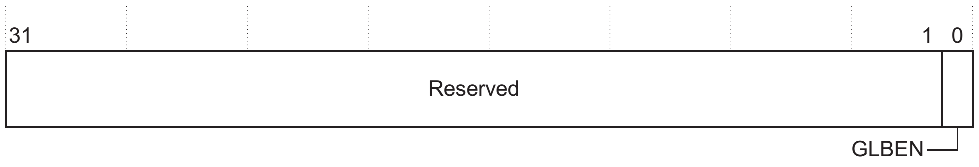


Figure 5-3 CTI Control Register bit assignments

Table 5-2 shows the bit assignments.

Table 5-2 CTI Control Register bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ DNM.
[0]	GLBEN	Enables or disables the ECT: 0 = disabled (reset) 1 = enabled. When disabled, all cross triggering mapping logic functionality is disabled for this processor.

5.4.2 CTI Interrupt Acknowledge Register, CTIINTACK, 0x010

The CTI Interrupt Acknowledge Register is write-only. Any bits written as a 1 cause the **CTITRIGOUT** output signal to be acknowledged. The acknowledgement is cleared when MAPTRIGOUT is deactivated. This register is used when the **CTITRIGOUT** is used as a sticky class output.

Figure 5-4 shows bit assignments.

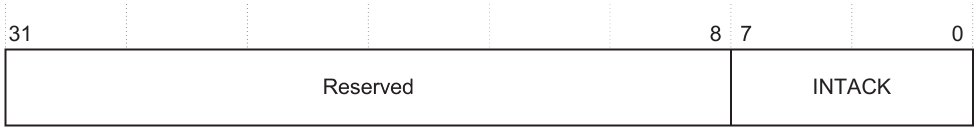


Figure 5-4 CTI Interrupt Acknowledge Register bit assignments

Table 5-3 shows the bit assignments.

Table 5-3 CTI Interrupt Acknowledge Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved RAZ DNM.
[7:0]	INTACK	Acknowledges the corresponding **CTITRIGOUT** output: 1 = **CTITRIGOUT** is acknowledged and is cleared when MAPTRIGOUT is LOW. 0 = no effect. There is one bit of the register for each **CTITRIGOUT** output.

5.4.3 CTI Application Trigger Set Register, CTIAPPSET, 0x014

The CTI Application Trigger Set Register is read/write. A write to this register causes a channel event to be raised, corresponding to the bit written to.

Figure 5-5 shows the bit assignments.

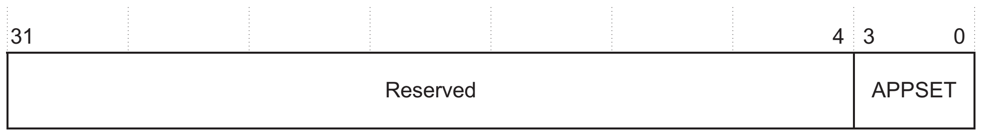


Figure 5-5 CTI Application Trigger Set Register bit assignments

Table 5-4 shows the bit assignments.

Table 5-4 CTI Application Trigger Set Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	APPSET	Setting a bit HIGH generates a channel event for the selected channel. Read: 0 = application trigger inactive (reset) 1 = application trigger active. Write: 0 = no effect 1 = generate channel event. There is one bit of the register for each channel.

Note

The CTIINEN registers do not affect the CTIAPPSET operation.

5.4.4 CTI Application Trigger Clear Register, CTIAPPCLEAR, 0x018

The CTI Application Trigger Clear Register is write-only. A write to this register causes a channel event to be cleared, corresponding to the bit written to.

Figure 5-6 shows the bit assignments.

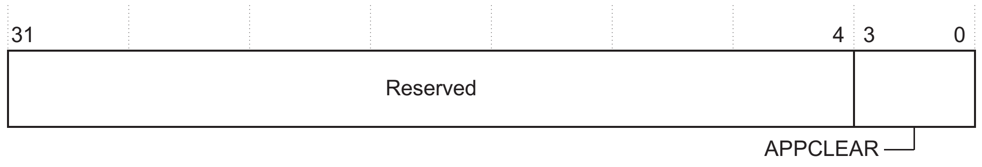


Figure 5-6 CTI Application Trigger Clear Register bit assignments

Table 5-5 shows the bit assignments.

Table 5-5 CTI Application Trigger Clear Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	APPCLEAR	Clears corresponding bits in the CTIAPPSET register. 1 = application trigger disabled in the CTIAPPSET register 0 = no effect. There is one bit of the register for each channel.

5.4.5 CTI Application Pulse Register, CTIAPPPULSE, 0x01C

The CTI Application Pulse Register is write-only. A write to this register causes a channel event pulse, one **CTICLK** period, to be generated, corresponding to the bit written to. The pulse external to the ECT can be extended to multi-cycle by the handshaking interface circuits. This register clears itself immediately, so it can be repeatedly written to without software having to clear it.

Figure 5-7 on page 5-14 shows the bit assignments.

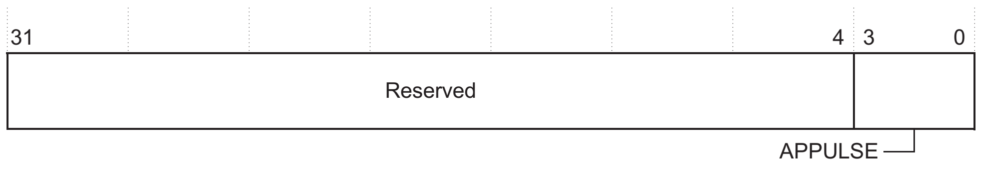


Figure 5-7 CTI Application Pulse Register bit assignments

Table 5-6 shows the bit assignments.

Table 5-6 CTI Application Pulse Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	APPULSE	Setting a bit HIGH generates a channel event pulse for the selected channel. Write: 1 = channel event pulse generated for one **CTICLK** period 0 = no effect. There is one bit of the register for each channel.

Note

The CTIINEN registers do not affect the CTIAPPPULSE operation.

5.4.6 CTI Trigger to Channel Enable Registers, CTIINEN0-7, 0x020-0x03C

The CTI Trigger to Channel Enable Registers enable the signalling of an event on CTM channels when the core issues a trigger, **CTITRIGIN**, to the CTI. There is one register for each of the eight **CTITRIGIN** inputs. Within each register there is one bit for each of the four channels implemented. These registers do not affect the application trigger operations.

Figure 5-8 shows the bit assignments.

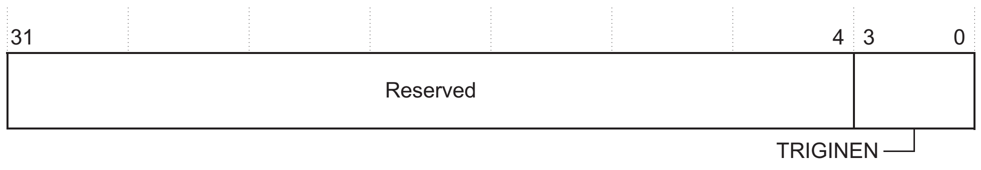


Figure 5-8 CTI Trigger to Channel Enable Registers bit assignments

Table 5-7 shows the bit assignments.

Table 5-7 CTI Trigger to Channel Enable Registers bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	TRIGINEN	Enables a cross trigger event to the corresponding channel when an **CTITRIGIN** is activated. 1 = enables the **CTITRIGIN** signal to generate an event on the respective channel of the CTM. There is one bit of the register for each of the 4 channels. For example in register CTIINEN0, TRIGINEN[0] set to 1 enables **CTITRIGIN** onto channel 0. 0 = disables the **CTITRIGIN** signal from generating an event on the respective channel of the CTM.

5.4.7 CTI Channel to Trigger Enable Registers, CTIOUTEN0-7, 0x0A0-0x0BC

The CTI Channel to Trigger Enable Registers define which channels can generate a **CTITRIGOUT** output. There is one register for each of the eight **CTITRIGOUT** outputs. Within each register there is one bit for each of the four channels implemented. These registers affect the mapping from application trigger to trigger outputs.

Figure 5-9 shows the bit assignments.

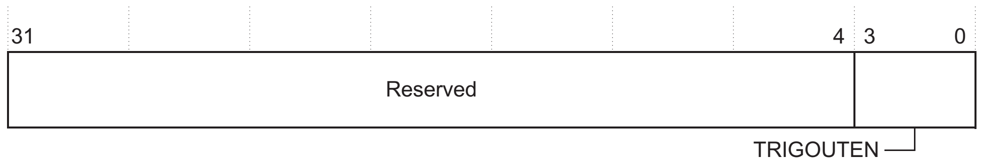


Figure 5-9 CTI Channel to Trigger Enable Registers bit assignments

Table 5-8 shows the bit assignments.

Table 5-8 CTI Channel to Trigger Enable Registers bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	TRIGOUTEN	Changing the value of this bit from a 0 to a 1 enables a channel event for the corresponding channel to generate an **CTITRIGOUT** output: 0 = the channel input (**CTICHIN**) from the CTM is not routed to the **CTITRIGOUT** output 1 = the channel input (**CTICHIN**) from the CTM is routed to the **CTITRIGOUT** output. There is one bit for each of the four channels. For example in register CTIOUTEN0, enabling bit 0 enables **CTICHIN[0]** to cause a trigger event on the **CTITRIGOUT[0]** output.

5.4.8 CTI Trigger In Status Register, CTITRIGINSTATUS, 0x130

The CTI Trigger In Status Register provides the status of the **CTITRIGIN** inputs.

Figure 5-10 shows the bit assignments.

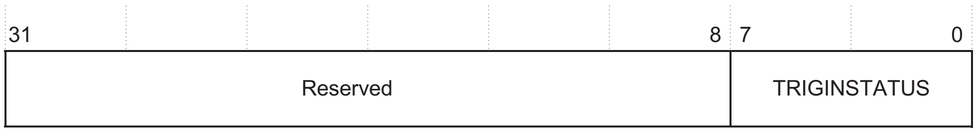


Figure 5-10 CTI Trigger In Status Register bit assignments

Table 5-9 shows the bit assignments.

Table 5-9 CTI Trigger In Status Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved RAZ DNM.
[7:0]	TRIGINSTATUS	Shows the status of the **CTITRIGIN** inputs: 1 = **CTITRIGIN** is active 0 = **CTITRIGIN** is inactive. Because the register provides a view of the raw **CTITRIGIN** inputs, the reset value is unknown. There is one bit of the register for each trigger input.

5.4.9 CTI Trigger Out Status Register, CTITRIGOUTSTATUS, 0x134

The CTI Trigger Out Status Register provides the status of the **CTITRIGOUT** outputs.

Figure 5-11 shows the bit assignments.

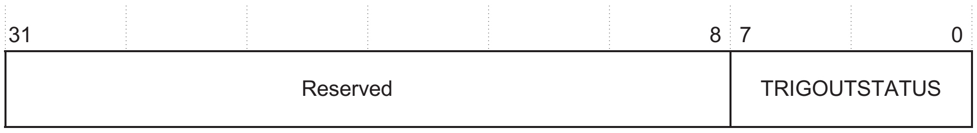


Figure 5-11 CTI Trigger Out Status Register bit assignments

Table 5-10 shows the bit assignments.

Table 5-10 CTI Trigger Out Status Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved RAZ DNM.
[7:0]	TRIGOUTSTATUS	Shows the status of the **CTITRIGOUT** outputs. 1 = **CTITRIGOUT** is active 0 = **CTITRIGOUT** is inactive (reset). There is one bit of the register for each trigger output.

5.4.10 CTI Channel In Status Register, CTICHINSTATUS, 0x138

The CTI Channel In Status Register provides the status of the CTI **CTICHIN** inputs.

Figure 5-12 shows the bit assignments.

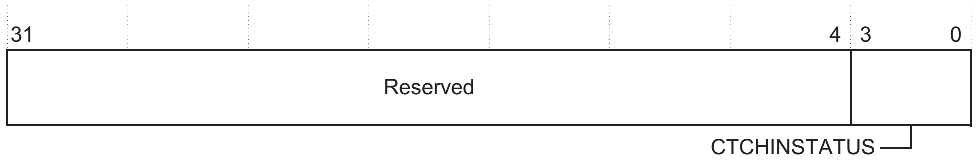


Figure 5-12 CTI Channel In Status Register bit assignments

Table 5-11 shows the bit assignments.

Table 5-11 CTI Channel In Status Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	CTICHINSTATUS	Shows the status of the **CTICHIN** inputs: 1 = **CTICHIN** is active 0 = **CTICHIN** is inactive. Because the register provides a view of the raw **CTICHIN** inputs from the CTM, the reset value is unknown. There is one bit of the register for each channel input.

5.4.11 CTI Channel Out Status Register, CTICHOUTSTATUS, 0x13C

The CTI Channel Out Status Register provides the status of the CTI **CTICHOUT** outputs.

Figure 5-13 shows the bit assignments.

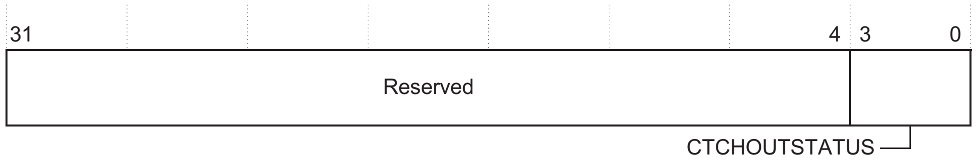


Figure 5-13 CTI Channel Out Status Register bit assignments

Table 5-12 shows the bit assignments.

Table 5-12 CTI Channel Out Status Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3:0]	CTICHOUTSTATUS	Shows the status of the **CTICHOUT** outputs. 1 = **CTICHOUT** is active 0 = **CTICHOUT** is inactive (reset). There is one bit of the register for each channel output.

5.4.12 Enable CTI Channel Gate Register, CTIGATE, 0x140

Figure 5-14 shows the bit assignments.

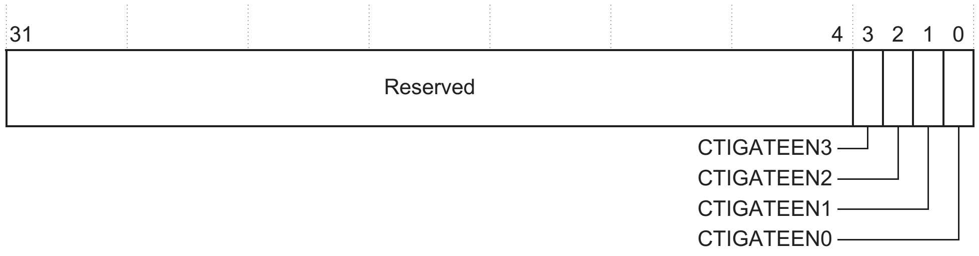


Figure 5-14 CTI Channel Gate Register bit assignments

Table 5-13 shows the bit assignments.

Table 5-13 CTI Channel Gate Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ DNM.
[3]	CTIGATEEN3	Enable CTICHOUT3. Set to 0 to disable channel propagation.
[2]	CTIGATEEN2	Enable CTICHOUT2. Set to 0 to disable channel propagation.
[1]	CTIGATEEN1	Enable CTICHOUT1. Set to 0 to disable channel propagation.
[0]	CTIGATEEN0	Enable CTICHOUT0. Set to 0 to disable channel propagation.

The Gate Enable Register prevents the channels from propagating through the CTM to other CTIs. This allows local cross-triggering, for example for causing an interrupt when the ETM trigger occurs. It can be used effectively with CTIAPPSET, CTIAPPCLEAR, and CTIAPPPULSE for asserting trigger outputs by asserting channels, without affecting the rest of the system. On reset, this register is `0xF`, and channel propagation is enabled.

Figure 5-15 shows channel gates used with the CTI.

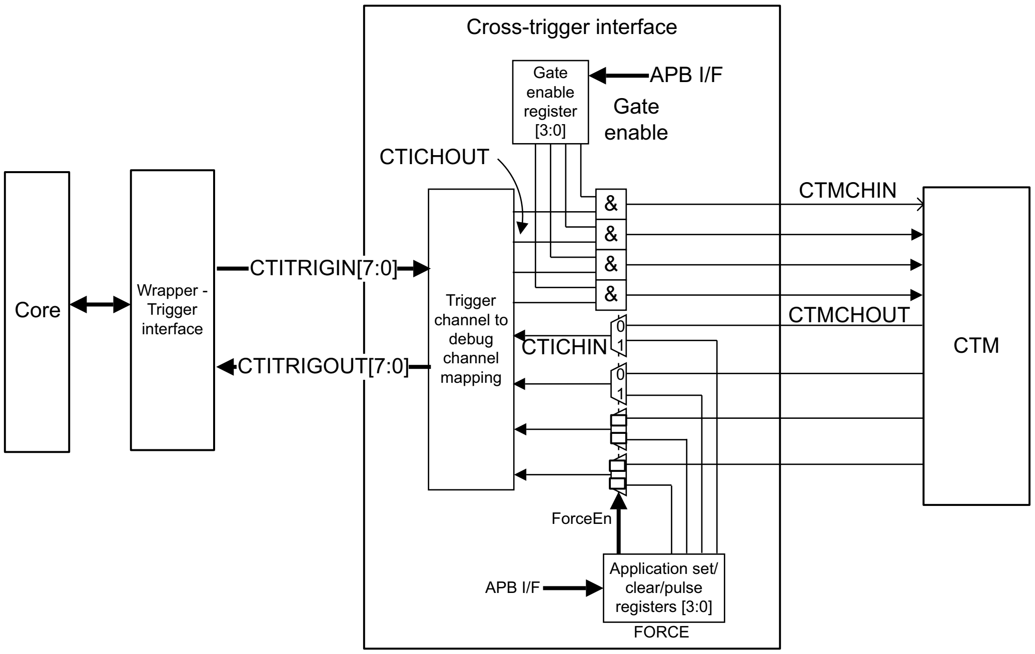

Cross-trigger interface
Gate enable register [3:0]
APB I/F
Gate enable
CTICHOUT
CTMCHIN
Core
Wrapper - Trigger interface
CTITRIGIN[7:0]
CTITRIGOUT[7:0]
Trigger channel to debug channel mapping
&
&
&
&
CTMCHOUT
CTICHIN
0
1
0
1
CTM
ForceEn
APB I/F
Application set/ clear/pulse registers [3:0]
FORCE

Figure 5-15 Channel gate used with the CTI

5.4.13 External Multiplexor Control Register, ASICCTL, 0x144

Figure 5-16 shows the bit assignments for the External Multiplexor Control Register.

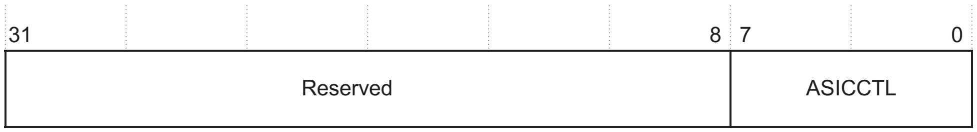


Figure 5-16 External Multiplexor Control Register bit assignments

Table 5-14 shows the bit assignments for the External Multiplexor Control Register.

Table 5-14 External Multiplexor Control Register bit assignments

Bits	Name	Description
[31:8]	-	Reserved RAZ DNM.
[7:0]	ASICCTL	Implementation-defined ASIC control, value written to the register is output on **ASICCTL[7:0]**. If external multiplexing of trigger signals is implemented then the number of multiplexed signals on each trigger must be reflected within the DevID Register. See *ECT CoreSight defined registers* on page 5-26.

5.5 ECT Integration Test Registers

Integration Test Registers are provided to simplify the process of verifying the integration of the ECT with other devices in a CoreSight system. These registers enable direct control of outputs and the ability to read the value of inputs. You must only use these registers when the Integration Test Control Register (`0xF00`) bit [0] is set to 1.

For details of how to use these signals, see the applicable *CoreSight Design Kit Implementation and Integration Manual*.

This sections describes the following registers:

- *ITCHINACK Register, 0xEDC*
- *ITTRIGINACK Register, 0xEE0* on page 5-22
- *ITCHOUT Register, 0xEE4* on page 5-22
- *ITTRIGOUT Register, 0xEE8* on page 5-23
- *ITCHOUTACK Register, 0xEEC* on page 5-23
- *ITTRIGOUTACK Register, 0xEF0* on page 5-24
- *ITCHIN Register, 0xEF4* on page 5-24
- *ITTRIGIN Register, 0xEF8* on page 5-25.

5.5.1 ITCHINACK Register, 0xEDC

This register is a write-only register. Figure 5-17 shows the bit assignments.

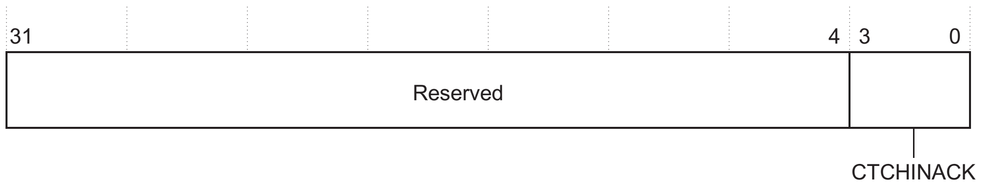


Figure 5-17 ITCINACK Register bit assignments

Table 5-15 shows the bit assignments.

Table 5-15 ITCHINACK Register bit assignments

Bits	Type	Name	Function
[31:4]	-	-	Reserved
[3:0]	WO	CTCHINACK	Set the value of the CTCHINACK outputs

5.5.2 ITTRIGINACK Register, 0xEE0

This register is a write-only register. Figure 5-18 shows the bit assignments.

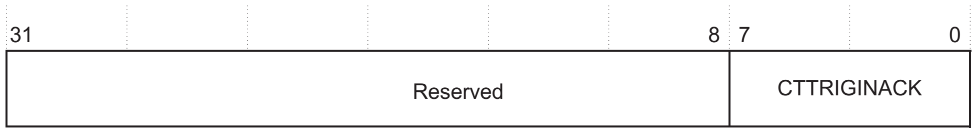


Figure 5-18 ITTRIGINACK Register bit assignments

Table 5-16 shows the bit assignments.

Table 5-16 ITTRIGINACK Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved
[7:0]	WO	CTTRIGINACK	Set the value of the **CTTRIGINACK** outputs

5.5.3 ITCHOUT Register, 0xEE4

This register is a write-only register. Figure 5-19 shows the bit assignments.

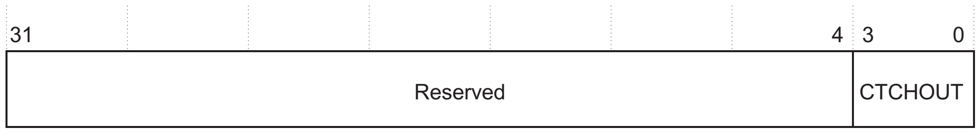


Figure 5-19 ITCHOUT Register bit assignments

Table 5-17 shows the bit assignments.

Table 5-17 ITCHOUT Register bit assignments

Bits	Type	Name	Function
[31:4]	-	-	Reserved
[3:0]	WO	CTCHOUT	Set the value of the **CTCHOUT** outputs

5.5.4 ITTRIGOUT Register, 0xEE8

This register is a write-only register. Figure 5-20 shows the bit assignments.

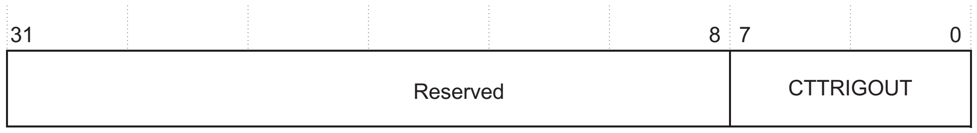


Figure 5-20 ITTRIGOUT Register bit assignments

Table 5-18 shows the bit assignments.

Table 5-18 ITTRIGOUT Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved
[7:0]	WO	CTTRIGOUT	Set the value of the **CTTRIGOUT** outputs

5.5.5 ITCHOUTACK Register, 0xEEC

This register is a read-only register. Figure 5-21 shows the bit assignments.

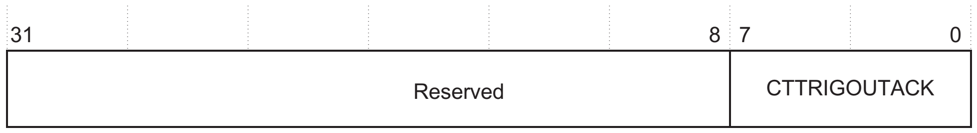


Figure 5-21 ITCHOUTACK Register bit assignments

Table 5-19 shows the bit assignments.

Table 5-19 ITCHOUTACK Register bit assignments

Bits	Type	Name	Function
[31:4]	-	-	Reserved
[3:0]	RO	CTCHOUTACK	Read the values of the **CTCHOUTACK** inputs

5.5.6 ITTRIGOUTACK Register, 0xEF0

This register is a read-only register. Figure 5-22 shows the bit assignments.

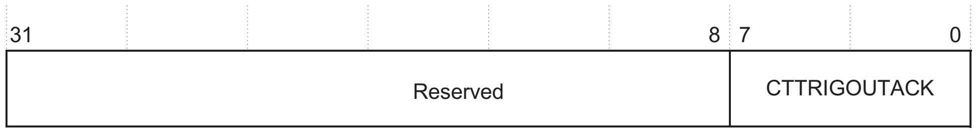


Figure 5-22 ITTRIGOUTACK Register bit assignments

Table 5-20 shows the bit assignments.

Table 5-20 ITTRIGOUTACK Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved
[7:0]	RO	CTTRIGOUTACK	Read the values of the **CTTRIGOUTACK** inputs

5.5.7 ITCHIN Register, 0xEF4

This register is a read-only register. Figure 5-23 shows the bit assignments.

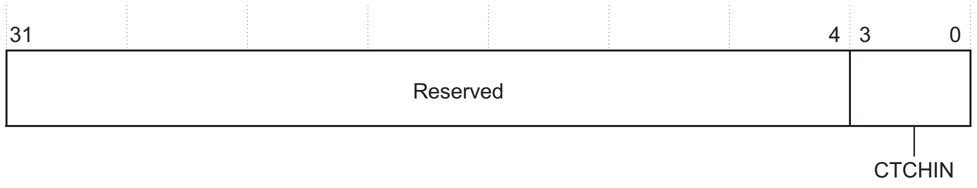


Figure 5-23 ITCHIN Register bit assignments

Table 5-21 shows the bit assignments.

Table 5-21 ITCHIN Register bit assignments

Bits	Type	Name	Function
[31:4]	-	-	Reserved
[3:0]	RO	CTCHIN	Read the values of the **CTCHIN** inputs

5.5.8 ITTRIGIN Register, 0xEF8

This register is a read-only register. Figure 5-24 shows the bit assignments.

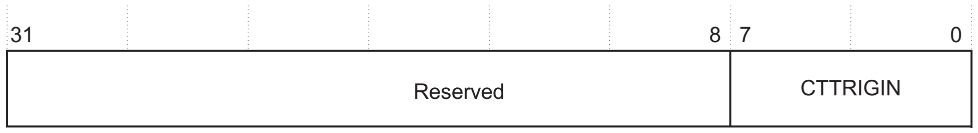


Figure 5-24 ITTRIGIN Register bit assignments

Table 5-22 shows the bit assignments.

Table 5-22 ITTRIGIN Register bit assignments

Bits	Type	Name	Function
[31:8]	-	-	Reserved
[7:0]	RO	CTTRIGIN	Read the values of the **CTTRIGIN** inputs

5.6 ECT CoreSight defined registers

See Chapter 2 *CoreSight Programmer's Model* for the definitions of these registers.

The information given here is specific to the ECT:

Authentication Status Register, 0xFB8

Reports the required security level. Table 5-23 shows the authentication values.

Table 5-23 Authentication values for ECT

Bits	Value	Description
[31:4]	`0x0000000`	Reserved
[3]	-	Current value of noninvasive debug enable signals
[2]	1'b1	Non-invasive debug controlled
[1]	-	Current value of invasive debug enable signals
[0]	1'b1	Invasive debug controlled

Device ID, 0xFC8

The CTI has the default value of `0x40800`. Table 5-24 shows the Device ID bit values.

Table 5-24 Device ID bit values

Bits	Value	Description
[31:20]	`0x000`	Reserved.
[19:16]	`0x4`	Number of ECT channels available.
[15:8]	`0x08`	Number of ECT triggers available.
[7:5]	3'b000	Reserved.
[4:0]	Implementation defined	Indicates the number of multiplexing available on Trigger Inputs and Trigger Outputs using **ASICCTL**. Default value of 5'b00000 indicating no multiplexing present. Reflects the value of the Verilog `define EXTMUXNUM which must be altered by the user accordingly.

Device Type Identifier, 0xFCC

`0x14` indicates this device is a debug control logic component and specifically an ECT.

Part number, 0xFE4[3:0], 0xFE0[7:4], and 0xFE0[3:0]

Upper, middle and lower BCD value of Device number. Set to `0x906`.

5.7 ECT connectivity recommendations

This section describes which signals to connect to a CTI in the ECT in a system based on an ARM7, ARM9, or ARM11 device.

The recommendations are to ensure that:

- Basic functionality is present in all ARM systems.
- Development tools wanting to perform complex cross-triggering can do so in most cases without modification by the user of default configuration values.

ECT support by development tools might only extend to naming the inputs and outputs and allowing their connection to be programmed, or it might program the ECT automatically for example for certain predefined profiling tasks.

If extra trigger inputs and outputs are required then you can use a second CTI.

Note

The inclusion of signals in this document does not constitute a commitment by ARM Limited to provide features in their development tools which specifically use those signals.

5.7.1 Connections for ARM7 and ARM9 systems

Table 5-25 shows the recommended trigger input connections for use with ARM7 and ARM9 systems.

Table 5-25 Trigger input connections for ARM7 and ARM9 systems

Trigger input bit	Source signal	Source device	Comments
[7]	**RANGEOUT[1]**	Core	Recommended.
[6]	**RANGEOUT[0]**	Core	Recommended. Can be acquired through the ETM and **EXTOUT[0]** if necessary.
[5]	**EXTOUT[1]**	ETM	Recommended if ETM present supporting at least two external outputs.
[4]	**EXTOUT[0]**	ETM	Compulsory if ETM present.
[3]	**ACQCOMP**	ETB	See FULL.

Table 5-25 Trigger input connections for ARM7 and ARM9 systems (continued)

Trigger input bit	Source signal	Source device	Comments
[2]	**FULL**	ETB	Recommended if an ETB is present. If a single ETB is shared between multiple cores, only connect to the CTI of one of the cores.
[1]	User defined	-	-
[0]	**DBGACK**	Core	Compulsory.

Inputs not connected as shown in Table 5-25 on page 5-28 can be used for user-defined functions.

Table 5-26 shows the recommended trigger output connections for ARM7 and ARM9 systems.

Table 5-26 Trigger output connections for ARM7 and ARM9 systems

Trigger output bit	Destination signal	Destination device	Comments
[7]	**DBGEXT[1]/EXTERN1**	Core	Recommended.
[6]	**DBGEXT[0]/ EXTERN0**	Core	Recommended.
[5]	**EXTIN[1]**	ETM	Recommended if ETM present.
[4]	**EXTIN[0]**	ETM	Compulsory if ETM present.
[3]	**VICINTSOURCE[y]**	VIC	Recommended if the VIC is present. Any interrupt can be used.
[2]	**VICINTSOURCE[x]**	VIC	Compulsory if the VIC is present. Any interrupt can be used.
	!nIRQ	Core	Compulsory if the VIC is not present. The output must be negated and connected to the core **nIRQ** input, ANDed with other signals using this input.
[1]	User-defined	-	-
[0]	**DBGRQ/EDBGRQ**	Core	Compulsory.

Outputs not connected as shown in Table 5-26 can be used for user-defined functions.

5.7.2 Connections for ARM11 systems

Table 5-27 shows the recommended input connections for use with ARM11 systems.

Table 5-27 Recommended input connections for use with ARM11 systems

Trigger input bit	Source signal	Source device	Comments
[7]	**!INT**	L2CC Event Monitor	Compulsory if L2CC Event Monitor present. **INT** must be negated. This is to ensure the same programming (active LOW) whether it is connected directly to **nIRQ** or through the ECT. Any L2CC event can be passed to the ECT by configuring a counter to select the event and generate an interrupt on increment. Set the interrupt type as edge-sensitive, 1-cycle pulse duration and active LOW.
[6]	**TRIGGER**	ETM	Recommended if ETM present.
[5]	**EXTOUT[1]**	ETM	Recommended if ETM present.
[4]	**EXTOUT[0]**	ETM	Compulsory if ETM present.
[3]	**ACQCOMP**	ETB	See FULL.
[2]	**FULL**	ETB	Recommended if an ETB is present. If a single ETB is shared between multiple cores, only connect to the CTI of one of the cores.
[1]	**!nPMUIRQ**	Core	Compulsory. **nPMUIRQ** must be negated.
[0]	**DBGACK**	Core	Compulsory.

Inputs not connected as shown in Table 5-27 can be used for user-defined functions.

Table 5-28 shows the recommended output connections for use with ARM11 systems.

Table 5-28 Recommended output connections for use with ARM11 systems

Trigger output bit	Destination signal	Destination device	Comments
[7]	**ETMEXTOUT[1]**	Core	See **ETMEXTOUT[0]**.
[6]	**ETMEXTOUT[0]**	Core	Recommended. Connect to the ECT instead of the signal of the same name on the ETM.
[5]	**EXTIN[1]**	ETM	Recommended if ETM present.
[4]	**EXTIN[0]**	ETM	Compulsory if ETM present.
[3]	**VICINTSOURCE[y]**	VIC	Recommended if the VIC is present. Any interrupt can be used.

Table 5-28 Recommended output connections for use with ARM11 systems (continued)

Trigger output bit	Destination signal	Destination device	Comments
[2]	**VICINTSOURCE[x]**	VIC	Compulsory if the VIC is present. Any interrupt can be used.
	!nIRQ	Core	Compulsory if the VIC is not present. The output must be negated and connected to the core **nIRQ** input, ANDed with other signals using this input.
[1]	User-defined	-	-
[0]	**EDBGRQ**	Core	Compulsory.

Outputs not connected as shown in Table 5-28 on page 5-30 can be used for user-defined functions.

5.7.3 Connections for CoreSight components

The following signals are expected to be connected to local CTIs but the exact connection location to the CTI is not mandated because it can be establishing using topology detection through full control of source and destination ports.

Note

Where two signals are listed, the second corresponds to an acknowledgement that must be connected to the corresponding **ACK** signal.

Table 5-29 shows the trigger inputs to the CTI.

Table 5-29 Trigger inputs to the CTI

CoreSight component	Component signal
HTM	**HTMEXTOUT[1]**
	HTMEXTOUT[0]
	HTMTRIGGER, **HTMTRIGGERACK**

Table 5-30 shows the trigger outputs from the CTI.

Table 5-30 Trigger outputs from CTI

CoreSight component	Component signal
HTM	**HTMEXTIN[1]**
	HTMEXTIN[0]
	HTMTRACEDISABLE
ETB	**FLUSHIN**, **FLUSHINACK**
	TRIGIN, **TRIGINACK**
TPIU	**FLUSHIN**, **FLUSHINACK**
	TRIGIN, **TRIGINACK**

5.8 ECT authentication requirements

This section describes the functionality that must be available in the ECT to permit authentication using the signals described in the *CoreSight Architecture Specification*, and describes how they must be connected. If a system does not support this level of control, then simplifications of the system design can be made.

The device does not require any inputs capable of disabling it as a whole. While it is possible for the device to be invasive, by asserting interrupts, it must continue to function when **DBGEN** is LOW, because it might be required for non-invasive debugging, for example to communicate profiling events or a trigger condition. As a result, individual trigger inputs and outputs must be masked as required.

5.8.1 Trigger inputs

The trigger inputs should be masked by **NIDEN** where they are not connected to another debug or trace device.

5.8.2 Trigger outputs

The trigger outputs should be masked by **DBGEN** where they might otherwise affect the behavior of a running system and do not first require to be specifically enabled by the system. Table 5-31 shows the signals recommended in the *CoreSight Architecture Specification*.

Table 5-31 ECT recommended trigger outputs

Destination signal	Destination device	Masking required	Comments
DBGRQ/ EDBGRQ	Core	No	This signal is ignored when **DBGEN** is LOW, and requires no further masking.
VICINTSOURCE	VIC	No	Privileged system software must explicitly enable the interrupt source for this to have any effect.
nIRQ (inverted)	Core	Yes	Directly changes the execution flow of the core.
EXTIN	ETM	No	Only affects the operation of the ETM.
DBGEXT/ EXTERN0/ EXTERN1	Core	No	Only affects the operation of the debug logic.
ETMEXTOUT	Core	No	Only affects the operation of the PMU.

5.8.3 ECT authentication signals

Table 5-32 shows the required authentication signals.

Table 5-32 ECT authentication signals

Signal	In/out	Description
DBGEN	Input	Debug enable signal to mask trigger outputs from a CTI. Note that this signal already exists with different functionality, disabling the entire CTI.
NIDEN	Input	Non-invasive debug enable input to mask the trigger inputs that are not connected to a trace or debug device.
TINIDENSELx[7:0]	Input	Select to enable individual trigger inputs to be masked when **NIDEN** is LOW. Each bit of **TINIDENSELx[7:0**] must be set HIGH to bypass **NIDEN**, or LOW to be masked by **NIDEN**.
TODBGENGSELx[7:0]	Input	Select to enable individual trigger outputs to be masked when **DBGEN** is LOW. Each bit of **TODBGENSELx[7:0]** must be set HIGH to bypass **DBGEN**, or LOW to be masked by **DBGEN**.

If **NIDEN** is LOW and **TINIDENSELx** is LOW, then any read of the CTI Trigger In Status Register returns the corresponding bit LOW. This applies to both Normal operation mode and Integration test mode. If the CTI is configured to generate trigger acknowledgements this must be maintained in normal mode, irrespective of the state of **NIDEN**.

If **DBGEN** is LOW and **TODBGENSELx** is LOW, then any read of the CTI Trigger Out Status Register register returns the corresponding bit LOW. The CTTRIGOUTx register is also LOW. This applies to normal operation mode. In Integration test mode all writes to the ITTRIGOUT Register are ignored. The CTTRIGOUTx register is LOW.

Chapter 6
ATB 1:1 Bridge

This chapter describes the *AMBA Trace Bus* (ATB) 1:1 bridge, which enables design integrators to meet timing closure that can occur on an ATB system. It contains the following sections:

6.1 About the ATB 1:1 bridge

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, a slave and a master. Both interfaces exist in the same clock domain which is why it is referred to as a 1:1 bridge.

There are no programmable registers and the component appears transparent to the user.

6.1.1 Normal operation

The synchronous bridge acts as a buffering element and adds a minimum of one clock cycle delay to ATB transactions. As shown in Figure 6-1, when the buffer is empty it can assert **ATREADYS** and can accept a packet of data from the trace source, **ATVALIDS**. When the buffer is full, this can be indicated using **ATVALIDM** and then the buffer can be emptied when **ATREADYM** goes HIGH.

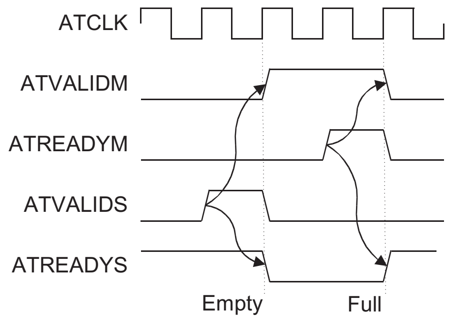


Figure 6-1 ATB basic operation

6.1.2 Flushing operation

Figure 6-2 on page 6-3 shows how a flush appears on the two interfaces to the bridge. All signals are registered so **AFVALIDS** goes HIGH one cycle later than **AFVALIDM**. This also causes a cycle delay of the flush acknowledgement, **AFREADYS** to **AFREADYM**.

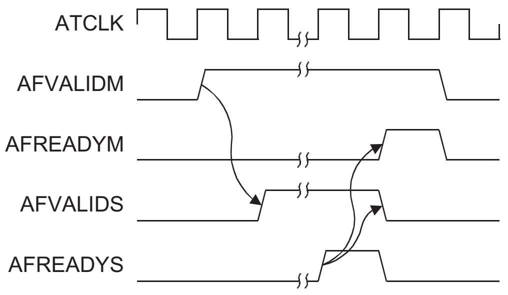


Figure 6-2 ATB flushing operation on master and slave interfaces

6.1.3 Flushing and data packets

The return of **AFREADYM** must be held up if there is data associated with the flush event. The indication of the flush completion must be delayed until all historical data has been transmitted. This includes data stored within the synchronous bridge.

Figure 6-3 shows an example of such a sequence. **AFREADYM** is only indicated after **AFREADYS** has been captured, from the trace source, and the data packet within the bridge has been accepted by **ATREADYM**, the trace sink.

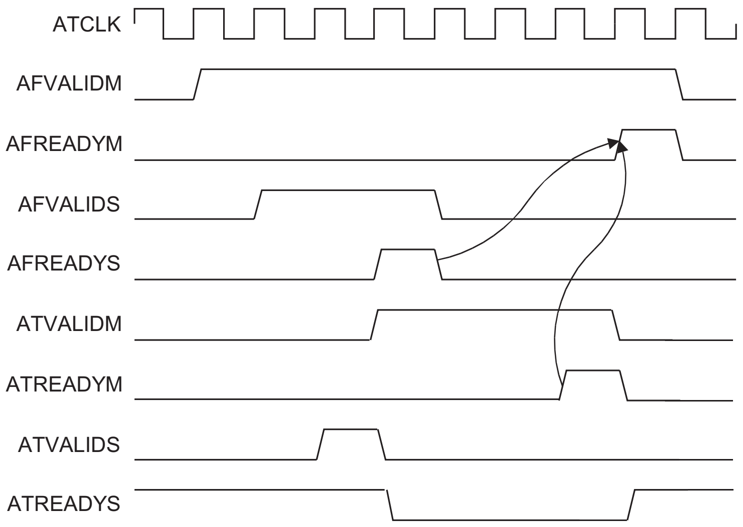


Figure 6-3 ATB flushing data packets

6.2 Authentication requirements for ATB 1:1 Bridge

No disable inputs are required because this component has no effect on the ATB data.

Chapter 7
ATB Replicator

This chapter describes the *AMBA Trace Bus* (ATB) replicator. The ATB replicator enables two trace sinks to be wired together and operate from the same incoming trace stream. It contains the following sections:

7.1 About the ATB replicator

The ATB replicator enables two trace sinks to be wired together and to operate from the same incoming trace stream. There are no programmable registers. This component is invisible to the user on a particular trace path, from source to sink. Figure 7-1 shows the example ATB replicator.

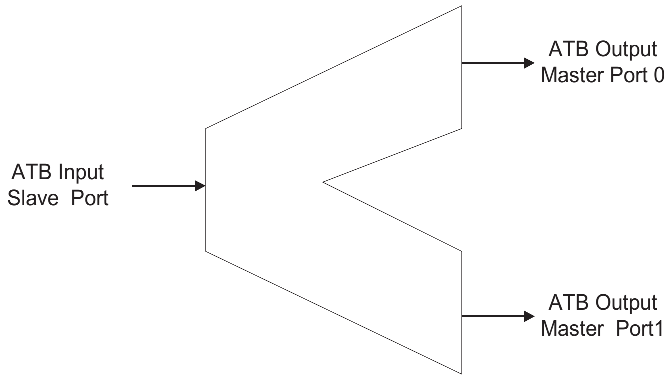


Figure 7-1 Example ATB replicator

7.1.1 Incoming ATB interface

The ATB replicator accepts trace data from a trace source, either directly or through a trace funnel.

7.1.2 Outgoing ATB interfaces

The ATB replicator sends identical trace data on outgoing master port interfaces.

7.2 ATB replicator connection behavior

The ATB replicator is a rewiring unit for the connection of two trace sinks. Buses cannot be connected directly together.

The only design considerations are for:

- **ATREADY** to the incoming ATB
- **ATVALID** to the outgoing ATB
- **AFVALID** to the incoming ATB.

Because of the simplicity of the block, no programmer's model exists for this block. It is a requirement of the attached trace sinks to set their **ATREADY** signal when they are disabled. This enables the replicator to operate as a pass-through to the remaining enable trace sink.

Any connected blocks, on both the inputs and the outputs, must operate on the same ATCLK domain.

7.2.1 ATREADYS and ATVALIDM

The exported **ATREADYS** to the original trace source can only be asserted when both of the trace outputs have accepted the current packet. One or both connected trace sinks might be disabled. If so, they must tie **ATREADYS** HIGH.

Because the trace sinks accept the trace data at different moments, the **ATVALIDM** signal exported to them is not asserted immediately after they have read the data packet because the data packet must persist for the other trace sink. The **ATVALIDM** signal is valid only for new packets until a valid **ATREADYM** cycle is seen. If **ATVALIDM** is incorrectly asserted when an old data value is still present then this might cause a packet duplication to appear in the trace stream for that source. This is highly likely to create problems at decompression.

Devices that connect to the output of the Replicator, the TPIU and the ETB for example, must tie the **ATREADYM** signal HIGH when they are disabled. If connected devices do not do this they cause the trace stream to stop the other channel that might still be enabled. This removes the requirement for the replicator to be aware of the state of downstream components

7.2.2 Flushing AFVALIDM and AFREADYM

Whenever one of the trace sinks initiates a flush to remove old information from the system then the replicator must propagate this request even when the other sink has not requested it.

This does not cause any problems with the non-requested trace sink, it just receives the flushed data. If the second master port receives a request to start a flush when there is already a flush operation under way that was begun by the first master port, then the Replicator must wait until the first request is serviced and then hold **AFVALIDS** HIGH to service the second request.

Figure 7-2 shows the expected interactions of the three **AFVALID** and **AFREADY** signal pairs. A flush request is started on master port 0 at time t0 which immediately propagates to the slave port interface, **AFVALIDM0** to **AFVALIDS**. While this is still being serviced a second request is made, at time t1, but on master port 1, **AFVALIDM1**. At t2, the first request is completed by the indication of **ATREADYS** on the slave port. This causes **AFREADYM0** to complete the flush request. Because of the second request being made, the Replicator holds **AFVALIDS** HIGH to initiate a second flush of the system which continues until **AFREADYS** goes HIGH a second time at time t3.

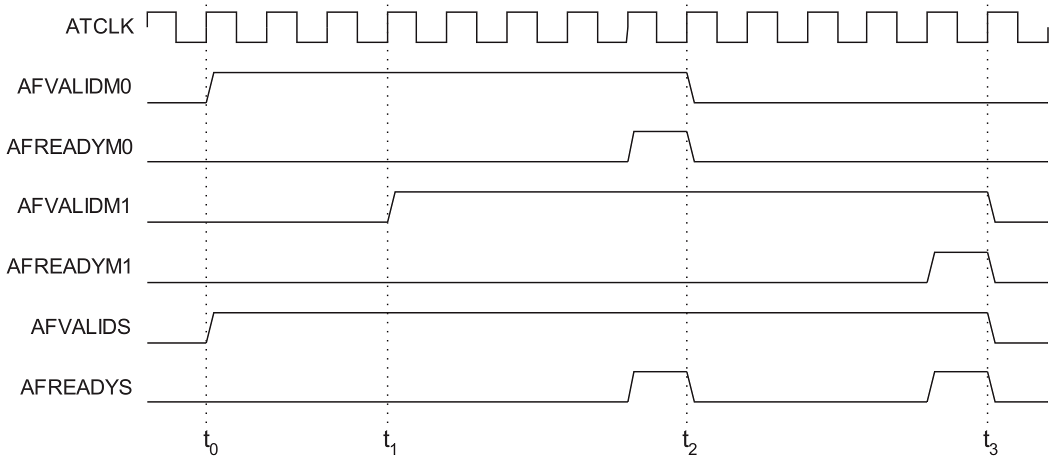


Figure 7-2 ATB replicator flushing behavior

7.3 Authentication requirements for replicators

ATB replicators do not require any inputs capable of disabling them.

Chapter 8
CoreSight Trace Funnel

This chapter describes the *CoreSight Trace Funnel* (CSTF). It contains the following sections:

8.1 About the CoreSight Trace Funnel

The CSTF is used when there is more than one trace source. The CSTF combines multiple trace streams onto a single ATB bus.

8.1.1 CSTF blocks

Figure 8-1 shows the blocks in the CSTF.

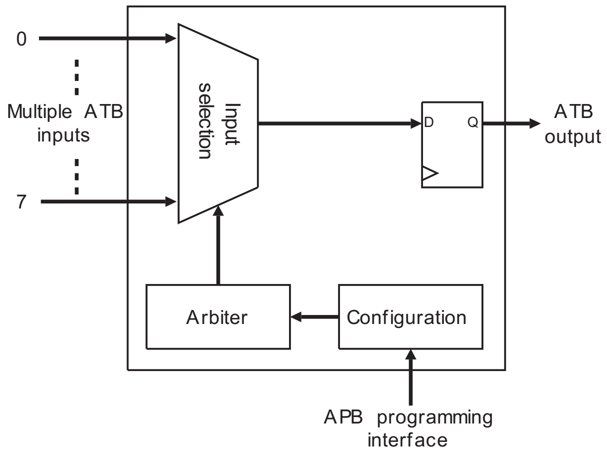


Figure 8-1 CSTF block diagram

The CSTF contains the following blocks:

Input selection multiplexor

In CoreSight this is a fixed input of eight ports.

Arbiter The arbiter determines the priority of the ATB inputs. The CoreSight arbiter operates a static priority scheme. The highest priority input is selected and after a fixed hold time the input with the next highest priority is selected, if active.

Configuration

Control registers for the programming interface.

8.1.2 Debug APB interface

The Debug APB interface is the only programming interface.

8.2 CSTF programmer's model

Table 8-1 shows the CSTF registers.

Table 8-1 CSTF visible registers

Offset	Type	Width	Reset value	Name	Description
`0x000`	R/W	12	`0x300`	Funnel Control Register	*CSTF Control Register, 0x000* on page 8-5
`0x004`	R/W	24	`0xFAC688`	Priority Control Register	*CSTF Priority Control Register, 0x004* on page 8-6
`0xEEC`	R/W	5	`0x00`	Integration Register, ITATBDATA0	*CSTF Integration Test Registers* on page 8-8
`0xEF0`	R/W	2	`0x0`	Integration Register, ITATBCTR2	*CSTF Integration Test Registers* on page 8-8
`0xEF4`	R/W	7	`0x00`	Integration Register, ITATBCTR1	*CSTF Integration Test Registers* on page 8-8
`0xEF8`	R/W	10	`0x000`	Integration Register, ITATBCTR0	*CSTF Integration Test Registers* on page 8-8
`0xF00`	R/W	1	`0x0`	Integration Mode Control Register	*CoreSight Management Registers summary* on page 2-12
`0xFA0`	R/W	4	`0xF`	Claim Tag Set Register	
`0xFA4`	R/W	4	`0x0`	Claim Tag Clear Register	
`0xFB0`	WO	32	-	Lock Access	
`0xFB4`	RO	3	`0x0`	Lock Status	
`0xFB8`	RO	8	`0x00`	Authentication status	*CoreSight management registers for CSTF* on page 8-14
`0xFC8`	RO	8	`0x28`	Device ID	*CoreSight management registers for CSTF* on page 8-14
`0xFCC`	RO	8	`0x12`	Device Type Identifier	*CoreSight management registers for CSTF* on page 8-14

Table 8-1 CSTF visible registers (continued)

Offset	Type	Width	Reset value	Name	Description
`0xFD0`	RO	8	`0x04`	Peripheral ID4	*Peripheral Identification Registers, 0xFD0-0xFEC* on page 2-5
`0xFD4`	RO	8	`0x00`	Peripheral ID5	
`0xFD8`	RO	8	`0x00`	Peripheral ID6	
`0xFDC`	RO	8	`0x00`	Peripheral ID7	
`0xFE0`	RO	8	`0x08`	Peripheral ID0	
`0xFE4`	RO	8	`0xB9`	Peripheral ID1	
`0xFE8`	RO	8	`0x0B`	Peripheral ID2	
`0xFEC`	RO	8	`0x00`	Peripheral ID3	
`0xFF0`	RO	8	`0x0D`	Component ID0	
`0xFF4`	RO	8	`0x90`	Component ID1	
`0xFF8`	RO	8	`0x05`	Component ID2	
`0xFFC`	RO	8	`0xB1`	Component ID3	

8.3 CSTF specific registers

This section describes the CSTF specific registers:

- *CSTF Control Register, 0x000*
- *CSTF Priority Control Register, 0x004* on page 8-6.

The CSTF Control Register and the CSTF Priority Control Register can only be changed when the whole trace system is stopped.

8.3.1 CSTF Control Register, 0x000

The CSTF Control Register enables the slave ports and defines the hold time of the slave ports.Hold time refers to the number of transactions that are output on the funnel master port from the same slave while that slave port **ATVALIDSx** is HIGH. Hold time does not refer to clock cycles in this context. Figure 8-2 shows the bit assignments.

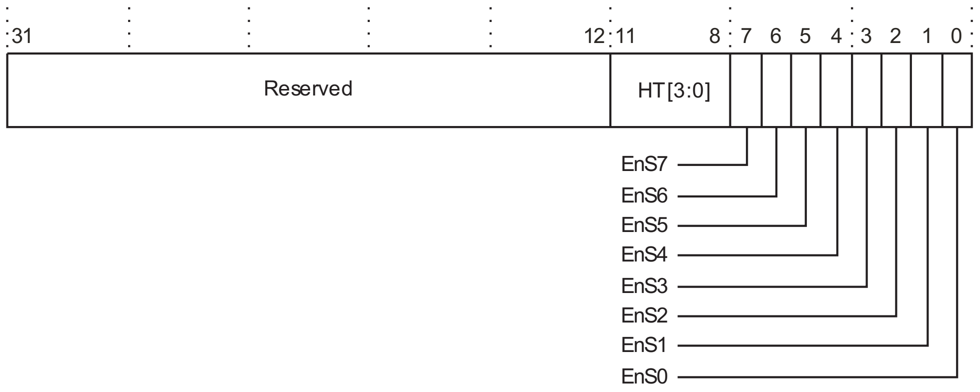


Figure 8-2 CSTF Control Register bit assignments

Table 8-2 shows the bit assignments.

Table 8-2 CSTF Control Register bit assignments

Bits	Field name	Description
[31:12]	-	Reserved SBZ
[11:8]	Minimum hold time[3:0]	The formatting scheme can easily become inefficient if fast switching occurs, so, where possible, this must be minimized. If a source has nothing to transmit, then another source is selected irrespective of the minimum number of cycles. Reset is `0x3`. The CSTF holds for the minimum hold time and one additional cycle. The maximum value that can be entered is `0xE` and this equates to 15 cycles. `0xF` is reserved.
[7]	Enable Slave port 7	Setting this bit enables this input, or slave, port. If the bit is not set then this has the effect of excluding the port from the priority selection scheme. The reset value is all clear, that is, all ports disabled.
[6]	Enable Slave port 6	
[5]	Enable Slave port 5	
[4]	Enable Slave port 4	
[3]	Enable Slave port 3	
[2]	Enable Slave port 2	
[1]	Enable Slave port 1	
[0]	Enable Slave port 0	

See the Data Overhead section in the *CoreSight System Design Guide* for more information.

8.3.2 CSTF Priority Control Register, 0x004

The CSTF Priority Control Register defines the order in which inputs are selected. Each 3-bit field represents a priority for each particular slave interface. Location 0 has the priority value for the first slave port. Location 1 is the priority value for the second slave port, Location 2 is the third, down to location 7 which has the priority value of the eighth slave port. Values represent the priority value for each port number. Figure 8-3 shows the bit assignments.

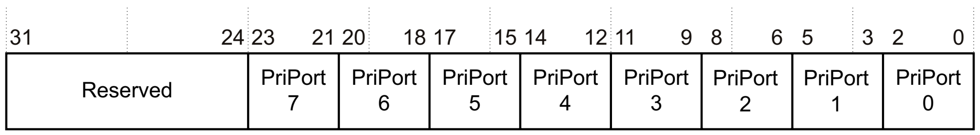


Figure 8-3 CSTF Priority Control Register bit assignments

Table 8-3 shows the bit assignments.

Table 8-3 CSTF Priority Control Register bit assignments

Bits	Field name	Description
[31:24]	-	Reserved.
[23:21]	PriPort 7	Priority value of the eighth port. The value written into this location is the value that you want to assign the eighth slave port.
[20:18]	PriPort 6	7th port priority value.
[17:15]	PriPort 5	6th port priority value.
[14:12]	PriPort 4	5th port priority value.
[11:9]	PriPort 3	4th port priority value.
[8:6]	PriPort 2	3rd port priority value.
[5:3]	PriPort 1	2nd port priority value.
[2:0]	PriPort 0	Priority value of the first slave port. The value written into this location is the value that you want to assign the first slave port.

At reset the default configuration assigns priority 0 to port 0, to priority 1 to port 1, and so on.

If you want to give highest priority to a particular slave port, the corresponding port must be programmed with the lowest value. Typically this is likely to be a port that has more important data or that has a small FIFO and is therefore likely to overflow.

If you want to give lowest priority to a particular slave port, the corresponding slave port must be programmed with the highest value. Typically this is likely to be a device that has a large FIFO that is less likely to overflow or a source that has information that is of lower importance.

A port programmed with value 0 gets the highest priority. A port programmed with value 7 gets the lowest priority.Priority must always go to the highest priority source that has valid data available, if enabled. If a priority value has been entered for multiple different slave ports then the arbitration logic selects the lowest port number of them.

Note

This register must only be altered when the trace system is disabled, that is, trace sources are off and the system is drained.

8.4 CSTF Integration Test Registers

Integration Test Registers are provided to simplify the process of verifying the integration of the CSTF with other devices in a CoreSight system. These registers enable direct control of outputs and the ability to read the value of inputs. You must only use these registers when the Integration Test Control Register (`0xF00`) bit [0] is set to 1.

There must be no trace in the system when you use the Integration registers. You use the same integration registers to check all the ATB slave interfaces in addition to the ATB master interface. To select which slave interface is connected to the integration registers, you must select the interface using the CSTF Control register at `0x000`. For integration purposes you must only enable one port at any time. When checking the ATB master port, the value of the CSTF Control Register has no effect.

For example, to read the **ATVALIDS1** input the CSTF Control Register must be set to `0x02` to only select slave port 1, then a read of ITATBCTR0 bit 0 yields the value of **ATVALIDS1**.

To set the value of **ATVALIDM**, the value in the CSTF Control Register is irrelevant and a write to ITATBCTR0 bit 0 sets the value of **ATVALIDM**.

For full details of how to use the integration registers in a system to verify the integration of multiple devices see the *CoreSight Integration and Implementation Manual*.

8.4.1 CSTF Integration Test Registers

This sections describes the following registers:

- *Integration Test ATB Data 0 Register, ITATBDATA0, 0xEEC*
- *Integration Test ATB Control 2 Register, ITATBCTR2, 0xEF0* on page 8-10
- *Integration Test ATB Control 1 Register, ITATBCTR1, 0xEF4* on page 8-11
- *Integration Test ATB Control 0 Register, ITATBCTR0, 0xEF8* on page 8-12.

Integration Test ATB Data 0 Register, ITATBDATA0, 0xEEC

The Integration Test ATB Data 0 Register performs different functions depending on whether the access is a read or a write:

- A write outputs data on **ATDATAM**.
- A read returns the data from **ATDATAS<n>**, where <n> is defined by the status of the CSTF Control register at `0x000`. The read data is only valid when **ATVALIDS<n>** is HIGH.

Figure 8-4 shows the bit assignments.

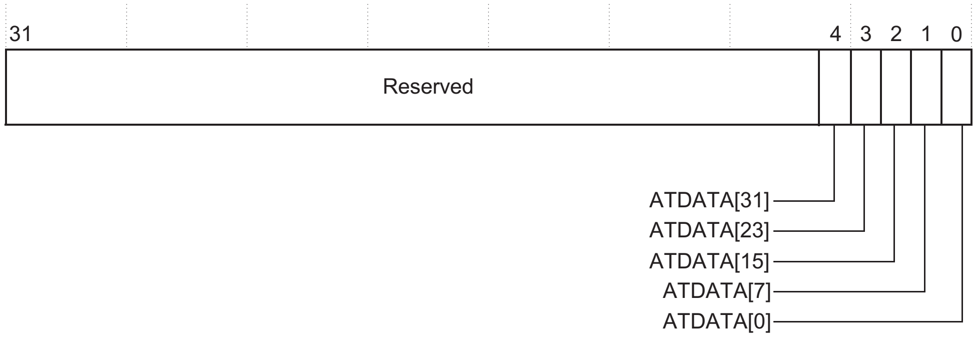


Figure 8-4 Integration Test ATB Data 0 Register bit assignments

Table 8-4 shows the bit assignments for this register on reads.

Table 8-4 Integration Test ATB Data 0 Register bit assignments on reads

Bits	Type	Name	Function
[31:5]	-	-	Reserved
[4]	RO	ATDATA[31]	Read the value of **ATDATAS<31>**
[3]	RO	ATDATA[23]	Read the value of **ATDATAS<23>**
[2]	RO	ATDATA[15]	Read the value of **ATDATAS<15>**
[1]	RO	ATDATA[7]	Read the value of **ATDATAS<7>**
[0]	RO	ATDATA[0]	Read the value of **ATDATAS<0>**

Table 8-5 shows the bit assignments for this register on writes.

Table 8-5 Integration Test ATB Data 0 Register bit assignments on writes

Bits	Type	Name	Function
[31:5]	-	-	Reserved
[4]	WO	ATDATA[31]	Set the value of **ATDATAM[31]**
[3]	WO	ATDATA[23]	Set the value of **ATDATAM[23]**
[2]	WO	ATDATA[15]	Set the value of **ATDATAM[15]**
[1]	WO	ATDATA[7]	Set the value of **ATDATAM[7]**
[0]	WO	ATDATA[0]	Set the value of **ATDATAM[0]**

Integration Test ATB Control 2 Register, ITATBCTR2, 0xEF0

The Integration Test ATB Control 2 Register performs different functions depending on whether the access is a read or a write:

- A write outputs data on **ATREADYS<n>** and **AFVALIDS<n>**, where <n> is defined by the status of the CSTF Control Register at `0x000`
- A read returns the data from **ATREADYM** and **AFVALIDM**.

Figure 8-5 shows the bit assignments.

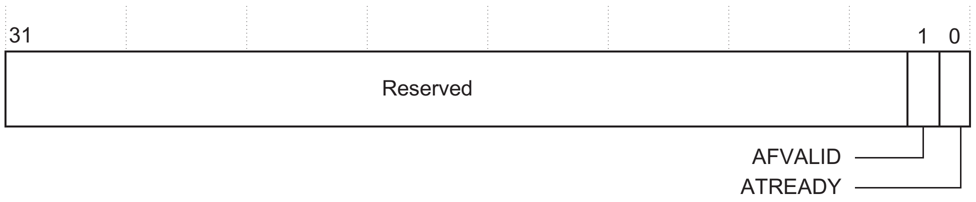


Figure 8-5 Integration Test ATB Control 2 Register bit assignments

Table 8-6 shows the bit assignments on reads.

Table 8-6 Integration Test ATB Control 2 Register bit assignments on reads

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	RO	AFVALID	Read the value of **AFVALIDM**
[0]	RO	ATREADY	Read the value of **ATREADYM**

Table 8-7 shows the bit assignments on writes.

Table 8-7 Integration Test ATB Control 2 Register bit assignments on writes

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	WO	AFVALID	Set the value of **AFVALIDS<n>**
[0]	WO	ATREADY	Set the value of **AFREADYS<n>**

Integration Test ATB Control 1 Register, ITATBCTR1, 0xEF4

The Integration Test ATB Control 1 Register performs different functions depending on whether the access is a read or a write:

- A write outputs data on **ATIDM**.
- A read returns the data from **ATIDS<n>**, where <n> is defined by the status of the CSTF Control register at `0x000`. The read data is only valid when **ATVALIDS<n>** is HIGH.

ITATBCTR1 contains the value of the **ATIDS** input to the CSTF. Figure 8-6 shows the bit assignments.

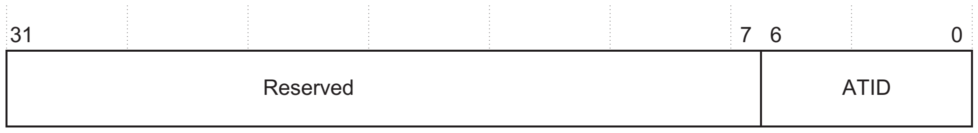


Figure 8-6 Integration Test ATB Control 1 Register bit assignments

Table 8-8 shows the bit assignments on reads.

Table 8-8 Integration Test ATB Control 1 Register bit assignments on reads

Bits	Type	Name	Function
[31:7]	-	-	Reserved
[6:0]	RO	ATID	Read the value of **ATIDS**

Table 8-9 shows the bit assignments on writes.

Table 8-9 Integration Test ATB Control 1 Register bit assignments on writes

Bits	Type	Name	Function
[31:7]	-	-	Reserved
[6:0]	WO	ATID	Set the value of **ATIDM**

Integration Test ATB Control 0 Register, ITATBCTR0, 0xEF8

The Integration Test ATB Control 0 Register performs different functions depending on whether the access is a read or a write:

- a write sets the value of the **ATVALIDM**, **AFREADYM**, and **ATBYTESM** signals.
- a read returns the value of the **ATVALIDS<n>**, **AFREADYS<n>**, and **ATBYTESS<n>** signals, where <n> is defined by the status of the CSTF Control register at `0x000`. The read value of **ATBYTESS<n>** is only valid when **ATVALIDS<n>** is HIGH.

Figure 8-7 shows the bit assignments.

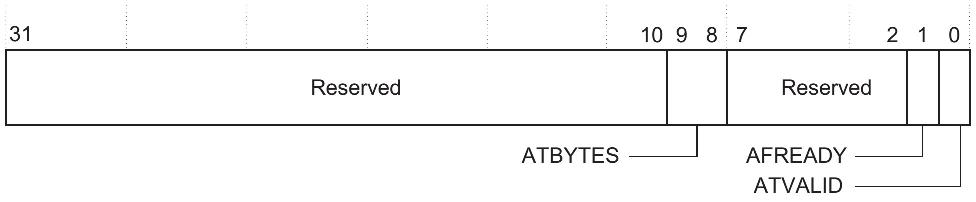


Figure 8-7 Integration Test ATB Control 0 Register bit assignments

Table 8-10 shows the bit assignments for this register on reads.

Table 8-10 Integration Test ATB Control 1 Register bit assignments on reads

Bits	Type	Name	Function
[31:10]	-	-	Reserved
[9:8]	RO	ATBYTES	Read the value of **ATBYTESS<n>**
[7:2]	-	-	Reserved
[1]	RO	AFREADYS	Read the value of **AFREADYS<n>**
[0]	RO	ATVALID	Read the value of **ATVALIDS<n>**

Table 8-11 shows the bit assignments for this register on writes.

Table 8-11 Integration Test ATB Control 0 Register bit assignments on writes

Bits	Type	Name	Function
[31:10]	-	-	Reserved
[9:8]	WO	ATBYTES	Set the value of **ATBYTESM**
[7:2]	-	-	Reserved
[1]	WO	AFREADYS	Set the value of **AFREADYM**
[0]	WO	ATVALID	Set the value of **ATVALIDM**

8.5 CoreSight management registers for CSTF

The definitions of these registers are in Chapter 2 *CoreSight Programmer's Model*. This section gives information specific to the CSTF programmable registers.

Authentication Status, 0xFB8 [7:0]

Reports the required security level. This is set to `0x00` for functionality not implemented.

Device ID, 0xFC8 The CSTF has the default value of `0x28`. Table 8-12 shows the Device ID bit assignments.

Table 8-12 CSTF Device ID bit assignments

Bits	Value	Description
[31:8]	`0x000000`	Reserved.
[7:4]	`0x2`	The CSTF implements a static priority scheme.
[3:0]	`0x8`	This is the value of the Verilog define PORTCOUNT and represents the number of input ports connected. By default all 8 ports are connected. `0x0` and `0x1` are illegal values.

Device Type Identifier, 0xFCC [7:0]

A value of `0x12` identifies this device as a trace link (`0x2`) and specifically as a funnel/router (`0x1`).

PartNumber, 0xFE4 [3:0], 0xFE0 [7:4], 0xFE0 [3:0]

Upper, middle, and lower BCD value of Device number. Set to `0x908`.

8.6 Unconnected slave interfaces

The CSTF is fixed as an eight-input device. Not all eight inputs might be required. When unused ports are present these must be suitably tied off as shown in Table 8-13. If ports are unconnected but not disabled, the operation is UNPREDICTABLE.

Table 8-13 Tie-offs for unconnected ports

Port name	Tie-off value
ATIDSx[6:0]	None required but a recommendation of 7'b0000000
ATDATASx[31:0]	None required but a recommendation of 32'h00000000
ATVALIDSx	1'b0
ATBYTESSx[1:0]	None required but a recommendation of 2'b00
ATREADYSx	None required because this is an output
AFVALIDSx	None required because this is an output
AFREADYSx	1'b1

8.7 Disabled slave interfaces

When a slave interface is disabled, that is, the corresponding bit is cleared in the Funnel Control Register, the interface returns **ATREADYx** HIGH and **AFALIDx** LOW.

8.8 CSTF input arbitration

Typical arbiters in AHB systems use a simple fixed priority scheme. This reduces the required complexity of the arbiter and also removes any direct user control in programmable registers. To ensure maximum flexibility is obtained before intended use is decided, the arbitration scheme used in the CSTF is more flexible than the simple fixed priority.

8.8.1 Static priority

Static priority is a small expansion of the standard fixed priority scheme. Rather than dictate the priorities of the inputs at the design stage, this can be altered before any trace capture is begun to enable refinement of input priority ordering for different requirements. The advantage of this scheme compared to the fixed version is that the integrator does not have to know trace usage requirements which might change depending on the particular application.

8.8.2 Minimum hold time

The Funnel Control Register enables slave inputs of the funnel to be held for a number of cycles before the arbiter logic changes to a higher priority source. This value is the minimum number of cycles of data that are output from a source before a change to a higher priority is performed, plus 1, that is, a value of `0x0` indicates a minimum hold time of 1 and a value of `0x5`, indicates a minimum hold time of six cycles. Large values enable the formatter in an ETB or TPIU to be less wasteful by having to indicate source changes in the protocol. These source changes reduce the amount of bandwidth available for trace data.

Figure 8-8 on page 8-18 shows an example of how a minimum hold time of two cycles stops the change of the arbiter from Slave1, **ATVALIDS1** and **ATREADYS1**, even though a higher priority source indicates valid data, **ATVALIDS0**, after t0.

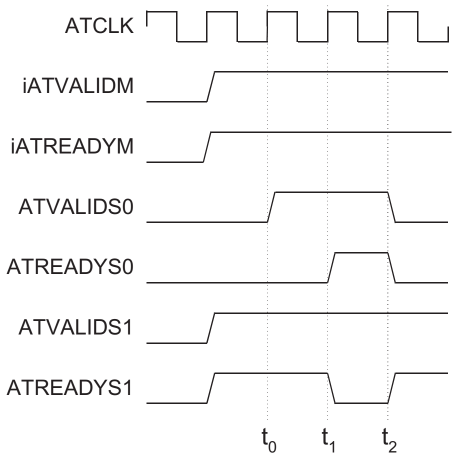


Figure 8-8 Funnel minimum hold time, two cycles

At point t1, Slave port 1 has performed the minimum of two cycles and the arbitration logic then switches to port 0, even though port 1 has more data valid. Finally at t2 there is no more data available from the higher priority Slave 0 and so the arbitration logic switches back to port 1.

Note

Figure 8-8 shows the internal Master port signals which have a one cycle difference to the final output Master port. These are indicated as **iATVALIDM** and **iATREADYM**, and they are the signals that are presented before the output register stage.

Figure 8-9 on page 8-19 is similar to Figure 8-8 but the trace sink, attached to the Trace Funnel master port, holds off fetching the second packet from Slave port 1 by de-asserting **ATREADYM** after t0. The minimum hold time is finally achieved by t1 where the arbiter selects Slave port 0.

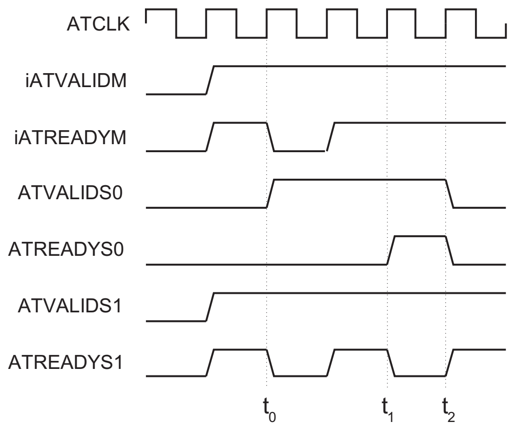


Figure 8-9 Minimum hold time, two cycles with wait

8.8.3 Initial register programming

Before any trace generation begins, you must program the Priority Control Register with the different priority levels of the slave ports. Program the port you want to assign the highest priority with the lowest value, 3’b000, the second highest priority port with the value 3’b001, down to the port with the lowest priority, with the value 3’b111.

Program any port inputs that are disabled with the remaining low priority values. The exact order is not important, because these registers can be disabled in the Funnel Control Register. In addition, you can adjust the Minimum Hold Time setting in the Funnel Control Register, depending on the dynamics of your system.

8.8.4 Operation example

Figure 8-10 on page 8-20 shows a possible sequence of events of the ports on the Trace Funnel and the multiplexor selection. This example also assumes the only four trace sources are connected and that the priority encoding follows the reset/default priority, highest priority to slave port 0, lowest to slave port 3. The Minimum HoldTime Register is set-up to hold sources for two cycles before changing to a higher priority source.

Up to time marker t0 the lowest priority source is selected, slave port 3, because it always has data ready to be emitted. The trace sink, connected to the Trace Funnel master port, is stalling and so slowing the draining of the trace sources.

At point t0, a higher priority source indicates valid data is available, **ATVALIDS2** goes HIGH, so the arbitration unit selects slave 2 because the minimum hold time for slave 3 is achieved.

At point t1, slave 1 asserts **ATVALIDS1** and no higher priority sources indicate valid data so the arbiter switches to slave 1. Even though slave 2 has valid data, the arbiter unit switches to slave 1 because the minimum hold time for slave 2 is already achieved.

At point t2, slave 0 asserts **ATVALIDS0** indicating valid data. Because the minimum hold time is not achieved by slave 1 at this point, the arbiter unit does not switch to slave port 0.

At point t3, the arbiter unit switches to slave port 0 even though slave port 1 has valid data because the minimum hold time is achieved by slave port 1.At point t4, the arbiter unit switches to slave port 1 because the slave port 0 deasserts **ATVALIDS0**.

At point t5, the arbiter unit switches back to slave port 2 because the slave port 1 deasserts **ATVALIDS1**.

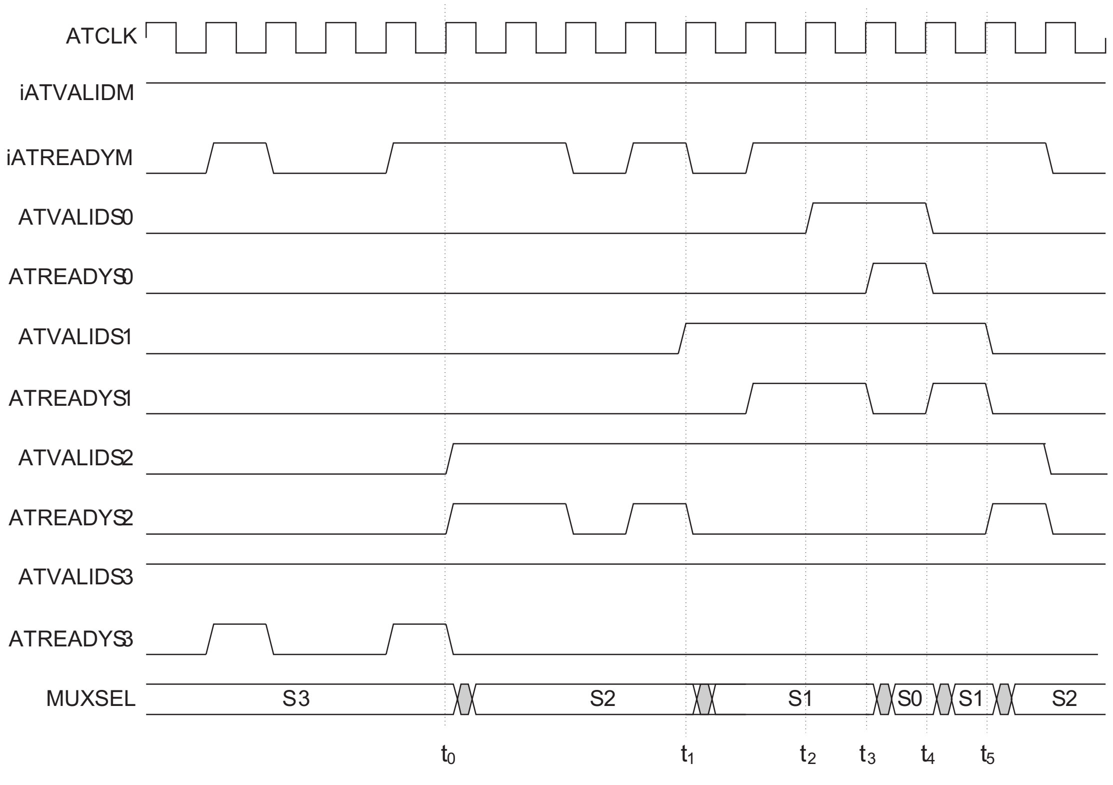


Figure 8-10 Example operation with four trace sources

8.8.5 Flushing

Flushing is the emptying of existing data from the trace source buffers before tracing new data begins. See the *CoreSight Architecture Specification* for more on flushing mechanisms.At all times during a flushing operation, all ATB signals must comply with the ATB protocol.

The behavior of the CSTF is that **AFREADYM** only responds HIGH when all Slave ports have returned **AFREADYS** HIGH.

On assertion of **AFVALIDM** to the CSTF, all the enabled slave ports should have their **AFVALIDS** signals asserted to ensure that the marker for flushing historical information is located at similar times. **AFVALIDS** should then remain asserted until the specific slave port responds with **AFREADYS** HIGH. **AFVALIDS** is deasserted on a port by port basis.

The CSTF flushing mechanism uses the funnel prioritization for flushing historical data from components connected to the funnel inputs.

When **AFVALIDS** is output HIGH on all CSTF slave ports, the first source to indicate that it has valid data, indicated by **ATVALIDS** HIGH, is accepted.

If multiple sources assert **ATVALIDS** HIGH on the same cycle, then the highest priority source is selected. When this source completes other sources can be serviced.

When the flush is complete for a particular trace source, it is ignored in the input selection priority selection scheme until all enabled inputs have completed the flush operation by returning **AFREADYS** HIGH. This occurs even if it might have more data available, that is, **ATVALID** is HIGH.

8.8.6 Flushing example

Figure 8-11 on page 8-23 has four sources connected to the first four slave ports and they are configured so that they are enabled with priorities reflecting their port number Slave port 0 is the highest priority, and port 3 is the lowest.

Before time t0, the system is under normal operation where different trace sources are selected according to the arbitration scheme, in this example slave port 2. At time t0 there is a request from the trace sink device to flush the system of historical data this request is then propagated onto the slave ports in the following cycle, at time t1. The CSTF, at this point, is then operating in a flushing mode where it selects each device that has not been drained in turn.

At time t1, CSTF selects slave port 1, highest priority with valid data, to drain first. At the same time the source connected to port 3 returns **AFREADYS3** which causes **AFVALIDS3** to be deasserted in the following cycle.

Draining of slave port 1 continues until time t2 where it responds with **AFREADYS1**. If all higher priority sources are already flushed, the CSTF then switches to the next highest priority source that is still to drain. In this example, slave port 2 is not selected. It is overridden because a higher priority port reports valid data which has not finished flushing, therefore slave port 0 is selected.

This continues until time t3 when **AFREADYS0** goes HIGH and draining of slave port 2 can begin and which continues until time t4, **AFREADYS2** going HIGH.

At time t4 all slave ports have responded with **AFREADY** HIGH. This indicates that there is no more historical information remaining at this level. This is indicated on **AFREADYM** with a HIGH response, which in turn results in **AFVALIDM** returning LOW. The flush sequence is now complete and the CSTF can return to normal operation, in this example by selecting slave port 1.

Slave port 1 is not selected for a second time before time t4 even though it is at a higher priority than slave port 2. This is because it has returned **AFREADYS1** HIGH previously.

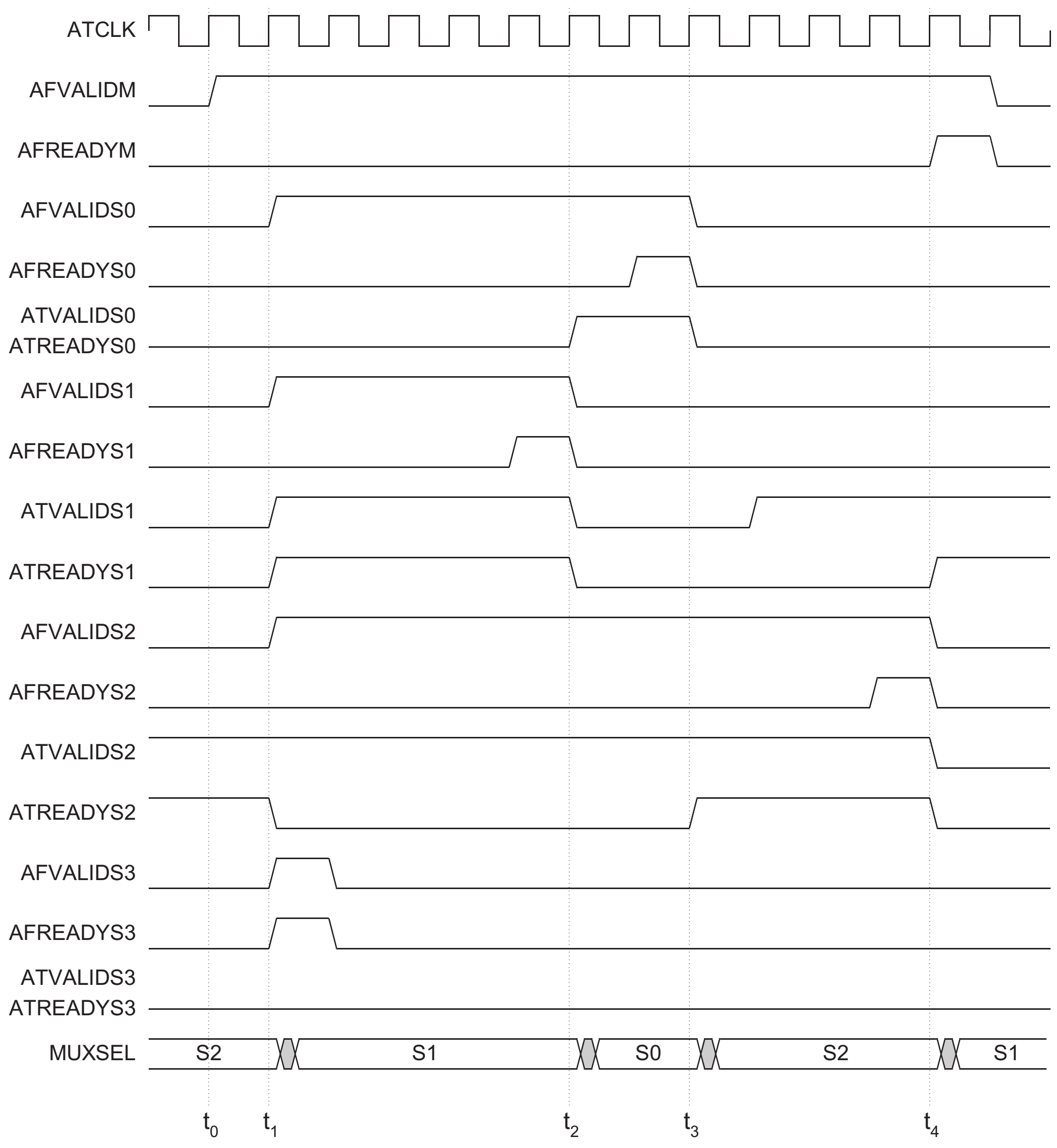


Figure 8-11 Flushing operation with four trace sources

8.9 Authentication requirements for funnels

Funnels do not require any inputs capable of disabling them.

Chapter 9
Trace Port Interface Unit

This chapter describes the *Trace Port Interface Unit* (TPIU). It contains the following sections:

9.1 About the Trace Port Interface Unit

The TPIU acts as a bridge between the on-chip trace data, with separate IDs, to a data stream, encapsulating IDs where required, that is then captured by a *Trace Port Analyzer* (TPA). Figure 9-1 shows the main blocks of the TPIU and the clock domains.

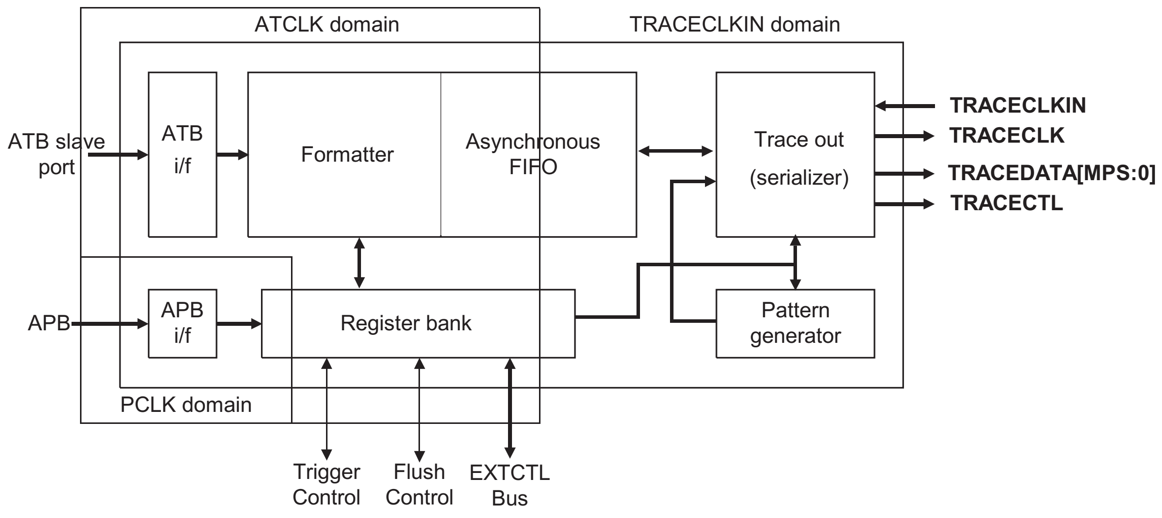


Figure 9-1 TPIU block diagram

The behavior of the blocks is as follows:

Formatter Inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source. See *TPIU formatter and FIFO* on page 9-34.

Asynchronous FIFO

Enables trace data to be driven out at a speed that is not dependent on the on-chip bus clock.

Register bank

Contains the management, control and status registers for triggers, flushing behavior and external control.

Trace out The trace out block serializes formatted data before it goes off-chip.

Pattern Generator

The pattern generator unit provides a simple set of defined bit sequences or patterns that can be output over the Trace Port and be detected by the TPA or other associated *Trace Capture Device* (TCD). The TCD can use these patterns to indicate if it is possible to increase or to decrease the trace port clock speed. See *TPIU pattern generator* on page 9-32 for more information.

9.1.1 ATB interface

The TPIU accepts trace data from a trace source, either direct from a trace source or using a Trace Funnel.

9.1.2 APB interface

The APB interface is the programming interface for the TPIU.

9.2 Trace Out Port

Table 9-1 shows the Trace Out Port signals.

Table 9-1 Trace Out Port signals

Name	Type	Description
TRACECLKIN	Input	Decoupled clock from ATB to allow easy control of the trace port speed. This is typically derived from a controllable clock source on chip but could be driven by an external clock generator if a high speed pin is used. Data changes on the rising edge only. See *Off-chip based TRACECLKIN* on page 9-30 for more details of off-chip operated **TRACECLKIN**.
TRACECLK	Output	Exported version of **TRACECLKIN**. This is **TRACECLKIN**/2, and data changes on both rising edges and falling edges. See *TRACECLK generation* on page 9-27 for more details about **TRACECLK** generation.
TRACEDATA[MPS:0]	Output	Output data. MPS is **TPMAXDATASIZE**. See *Supported Port Size Register, 0x000* on page 9-10.
TRACECTL	Output	Used to indicate nonvalid trace data and triggers. See *Other TPIU design considerations* on page 9-27.
TRESETn	Input	This is a reset signal for the TRACECLKIN domain. Because off-chip devices connect to the Trace Out port, this signal is related to the Trace Bus Resetting signal, **ATRESETn**.
TPCTL	Input	ASIC tie-off to report the presence of the **TRACECTL** pin. If **TRACECTL** is not present then this must be tied LOW. This input affects bit 2 of the Formatter and Flush Status Register. See *Formatter and Flush Status Register, 0x300* on page 9-15.
TPMAXDATASIZE[4:0]	Input	Tie-off to indicate the maximum TRACEDATA width available on the ASIC. The valid values are 1-32 (`0x00-0x1F`), for example if only a maximum of a 16-bit data port is available then this takes the value `0x0F`. This input affects the Supported Port Size Register *Supported Port Size Register, 0x000* on page 9-10.

9.3 Miscellaneous connections

These ports supplement the operation of the TPIU and are for the connection of other CoreSight components or other ASIC blocks. Table 9-2 shows the ports not described elsewhere.

Table 9-2 TPIU miscellaneous ports

Name	Type	Description
TRIGIN	Input	From either a CTI or direct from a trace source. Used to enable the trigger to affect the output trace stream.
TRIGINACK	Output	Return response to the CTI acknowledgement to **TRIGIN**.
FLUSHIN	Input	External control used to invoke an ATB signal and drain any old historical information on the bus.
FLUSHINACK	Output	Flush response, goes HIGH to acknowledge **FLUSHIN**. If the completion of a flush is required then this can be established by the Formatter Flush and Control Register. See *Formatter and Flush Control Register, 0x304* on page 9-16.
EXTCTLOUT[7:0]	Output	Used as control signals for any configurable drivers on the output of the trace port such as serializers, output multiplexor and so on.
EXTCTLIN[7:0]	Input	Used as an input port for any configurable drivers on the output of the trace port such as serializers, output multiplexor and so on.

9.4 TPIU programmer's model

This section describes all the visible registers that can be accessed from the APB interface. Table 9-3 shows the TPIU programmable registers.

Table 9-3 TPIU programmable registers

Offset	Type	Width	Reset value	Name	Description
`0x000`	RO	32	`0xFFFFFFF`	Supported port sizes	See *Supported Port Size Register, 0x000* on page 9-10
`0x004`	R/W	32	`0x0000001`	Current port size	See *Current Port Size Register, 0x004* on page 9-11
`0x100`	RO	18	`0x11F`	Supported trigger modes	See *Supported Trigger Modes Register, 0x100* on page 9-11
`0x104`	R/W	8	`0x00`	Trigger counter value	See *Trigger Counter Register, 0x104* on page 9-12
`0x108`	R/W	5	`0x00`	Trigger multiplier	See *Trigger Multiplier Register, 0x108* on page 9-13
`0x200`	RO	18	`0x3000F`	Supported test pattern/modes	See *Supported Test Patterns/Modes Register, 0x200* on page 9-13
`0x204`	RO	18	`0x00000`	Current test pattern/mode	See *Current Test Patterns/Modes Register, 0x204* on page 9-14
`0x208`	R/W	8	`0x00`	Test pattern repeat counter	See *TPIU Test Pattern Repeat Counter Register, 0x208* on page 9-15
`0x300`	RO	3	`0x6`	Formatter and flush status	See *Formatter and Flush Status Register, 0x300* on page 9-15
`0x304`	R/W	14	`0x1000`	Formatter and flush control	See *Formatter and Flush Control Register, 0x304* on page 9-16
`0x308`	R/W	12	`0x040`	Formatter synchronization counter	See *Formatter Synchronization Counter Register, 0x308* on page 9-18
`0x400`	RO	8	Undefined	**EXTCTL** In Port	See *TPIU EXCTL Port Registers* on page 9-23
`0x404`	R/W	8	`0x00`	**EXTCTL** Out Port	See *TPIU EXCTL Port Registers* on page 9-23

Table 9-3 TPIU programmable registers (continued)

Offset	Type	Width	Reset value	Name	Description
`0xEE4`	WO	2	-	Integration Register, ITTRFLINACK	See *Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4* on page 9-19
`0xEE8`	RO	2	Undefined	Integration Register, ITTRFLIN	See *Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8* on page 9-20
`0xEEC`	RO	5	Undefined	Integration Register, ITATBDATA0	See *Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC* on page 9-20
`0xEF0`	WO	2	-	Integration Register, ITATBCTR2	See *Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 9-21
`0xEF4`	RO	7	Undefined	Integration Register, ITATBCTR1	See *Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4* on page 9-22
`0xEF8`	RO	10	Undefined	Integration Register, ITATBCTR0	See *Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 9-22
`0xF00`	R/W	1	`0x0`	Integration Mode Control Register	See *CoreSight Management Registers summary* on page 2-12
`0xFA0`	R/W	4	`0x0`	Claim Tag Clear	
`0xFA4`	R/W	4	`0xF`	Claim Tag Set	
`0xFB4`	RO	3	`0x3`	Lock status	
`0xFB0`	WO	32	-	Lock Access	
`0xFB8`	RO	8	`0x00`	Authentication status	See *TPIU CoreSight management registers* on page 9-9
`0xFC8`	RO		`0x0A0`	Device ID	
`0xFCC`	RO	8	`0x11`	Device type identifier	

Table 9-3 TPIU programmable registers (continued)

Offset	Type	Width	Reset value	Name	Description
0xFD0	RO	8	0x04	Peripheral ID4	See *CoreSight Management Registers summary* on page 2-12
0xFD4	RO	8	0xXX (unused)	Peripheral ID5	
0xFD8	RO	8	0xXX (unused)	Peripheral ID6	
0xFDC	RO	8	0xXX (unused)	Peripheral ID7	
0xFE0	RO	8	0x12	Peripheral ID0	See *TPIU CoreSight management registers* on page 9-9
0xFE4	RO	8	0xB9	Peripheral ID1	
0xFE8	RO	8	0x0B	Peripheral ID2	See *Peripheral Identification Registers, 0xFD0-0xFEC* on page 2-5
0xFEC	RO	8	0x00	Peripheral ID3	
0xFF0	RO	8	0x0D	Component ID0	
0xFF4	RO	8	0x90	Component ID1	
0xFF8	RO	8	0x05	Component ID2	
0xFFC	RO	8	0xB1	Component ID3	

9.5 TPIU CoreSight management registers

See Chapter 2 *CoreSight Programmer's Model* for the definitions of these registers.

The information given here is specific to the TPIU:

Authentication Status Register, 0xFB8

Reports the required security level. The TPIU has a default value of `0x00` to indicate that this functionality is not implemented.

Device ID, 0xFC8

The TPIU has a default value of `0x0A0`.

Table 9-4 shows the Device ID bit values.

Table 9-4 Device ID bit values

Bits	Value	Description
[31:12]	`0x00000`	Reserved.
[11]	0	Serial Wire Output (UART/NRZ) supported.
[10]	0	Serial Wire Output (Manchester) supported.
[9]	0	Trace clock + data not supported.
[8:6]	3'b010	FIFO size in powers of 2. A value of 2 gives a FIFO size of 4 entries, 16 bytes.
[5]	1'b1	Indicates the relationship between **ATCLK** and **TRACECLKIN**. `0x1` indicates asynchronous.
[4:0]	5'b0000	Hidden Level of Input multiplexing. When nonzero this value indicates the type/number of ATB multiplexing present on the input to the ATB. Currently only `0x00` is supported, that is, no multiplexing present. This value is used to assist topology detection of the ATB structure.

Device Type Identifier, 0xFCC

`0x11` indicates this device is a trace sink and specifically a TPIU.

Part number, 0xFE4[3:0], 0xFE0[7:4], and 0xFE0[3:0]

Upper, middle, and lower BCD value of Device number. This is set to `0x912`.

9.6 Trace port control registers

This section describes the trace port control registers:

9.6.1 Supported Port Size Register, 0x000

This register is read/write. Each bit location represents a single port size that is supported on the device, that is, 32-1 in bit locations [31:0]. If the bit is set then that port size is allowable. By default the RTL is designed to support all port sizes, set to `0xFFFFFFFF`. This register reflects the value of the CSTPIU_SUPPORTSIZE_VAL Verilog `define value, currently not user modifiable, and is further constrained by the input tie-off **TPMAXDATASIZE**.

The external tie-off, **TPMAXDATASIZE**, must be set during finalization of the ASIC to reflect the actual number of **TRACEDATA** signals being wired to physical pins. This is to ensure that tools do not attempt to select a port width that cannot be captured by an attached TPA. The value on **TPMAXDATASIZE** causes bits within the Supported Port Size register that represent wider widths to be clear, that is, unsupported.

Figure 9-2 shows the Supported Port Size Register bit assignments.

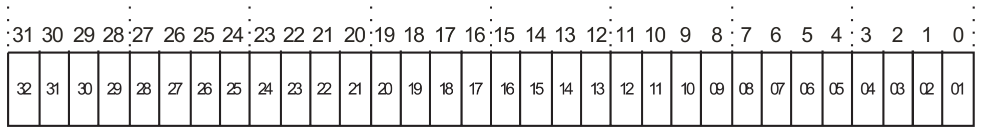


Figure 9-2 Supported Port Size Register bit assignments

9.6.2 Current Port Size Register, 0x004

This register is read/write.The Current Port Size Register has the same format as the Supported Port Sizes register but only one bit is set, and all others must be zero. Writing values with more than one bit set or setting a bit that is not indicated as supported is not supported and causes unpredictable behavior.

On reset this defaults to the smallest possible port size, 1 bit, and so reads as `0x00000001`.

9.6.3 Supported Trigger Modes Register, 0x100

The Supported Trigger Modes Register is read only. This register indicates the implemented Trigger Counter multipliers and other supported features of the trigger system. Figure 9-3 shows the Supported Trigger Modes Register bit assignments.

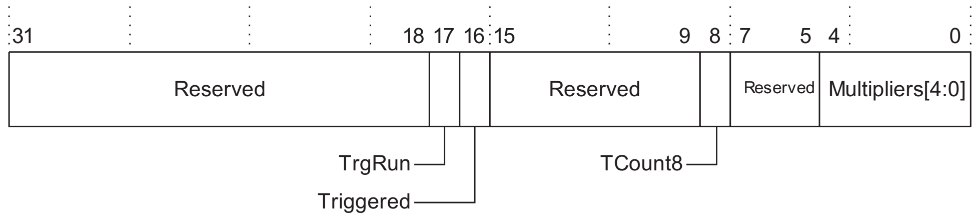


Figure 9-3 Supported Trigger Modes Register bit assignments

Table 9-5 shows the Supported Trigger Modes Register bit assignments.

Table 9-5 Supported Trigger Modes Register bit assignments

Bits	Type	Name	Description
[31:18]	-	-	Reserved RAZ/SBZP
[17]	RO	TrgRun	Trigger Counter running. A trigger has occurred but the counter is not at zero.
[16]	RO	Triggered	Triggered. A trigger has occurred and the counter has reached zero.
[15:9]	-	-	Reserved RAZ/SBZP
[8]	RO	TCount8	8-bit wide counter register implemented.
[7:5]	-	-	Reserved RAZ/SBZP
[4]	RO	Mult64k	Multiply the Trigger Counter by 65536 supported.
[3]	RO	Mult256	Multiply the Trigger Counter by 256 supported.

Table 9-5 Supported Trigger Modes Register bit assignments (continued)

Bits	Type	Name	Description
[2]	RO	Mult16	Multiply the Trigger Counter by 16 supported.
[1]	RO	Mult4	Multiply the Trigger Counter by 4 supported.
[0]	RO	Mult2	Multiply the Trigger Counter by 2 supported.

9.6.4 Trigger Counter Register, 0x104

The Trigger Counter Register enables delaying the indication of triggers to any external connected trace capture or storage devices. This counter is only eight bits wide and is intended to only be used with the counter multipliers in the Trigger Multiplier Register, `0x108`. When a trigger is started, this value, in combination with the multiplier, is the number of words before the trigger is indicated. When the trigger counter reaches zero, the value written here is reloaded. Writing to this register causes the trigger counter value to reset but not reset any values on the multiplier. Reading this register returns the preset value not the current count.

Figure 9-4 shows the Trigger Counter Register bit assignments.

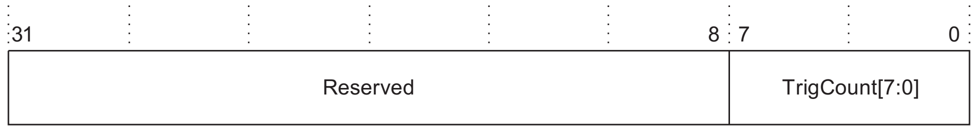


Figure 9-4 Trigger Counter Register bit assignments

Table 9-6 shows the Trigger Counter Register bit assignments.

Table 9-6 Trigger Counter Register bit assignments

Bits	Type	Name	Description
[31:8]	-	-	Reserved RAZ/SBZP
[7:0]	R/W	TrigCount	8-bit counter value for the number of words to be output from the formatter before a trigger is inserted. Reset value is `0x00`.

At reset the value is zero and this value has the affect of disabling the register, that is, there is no delay.

9.6.5 Trigger Multiplier Register, 0x108

This register contains the selectors for the Trigger Counter Multiplier. Several multipliers can be selected to create the required multiplier value, that is, any value between 1 and approximately $2x10^9$. The default value is multiplied by 1, `0x0`.

Writing to this register causes the internal trigger counter and the state in the multipliers to be reset to initial count position, that is, trigger counter is reloaded with the Trigger Counter Register value and all multipliers are reset.

Figure 9-5 shows the Trigger Multiplier Register bit assignments.

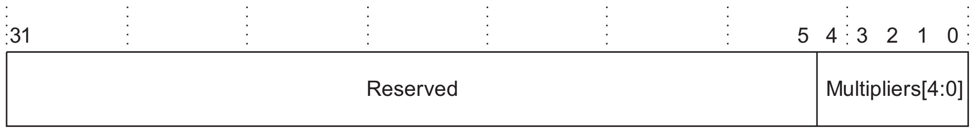


Figure 9-5 Supported Trigger Multiplier Register bit assignments

Table 9-7 shows the Trigger Multiplier Register bit assignments.

Table 9-7 Supported Trigger Multiplier Register bit assignments

Bits	Type	Name	Description
[31:5]	-	-	Reserved RAZ/SBZP
[4]	R/W	Mult64k	Multiply the Trigger Counter by 65536 (2^{16})
[3]	R/W	Mult256	Multiply the Trigger Counter by 256 (2^8)
[2]	R/W	Mult16	Multiply the Trigger Counter by 16 (2^4)
[1]	R/W	Mult4	Multiply the Trigger Counter by 4 (2^2)
[0]	R/W	Mult2	Multiply the Trigger Counter by 2 (2^1)

9.6.6 Supported Test Patterns/Modes Register, 0x200

The pattern generator unit provides a set of known bit sequences or patterns that can be output over the Trace Port and be detected by the TPA or other associated trace capture device. See *TPIU pattern generator* on page 9-32 for more information about the pattern generator unit.

Figure 9-6 on page 9-14 shows the Supported Test Patterns/Modes Register bit assignments.

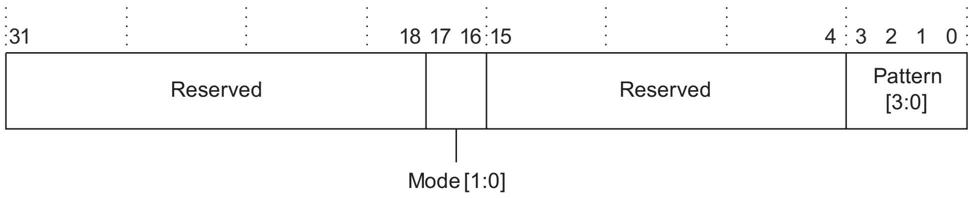


Figure 9-6 Supported Test Patterns/Modes Register bit assignments

Table 9-8 shows the Supported Test Patterns/Modes Register bit assignments.

Table 9-8 Supported Test Patterns/Modes Register bit assignments

Bits	Type	Name	Description
[31:18]	-	-	Reserved RAZ/SBZP
[17]	RO	PContEn	Continuous mode
[16]	RO	PTimeEn	Timed mode
[15:4]	-	-	Reserved RAZ/SBZP
[3]	RO	PatF0	FF/00 Pattern
[2]	RO	PatA5	AA/55 Pattern
[1]	RO	PatW1	Walking 1s Pattern
[0]	RO	PatW0	Walking 0s Pattern

9.6.7 Current Test Patterns/Modes Register, 0x204

This register follows the same structure as the Supported Test Modes/Patterns register. Only one of the modes can be set, using bits 17-16, but a multiple number of bits for the patterns can be set using bits 3-0. If Timed Mode is chosen, then after the allotted number of cycles has been reached, the mode automatically switches to Off Mode. On reset this register is set to 18'h00000, Off Mode with no selected patterns.

9.6.8 TPIU Test Pattern Repeat Counter Register, 0x208

This is an eight-bit counter start value that is decremented. A write sets the initial counter value and a read returns the programmed value. On reset this value is set to 0.

Figure 9-7 shows the TPIU Test Pattern Repeat Counter Register bit assignments.

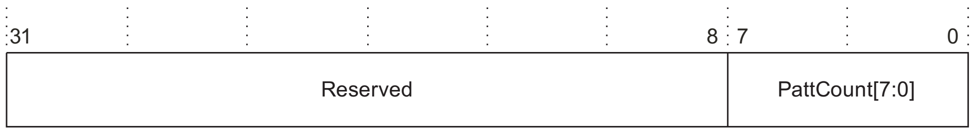


Figure 9-7 Test Pattern Repeat Counter Register bit assignments

Table 9-9 shows the TPIU Test Pattern Repeat Counter Register bit assignments.

Table 9-9 Test Pattern Repeat Counter Register bit assignments

Bits	Type	Name	Description
[31:8]	-	-	Reserved RAZ/SBZP
[7:0]	R/W	PattCount	8-bit counter value to indicate the number of **TRACECLKIN** cycles that a pattern runs for before switching to the next pattern. Default value is 0.

9.6.9 Formatter and Flush Status Register, 0x300

The Formatter and Flush Status Register is read only. This register indicates the implemented Trigger Counter multipliers and other supported features of the trigger system. Figure 9-8 shows the Formatter and Flush Status Register bit assignments.

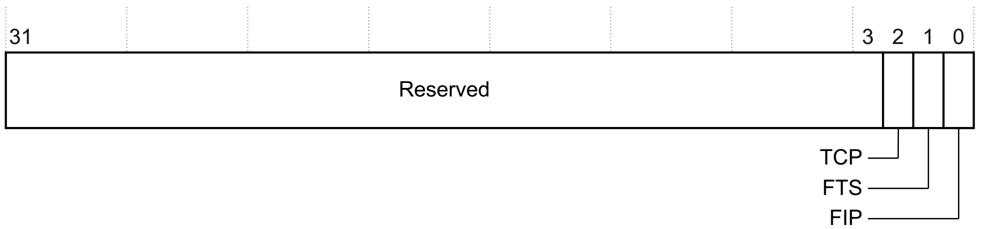


Figure 9-8 Formatter and Flush Status Register bit assignments

Table 9-10 shows the Formatter and Flush Status Register bit assignments.

Table 9-10 Formatter and Flush Status Register bit assignments

Bits	Type	Name	Description
[31:3]	-	-	Reserved RAZ/SBZP.
[2]	RO	TCPresent	If this bit is set then **TRACECTL** is present. If no **TRACECTL** pin is available, that is, this bit is zero, then the data formatter must be used and only in continuous mode. This is constrained by the CSTPIU_TRACECTL_VAL Verilog `define, which is not user modifiable, and the external tie-off **TPCTL**. If either constraint reports zero/LOW then no **TRACECTL** is present and this inability to use the pin is reflected in this register. See *TRACECTL removal* on page 9-29 for more information.
[1]	RO	FtStopped	Formatter stopped. The formatter has received a stop request signal and all trace data and post-amble has been output. Any more trace data on the ATB interface is ignored and **ATREADYS** goes HIGH.
[0]	RO	FlInProg	Flush In Progress. This is an indication of the current state of **AFVALIDS**.

9.6.10 Formatter and Flush Control Register, 0x304

This register controls the generation of stop, trigger, and flush events. Figure 9-9 shows the Formatter and Flush Control Register bit assignments.

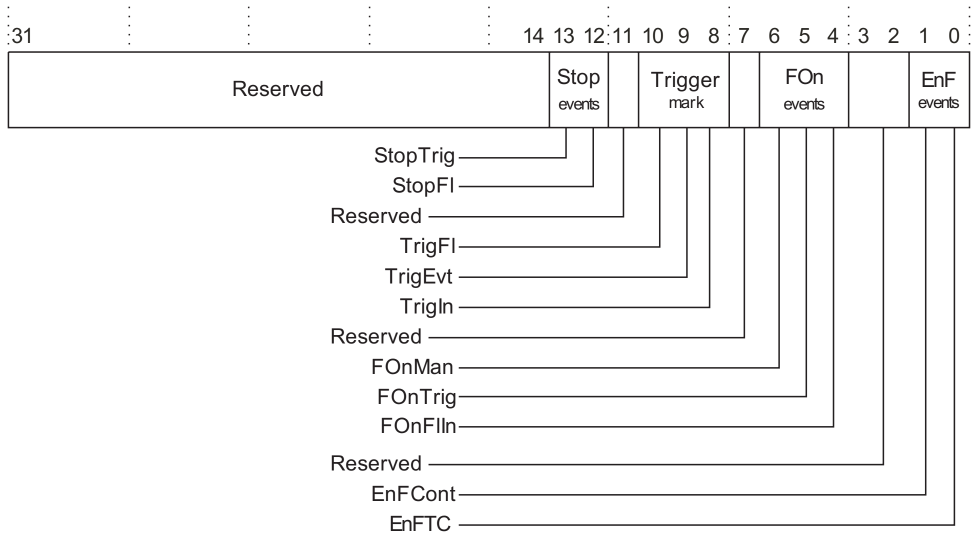


Figure 9-9 Formatter and Flush Control Register bit assignments

Table 9-11 shows the Formatter and Flush Control Register bit assignments.

Table 9-11 Formatter and Flush Control Register bit assignments

Bits	Type	Name	Description
[31:14]	-	-	Reserved RAZ/SBZP.
[13]	R/W	StopTrig	Stop the formatter after a Trigger Event[a] is observed. Reset to disabled, or zero.
[12]	R/W	StopFl	Stop the formatter after a flush completes (return of **AFREADYS**). This forces the FIFO to drain off any part-completed packets. Setting this bit enables this function but this is clear on reset, or disabled.
[11]	-	-	Reserved RAZ/SBZP.
[10]	R/W	TrigFl	Indicates a trigger on Flush completion on **AFREADYS** being returned.
[9]	R/W	TrigEvt	Indicate a trigger on a Trigger Event[a].
[8]	R/W	TrigIn	Indicate a trigger on **TRIGIN** being asserted.
[7]	-	-	Reserved RAZ/SBZP.
[6]	R/W	FOnMan	Manually generate a flush of the system. Setting this bit causes a flush to be generated. This is cleared when this flush has been serviced. This bit is clear on reset.
[5]	R/W	FOnTrig	Generate flush using Trigger event. Set this bit to cause a flush of data in the system when a Trigger Event[a] occurs. Reset value is this bit clear.
[4]	R/W	FOnFlIn	Generate flush using the **FLUSHIN** interface. Set this bit to enable use of the **FLUSHIN** connection. This is clear on reset.
[3:2]	-	-	Reserved RAZ/SBZP.
[1]	R/W	EnFCont	Continuous Formatting, no **TRACECTL**. Embed in trigger packets and indicate null cycles using Sync packets. Reset value is this bit clear. Can only be changed when FtStopped is HIGH.
[0]	R/W	EnFTC	Enable Formatting. Do not embed Triggers into the formatted stream. Trace disable cycles and triggers are indicated by **TRACECTL**, where fitted. Reset value is this bit clear. Can only be changed when FtStopped is HIGH.

a. A Trigger Event is defined as when the Trigger counter reaches zero or, in the case of the Trigger counter being zero, when **TRIGIN** is HIGH.

To disable formatting and put the formatter into bypass mode, bits 1 and 0 must be clear. Setting both bits is the same as setting bit 1.

All three flush generating conditions can be enabled together. However, if a second or third flush event is generated from another condition then the current flush completes before the next flush is serviced. Flush from **FLUSHIN** takes priority over flush from Trigger, which in turn completes before a manually activated flush. All Trigger indication conditions can be enabled simultaneously although this can cause the appearance of multiple triggers if flush using trigger is also enabled.

Both 'Stop On' settings can be enabled, although if flush on trigger is set up then none of the flushed data is stored. When the system stops, it returns **ATREADYS** and does not store the accepted data packets. This is to avoid stalling of any other devices which are connected to a Trace Replicator.

9.6.11 Formatter Synchronization Counter Register, 0x308

The Formatter Synchronization Counter Register enables effective use on different sized *Trace Port Analyzers* (TPAs) without wasting large amounts of the storage capacity of the capture device.

This counter is the number of formatter frames since the last synchronization packet of 128 bits, and is a 12-bit counter with a maximum count value of 4096. This equates to synchronization every 65536 bytes, that is, 4096 packets x 16 bytes per packet. The default is set up for a synchronization packet every 1024 bytes, that is, every 64 formatter frames.

Figure 9-10 shows the Formatter Synchronization Counter Register bit assignments.

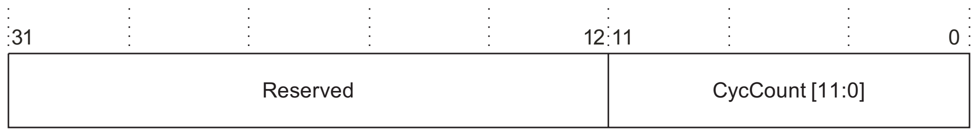


Figure 9-10 Formatter Synchronization Counter Register bit assignments

Table 9-12 shows the Formatter Synchronization Counter Register bit assignments.

Table 9-12 Formatter Synchronization Counter Register bit assignments

Bits	Type	Name	Description
[31:12]	-	-	Reserved RAZ/SBZP.
[11:0]	R/W	CycCount	12-bit counter value to indicate the number of complete frames between full synchronization packets. Default value is 64 (`0x40`).

If the formatter has been configured for continuous mode, full and half-word sync frames are inserted during normal operation. Under these circumstances the count value represents the maximum number of complete frames between full synchronization packets.

See *Supported Trigger Modes Register, 0x100* on page 9-11 for more on different modes.

9.6.12 TPIU Integration Test Registers

Integration Test Registers are provided to simplify the process of verifying the integration of the TPIU with other devices in a CoreSight system. These registers enable direct control of outputs and the ability to read the value of inputs. You must only use these registers when the Integration Test Control Register (`0xF00`) bit [0] is set to 1.

The registers in the TPIU enable the system to set the **FLUSHINACK** and **TRIGINACK** output pins. The **FLUSHIN** and **TRIGIN** inputs to the TPIU can also be read. The other Integration Test Registers are for testing the integration of the ATB slave interface on the TPIU. For details of how to use these signals see the applicable CoreSight DK Integration Manual.

This section describes the following registers:

- *Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4*
- *Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8* on page 9-20
- *Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC* on page 9-20
- *Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 9-21
- *Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4* on page 9-22
- *Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 9-22.

Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4

The Integration Test Trigger In and Flush In Acknowledge Register enables control of the **TRIGINACK** and **FLUSHINACK** outputs from the TPIU. Figure 9-11 shows the bit assignments.

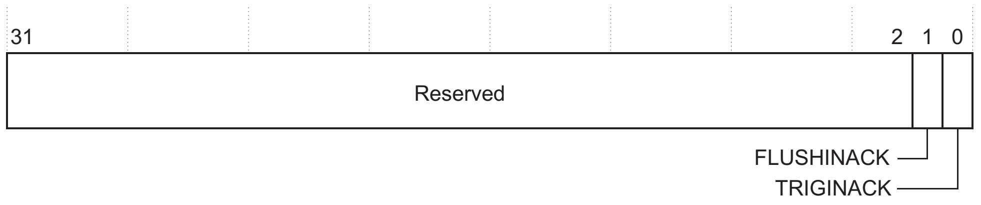


Figure 9-11 Integration Test Trigger In and Flush In Acknowledge Register bit assignments

Table 9-13 shows the bit assignments.

Table 9-13 Integration Test Trigger In and Flush In Acknowledge Register bit assignments

Bits	Type	Name	Description
[31:2]	-	-	Reserved
[1]	WO	FLUSHINACK	Set the value of **FLUSHINACK**
[0]	WO	TRIGINACK	Set the value of **TRIGINACK**

Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8

The Integration Test Trigger In and Flush In Register contains the values of the **FLUSHIN** and **TRIGIN** inputs to the TPIU. Figure 9-12 shows the bit assignments.

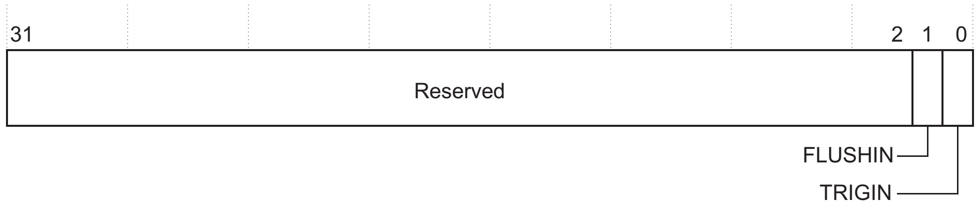


Figure 9-12 Integration Test Trigger In and Flush In Register bit assignments

Table 9-14 shows the bit assignments.

Table 9-14 Integration Test Trigger In and Flush In Register bit assignments

Bits	Type	Name	Description
[31:2]	-	-	Reserved RAZ/SBZP
[1]	RO	FLUSHIN	Read the value of **FLUSHIN**
[0]	RO	TRIGIN	Read the value of **TRIGIN**

Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC

The Integration Test ATB Data Register 0 contains the value of the **ATDATAS** inputs to the TPIU. The values are only valid when **ATVALIDS** is HIGH. Figure 9-13 on page 9-21 shows the bit assignments.

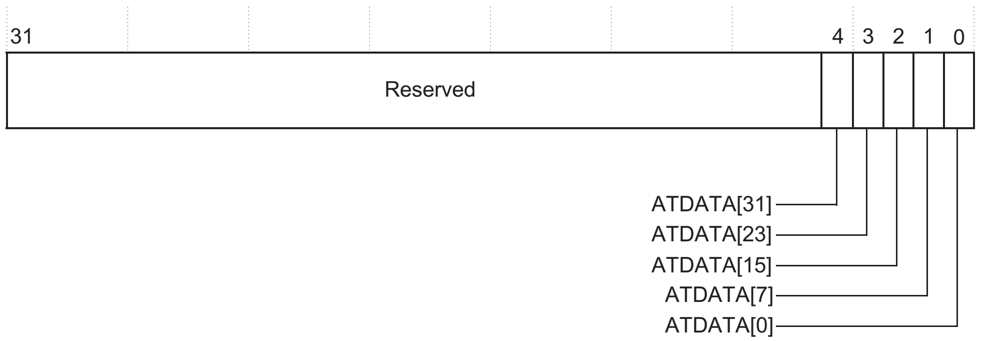


Figure 9-13 Integration Test ATB Data Register 0 bit assignments

Table 9-15 shows the bit assignments.

Table 9-15 Integration Test ATB Data Register 0 bit assignments

Bits	Type	Name	Description
[31:5]	-	-	Reserved
[4]	RO	ATDATA[31]	Read the value of **ATDATAS[31]**
[3]	RO	ATDATA[23]	Read the value of **ATDATAS[23]**
[2]	RO	ATDATA[15]	Read the value of **ATDATAS[15]**
[1]	RO	ATDATA[7]	Read the value of **ATDATAS[7]**
[0]	RO	ATDATA[0]	Read the value of **ATDATAS[0]**

Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0

The Integration Test ATB Control Register 2 enables control of the **ATREADYS** and **AFVALIDS** outputs of the TPIU. Figure 9-14 shows the bit assignments.

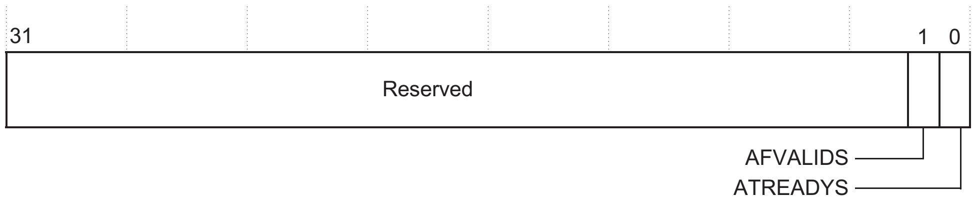


Figure 9-14 Integration Test ATB Control Register 2 bit assignments

Table 9-16 shows the bit assignments.

Table 9-16 Integration Test ATB Control Register 2 bit assignments

Bits	Type	Name	Description
[31:2]	-	-	Reserved
[1]	WO	AFVALID	Set the value of **AFVALIDS**
[0]	WO	ATREADY	Set the value of **ATREADYS**

Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4

The Integration Test ATB Control Register 1 contains the value of the **ATIDS** input to the TPIU. This is only valid when **ATVALIDS** is HIGH. Figure 9-15 shows the bit assignments.

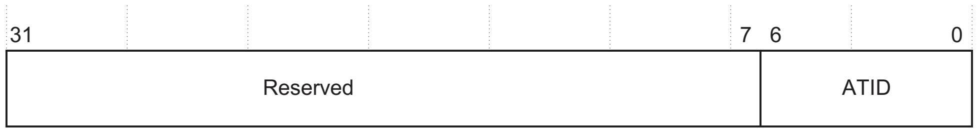


Figure 9-15 Integration Test ATB Control Register 1 bit assignments

Table 9-17 shows the bit assignments.

Table 9-17 Integration Test ATB Control Register 1 bit assignments

Bits	Type	Name	Description
[31:7]	-	-	Reserved RAZ/SBZP
[6:0]	RO	ATID	Read the value of **ATIDS**

Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8

The Integration Test ATB Control Register 0 captures the values of the **ATVALIDS**, **AFREADYS**, and **ATBYTESS** inputs to the TPIU. To ensure the integration registers work correctly in a system, the value of **ATBYTESS** is only valid when **ATVALIDS**, bit [0], is HIGH. Figure 9-16 on page 9-23 shows the bit assignments.

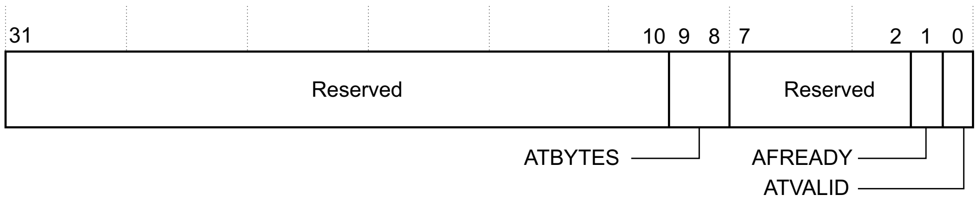


Figure 9-16 Integration Test ATB Control Register 0 bit assignments

Table 9-18 shows the bit assignments.

Table 9-18 Integration Test ATB Control Register 0 bit assignments

Bits	Type	Name	Description
[31:10]	-	-	Reserved RAZ/SBZP
[9:8]	RO	ATBYTES	Read the value of **ATBYTESS**
[7:2]	-	-	Reserved RAZ/SBZP
[1]	RO	AFREADY	Read the value of **AFREADYS**
[0]	RO	ATVALID	Read the value of **ATVALIDS**

9.6.13 TPIU EXCTL Port Registers

Two ports can be used as a control and feedback mechanism for any serializers, pin sharing multiplexors or other solutions that might be added to the trace output pins either for pin control or a high speed trace port solution. These ports are raw register banks that sample or export the corresponding external pins. The output register bank is set to all zeros on reset. The input registers sample the incoming signals and as such are undefined.

9.7 TPIU trace port sizes

The TPIU is configured for the largest port size allowable, 32 bits of **TRACEDATA**, **TRACECLK**, and **TRACECTL**.

- **TRACECLK** is always exported to enable synchronization back with the data and so is not optional.
- **TRACECTL** is required unless a new TPA is used that is aware of the formatter protocol and so can remove extra packets used to expand sequences. For Normal and Bypass modes, **TRACECTL** must be present.
- **TRACEDATA** can be defined as any size up to 32 bits. For backwards compatibility and usage with ETMv3 trace capture devices, a minimum port width of 2 bits is permitted, that is, **TRACEDATA[1:0]**.

Table 9-19 shows some typical Trace Out Port sizes.

Table 9-19 Example Trace Out Port sizes

TRACECLK present	TRACECTL present	TRACEDATA width	Total pin count	Comment
Yes	Yes	32 bits [31:0]	34	Largest implementation
Yes	No	9 bits [8:0]	10	Extra data pin available in comparison to the typical ETM implementation.
Yes	Yes	8 bits [7:0]	10	Typical ETM-compatible TPA implementation.
Yes	Yes	2 bits [1:0]	4	Smallest implementation with typical TPAs.
Yes	No	1 bit [0]	2	Smallest implementation with a protocol-aware TPA

9.7.1 Programming registers

There are two registers that contain information relating to the physical Trace Out port. These are the Supported Port Size Register and the Formatter and Flush Status Register. For more information about these registers see *Supported Port Size Register, 0x000* on page 9-10 and *Formatter and Flush Control Register, 0x304* on page 9-16.

Constraining the supported TRACEDATA port widths

The TPIU currently supports data port widths from 1-32 bits. A Verilog `define, CSTPIU_SUPPORTSIZE_VAL, is used to state the supported port sizes by the RTL. This is currently set to report that all options are supported and is not user modifiable.

When placing on an *Application Specific Integrated Circuit* (ASIC), not all the signals of **TRACEDATA** can go to pads, that is, only **TRACEDATA**[(MDS-1):0] go to pins, where MDS represents the *Maximum Data Size* going to pins. If MDS is not 32, that is, the maximum supported data width for capture by a TPA is less than 32 bits of data, this must be reflected in the programmer's view of the Supported Port Size Register through the tie-off input **TPMAXDATASIZE[4:0]**. See *Supported Port Size Register, 0x000* on page 9-10 for more details.

The tie-off must be set to represent the number of **TRACEDATA** connections that go to ASIC pads, with all LOW indicating 1 pin, which is the minimum possible, and all HIGH indicating 32 pins, which is a full **TRACEDATA** bus. For example, if a 16-bit trace port is implemented, that is **TRACEDATA[15:0]** is connected, then **TPMAXDATASIZE[4:0]** must be tied to `0x0F`. With a maximum **TRACEDATA[7:0]** connected to the ASIC, the tie-offs must be set to `0x07`.

9.7.2 Omission of TRACECTL

For restricted pin devices where removal of the trace data is important, the **TRACECTL** can be removed. Omitting **TRACECTL** is only possible with the formatter enabled and continuous mode selected. See *Formatter and Flush Control Register, 0x304* on page 9-16.

There are two mechanisms that can report the omission or presence of the **TRACECTL** pin:

- A Verilog `define, CSTPIU_TRACECTL_VAL, in the RTL that can be set to zero. This is not user-modifiable.
- An external tie-off, **TPCTL**, that must be tied LOW if no **TRACECTL** is present.

Currently, only the changing of **TPCTL** is supported and this must be done within the ASIC to indicate the presence, tied HIGH, or absence, tied LOW, of a **TRACECTL** pin.

9.8 TPIU triggers

Currently the only usage of triggers is by the trace capture device. This method is straightforward when using one trace source. When using multiple trace sources there can be a time disparity between the trace sources that generate a trigger to when the trigger packet, from trace sources, appears at the output of the trace port. See the *CoreSight Architecture Specification* for more information on triggers.

The outside world could interpret a trigger as an event that occurred. This could be:

- directly from an event such as a pin toggle from the CTI
- a delayed event such as a pin toggle that has been delayed coming through the Trigger Counter Register
- the completion of a flush.

Table 9-20 extends the ETMv3 specification on how a trigger is represented

Table 9-20 CoreSight representation of triggers

TRACECTL	TRACEDATA		Trigger	Capture	Description
	[1]	[0]	Yes/No	Yes/No	
0	x	x	No	Yes	Normal Trace Data
1	0	0	Yes	Yes	Trigger Packet[a]
1	1	0	Yes	No	Trigger
1	x	1	No	No	TraceDisable

a. The trigger packet encoding is required for the current ETMv3 protocol which uses a special encoding for triggers that always occur on the lower bits of **TRACEDATA**.

9.8.1 Correlation with AFVALID

When a trigger signal is received by the TPIU, depending on the Formatter Control Register, the **AFVALID** port can be asserted to cause a flush of all current trace information. This causes all information around the trigger event to be flushed from the system before normal trace information is resumed. This ensures that all information that could relate to the trigger is output before the TPA, or other capture device, is stopped.

Using FOnTrig HIGH, it is possible to indicate the trigger on completion of the flush routine, so ensuring that if the TPA stopped capture on a trigger, the TPA does get all historical data relating to the trigger. See *Formatter and Flush Control Register, 0x304* on page 9-16.

9.9 Other TPIU design considerations

This section presents the following design considerations:

- *TRACECLK generation*
- *TRACECTL removal* on page 9-29
- *TRACECTL and TRACEDATA multiplexing* on page 9-30
- *Off-chip based TRACECLKIN* on page 9-30.

9.9.1 TRACECLK generation

In an ideal environment **TRACECLK** is 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 large number of issues during clock-tree synthesis, layout and static timing analysis.

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.

There is no requirement to either invert the clock or use negative-edge registers in the generation of **TRACECLK**. 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 in the exported data, a delay must be added to the path of **TRACECLK** between the register and the pad.

Figure 9-17 on page 9-28 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 shown at point A in Figure 9-17 on page 9-28.

All the signal paths to the pads are subject to delays as a result of the differing 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 center of stable data on **TRACECTL** and **TRACEDATA** as shown in Figure 9-18 on page 9-29.

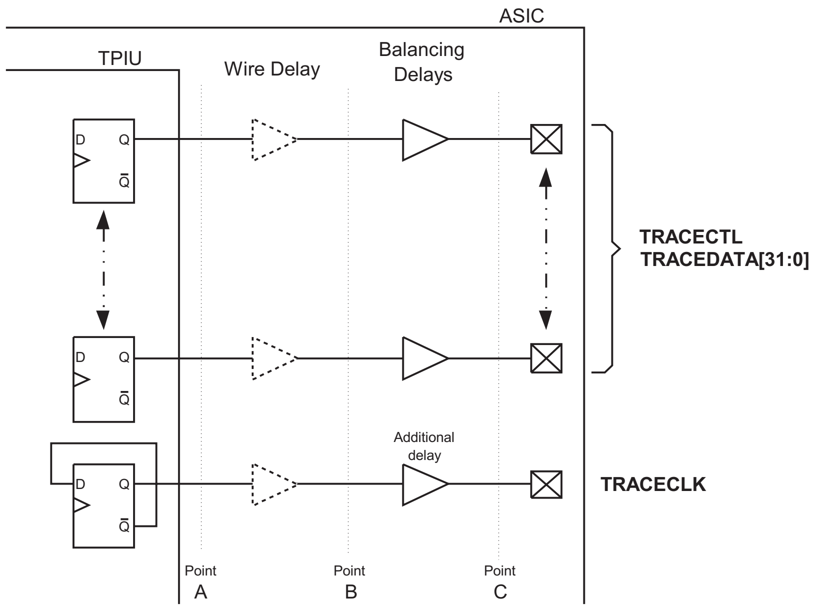


Figure 9-17 Paths of TRACECLK, TRACEDATA, and TRACECTL to pads

Figure 9-18 on page 9-29 shows how the rising and falling edges of **TRACECLK** correspond to the center of stable data on **TRACECTL** and **TRACEDATA** at point C.

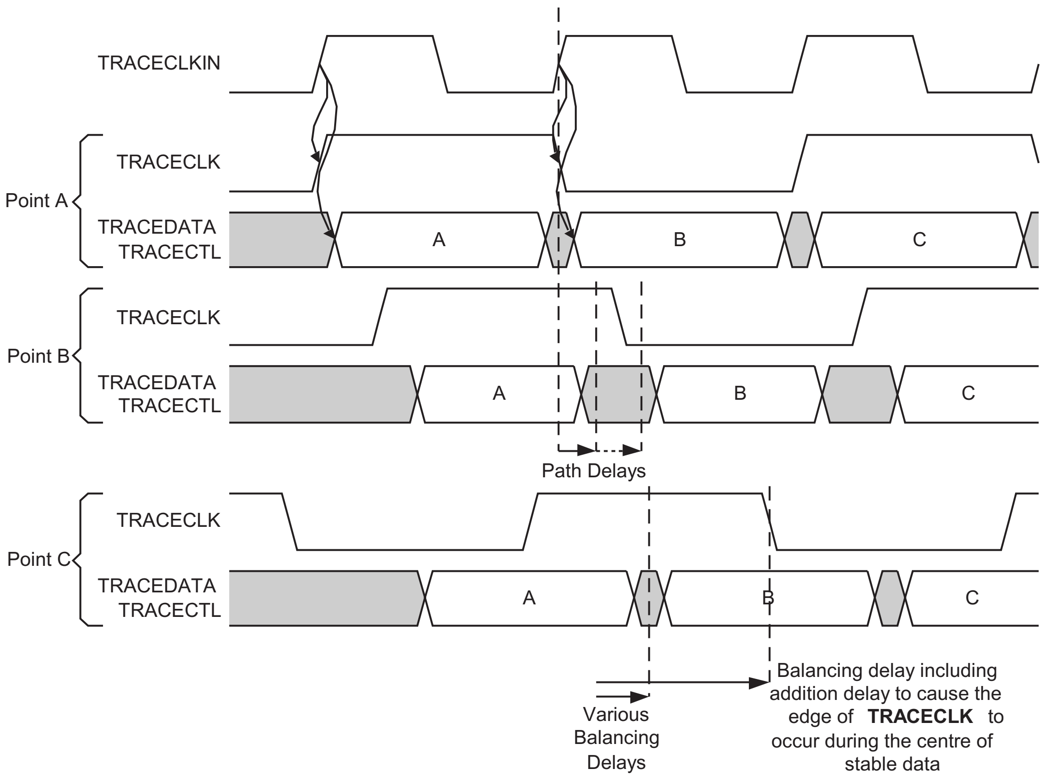


Figure 9-18 TRACECLK timing in relation to TRACEDATA and TRACECTL

9.9.2 TRACECTL removal

The TPIU supports two modes:

- data + control + clock, with a minimum data width of 2
- data + clock, with a minimum data width of 1.

The chosen mode depends on the connected trace port analyzer/capture device. Legacy capture devices use the control pin to indicate the packet type. Newer capture devices can use more pins for data and do not require a reserved data pin.

Support for both of these modes is required to ensure backwards compatibility and for future, higher port speeds. If a low pin count or an optimized design is required, it is not necessary to implement the **TRACECTL** pin. This design choice must be reflected in the programmer's model to enable tools to always enable the formatter and run in continuous mode.

9.9.3 TRACECTL and TRACEDATA multiplexing

If pin minimization is a priority, and it is also necessary to support legacy TPAs that still require **TRACECTL**, it is possible to support both systems in the same implementation by multiplexing the **TRACECTL** pin with a data pin. This enables the support of the current method of using the control pin at the same time as allowing the connection of a next generation trace capture device with the added advantage of an extra data pin. A possible choice for this extra data pin would be the most significant bit because this could be switched without having to change the signal paths of any other connection.

The ability to switch **TRACECTL** with a data pin is not directly supported by the TPIU. Problems can arise when trying to drive multiple connector pins, for connector re-use, because of impedance and load differences. In addition, timing can be affected because of the inclusion of a multiplexor on a limited number of signals

9.9.4 Off-chip based TRACECLKIN

Future CoreSight-aware TPAs might directly control a clock source for the Trace Out port. By running through a known sequence of patterns, from the pattern generator within the TPIU, a TPA could automatically establish the port width and ramp up the clock speed until the patterns degrade, thereby establishing a maximum data rate. Figure 9-19 shows how an off-chip **TRACECLKIN** could be generated.

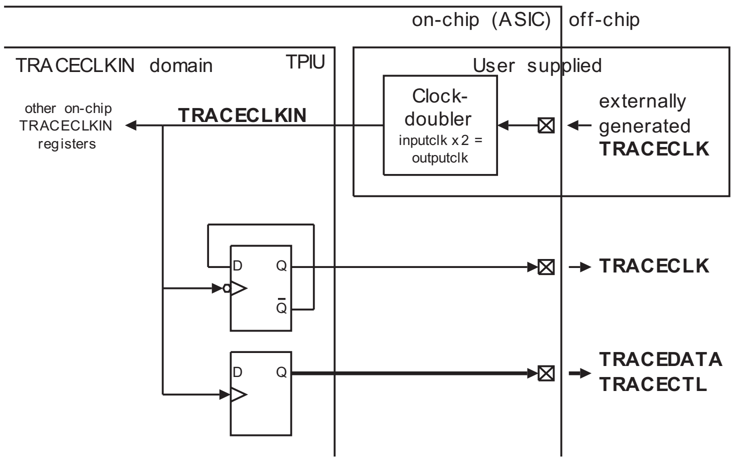

on-chip (ASIC)
off-chip
TRACECLKIN domain
TPIU
User supplied
other on-chip TRACECLKIN registers
TRACECLKIN
Clock-doubler inputclk x 2 = outputclk
externally generated TRACECLK
D
Q
Q̄
TRACECLK
D
Q
TRACEDATA
TRACECTL

Figure 9-19 Externally derived TRACECLK

The off-chip clock would operate in a similar way to the currently exported **TRACECLK**, that is, an externally derived clock source would be clock-doubled to enable the exported data to change at both edges of the clock.

9.10 Authentication requirements for TPIUs

TPIUs do not require any authentication signals capable of disabling them.

9.11 TPIU pattern generator

A simple set of defined bit sequences or patterns can be output over the Trace Port and be detected by the TPA or other associated trace capture device. Analysis of the output can indicate if it was possible to increase or, for reliability, to decrease the trace port clock speed. The patterns can also be used to determine the timing characteristics and so alter any delay components on the data channels in a TCD, to ensure reliable data capture.

9.11.1 Pattern generator modes of operation

There are a number of patterns supported to allow a number of metrics to be determined, for example, timing between pins, data edge timing, voltage fluctuations, ground bounce, and cross talk. When examining the trace port, there are two pattern modes you can choose:

Timed Each pattern runs for a programmable number of **TRACECLKIN** cycles after which the pattern generator unit reverts back to an off state where normal trace is output, assuming trace output is enabled. The first thing the trace port outputs after returning to normal trace is a synchronization packet. This is useful with special trace port analyzers and capture devices that are aware of the pattern generator, and so the TPIU can be set to a standard configuration which the capture device is expecting. The preset test pattern can then be run, by the end of which, the TCD is calibrated ready for normal operation, which the TPIU switches to automatically, without the requirement to reprogram the TPIU.

Continuous The selected pattern runs continuously until manually disabled. This is primarily intended for manual refinement of electrical characteristics and timing.

Off When neither of the other two modes is selected the device reverts to outputting any trace data. After timed operation finishes, the pattern generator reverts back to the Off Mode.

9.11.2 Supported options

Patterns operate over all the **TRACEDATA** pins for a given port width setting. Test patterns are aware of port sizes and always align to **TRACEDATA[0]**. Walking bit patterns wrap at the highest data pin for the selected port width even if the device has a larger port width available. Also, the alternating patterns do not affect unenabled data pins on smaller trace port sizes.

Walking 1s

All output pins clear (0) with a single bit set at a time, tracking across every **TRACEDATA** output pin. This can be used to watch for data edge timing, or synchronization, high voltage level of logic 1 and cross talk against adjacent wires. This can also be used as a simple way to test for broken/faulty cables and data signals.

Walking 0s

All output pins are set (1) with a single bit cleared at a time, tracking across every **TRACEDATA** output pin. In a similar way to the walking 1s this can be used to watch for data edge timing, or synchronization, low voltage level of logic 0, cross talk, and ground lift.

Alternating AA/55 pattern

Alternate **TRACEDATA** pins set with the others clear. This alternates every cycle with the sequence starting with **TRACEDATA[1]** set ('AA' pattern == 8'b1010_1010) and then **TRACEDATA[0]** set ('55' pattern == 8'b0101_0101). The pattern repeats over the entire selected bus width. This pattern can be used to check voltage levels, cross talk and data edge timing.

Alternating FF/00 pattern

On each clock cycle the **TRACEDATA** pins are either all set ('FF' pattern) or all cleared ('00' pattern). This sequence of alternating the entire set of data pins is a good way to check any power supply stability to the TPIU and the final pads because of the stresses the drivers are under.

Combinations of patterns

Each selected pattern is repeated for a defined number of cycles before moving onto the next pattern. After all patterns have been performed, the unit switches to normal mode that is, tracing data. If some combination is chosen and the continuous mode selected, each pattern runs for the number of cycles indicated in the repeat counter register before looping around enabled parameters.

9.12 TPIU formatter and FIFO

This section describes the functionality of the formatter and the FIFO.

The formatter is the final unit on the ATB that inserts the **ATID[6:0]** into a special format data packet stream to enable trace data to be re-associated with a trace source.

The FIFO is 128 bits wide. To reduce the amount of logic required for this operation and that of the high-speed bridge, the FIFO is of an asynchronous design.

9.12.1 Operational description

Figure 9-20 shows the arrangement of four words and the positioning of the bits that are removed from some of the data packets.

When a trace source is changed the appropriate flag bit, F, is set (1 = ID, 0 = Data on the following byte). The second byte is always data and the corresponding bit at the end of the sequence (bits A-J) indicates if this second byte corresponds to the new ID (F bit clear) or the previous ID (F bit set).

If the trace source has not changed in eight formatter frames, an ID change is indicated, even though the new ID is the current ID value. This is done to ensure the external TCD log has a record in its buffer of the existing trace source.

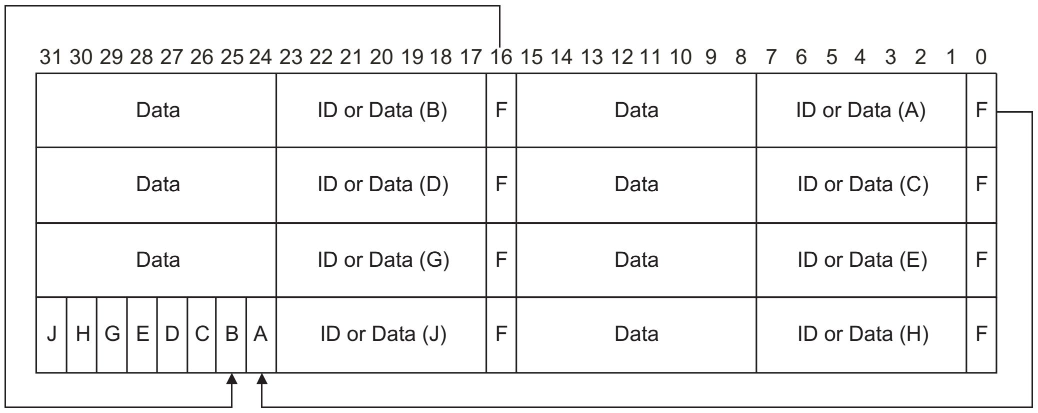


Figure 9-20 Construction of formatter data packets

See the *CoreSight Architecture Specification* for more information on the formatter protocol.

9.12.2 Special trace source IDs

The following IDs are set aside for special purposes and must not be used under normal operation:

0x00	NULL trace source. Typically this is not used except when draining the formatter where filling of empty locations of frames is required.
0x70-0x7C	Reserved.
0x7D	Trigger event.
0x7E	Reserved.
0x7F	Reserved. This must never be used as a trace source ID because this can cause issues with correctly detecting synchronization packets.

9.12.3 Supported modes of operation

The formatter within the TPIU supports three basic modes of operation:-

Bypass	IDs are not incorporated and raw trace data is emitted. It is assumed that the trace source is not changing. This requires the use of **TRACECTL**.
Normal	Typical operation with capture devices that require use of a **TRACECTL** pin.
Continuous	An extension of normal mode except when the control pin would normally indicate no data to be captured, there is data emitted that shows this fact. Also, triggers are embedded into the data stream because they cannot be indicated otherwise.

See the *CoreSight Architecture Specification* for more information on these modes.

9.12.4 Periodic synchronization

When the Formatter is in Continuous mode it can output more synchronization packets, both intraframe and interframe. When an interframe synchronization packet is emitted because of continuous mode, this causes the synchronization counter to restart counting. Full and half synchronization packets are output within interframe and intraframe respectively. See the *CoreSight Architecture Specification* for more information.

9.13 Configuration options

Existing TPAs that are only capable of operation with **TRACECTL** should only use the formatter in either bypass or normal mode, not continuous.

9.13.1 Configuration guidelines

TPIU configuration guidelines are as follows:

- It is recommended that following a trigger event within a multi-trace source configuration, a flush must be performed to ensure that all historical information that could relate to the trigger has been output.
- If Flush on Trigger Event and Stop on Trigger Event options are chosen then any data after the trigger is not captured by the TPA. When the TPIU is instructed to stop, it returns **ATREADYS** HIGH and does not store any of the accepted data.
- Although multiple flushes can be scheduled using Flush on Trigger Event, Flush on **FLUSHIN** and manual flush, when one of these requests have been made it masks any further requests of the same type, that is, repeated writing to the manual flush bit does not schedule multiple manual requests unless each is allowed to complete first. See *Generation of flush on FLUSHIN* on page 10-28, *Generation of flush from a trigger event* on page 10-29, and *Generation of a flush on manual* on page 10-30 for diagrams showing these sequences in detail.
- Unless multiple triggers are required, it is not advisable to set both Trigger on Trigger Event and Trigger on Flush Completion, if Flush on Trigger Event is also enabled. In addition, if Trigger on **TRIGIN** is enabled with this configuration, it can also cause multiple trigger markers from one trigger request.

9.14 Example configuration scenarios

This section describes a number of configurations scenarios:

- *Capturing trace after an event and stopping*
- *Only indicating triggers and still flushing* on page 9-38
- *Multiple trigger indications* on page 9-38
- *Independent triggering and flushing* on page 9-39.

9.14.1 Capturing trace after an event and stopping

Two things must happen before trace capture can be stopped:

- a suitable length of time has to elapse to encompass knock-on effects within trace data
- all historical information relating to these previous events must have been emitted.

Figure 9-21 shows a possible timeline of events where an event of interest, referred to as a trigger event, causes some trace which must be captured and thereafter the trace capture device can be stopped.

When one trace source is used, there is no requirement to flush the system, instead the length of the trigger counter delay can be increased to allow more trace to be generated, thereby pushing out historical information.

Traditionally only the initial trigger event is sent to the TPA at time t1, indicated using **TRACECTL** and a special encoding on **TRACEDATA**. This can still be done but if trace is stopped at this point, there might still be related trace stalled within the ATB system. Trigger signals are now abstracted from ATB through the CTI/CTM infrastructure. To allow all trace information to have been output that could have related to an internally generated trigger event, the system must be flushed after which trace capture can be safely stopped.

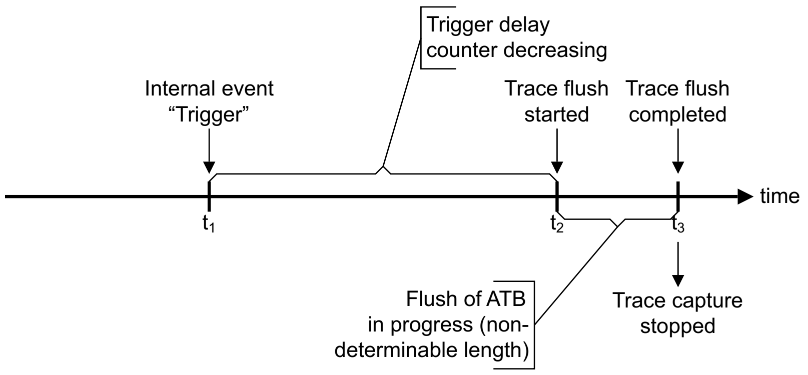


Figure 9-21 Capturing trace after an event and stopping

In Figure 9-21 on page 9-37 the action to cause trace capture to be stopped at time t3 could be:

- the TPA could watch for a trigger to be indicated through **TRACECTL** and stop
- the TPA could watch for a trigger to be indicated in the **TRACEDATA** stream that is, using continuous mode without the requirement for **TRACECTL**)
- the TPIU could automatically stop trace if the TPA could only recognize trace disable cycles, that is, it cannot act on trigger markers.

9.14.2 Only indicating triggers and still flushing

It is possible to still indicate a trigger at the soonest possible moment and cause a flush while at the same time still allowing externally requested flushes. This enables trace around a key event to be captured and all historical information to be stored within a period immediately following the trigger, and have some secondary event causing regular trace flushes.

9.14.3 Multiple trigger indications

After a trigger has been indicated to external tools this could cause more than just trace capture stopping. For example, in cases where the events immediately before the trigger might be important but only a small buffer is available, uploads to a host computer for decompression could occur so reducing the amount stored in the TPA. This would also be useful where the trigger originated from a device not directly associated with a trace source and is a marker for a repeating interesting event. See Figure 9-22.

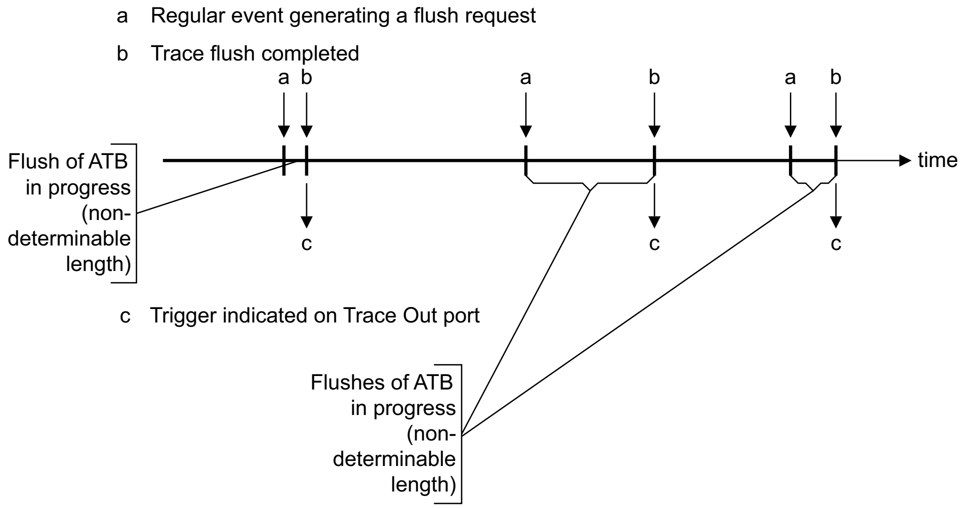

a Regular event generating a flush request
b Trace flush completed
a
b
a
b
a
b
time
Flush of ATB in progress (non-determinable length)
c
c
c
c Trigger indicated on Trace Out port
Flushes of ATB in progress (non-determinable length)

Figure 9-22 Multiple trigger indications from flushes

9.14.4 Independent triggering and flushing

The TPIU has separate inputs for flushes and triggers and, although one can be configured to generate the other, there might be a requirement to keep them separate. To enable a consistent flow of new information through the Trace Out port, there might be a regular flush scheduled, generated from a timing block connected to a CTI. These regular events must not be marked in the trace stream as triggers. Special events coming through the CTI that do require a marker should be passed through the **TRIGIN** pin which can either be immediately indicated or, as shown in Figure 9-23, it can be delayed through other flushes and then indicated to the TPA.

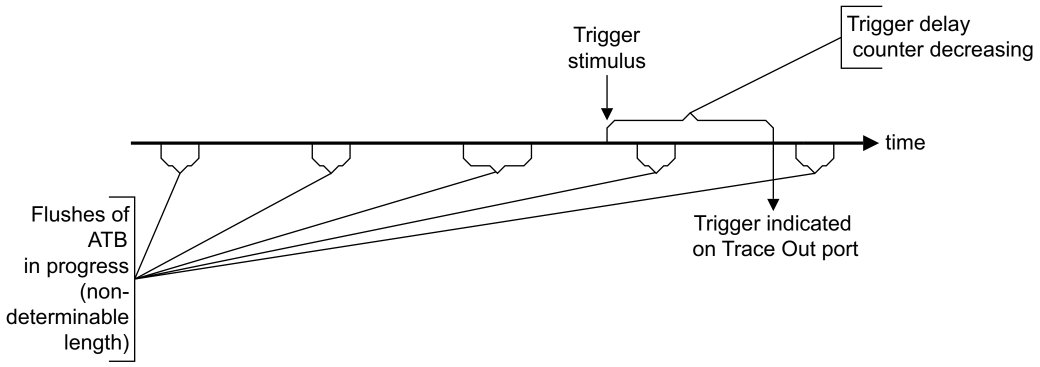

Trigger stimulus
Trigger delay counter decreasing
time
Flushes of ATB in progress (non-determinable length)
Trigger indicated on Trace Out port

Figure 9-23 Independent triggering during repeated flushes

Chapter 10
Embedded Trace Buffer

This chapter describes the *Embedded Trace Buffer* (ETB) for CoreSight. It contains the following sections:

10.1 About the ETB for CoreSight

The ETB provides on-chip storage of trace data using 32-bit RAM. Figure 10-1 shows the main ETB blocks.

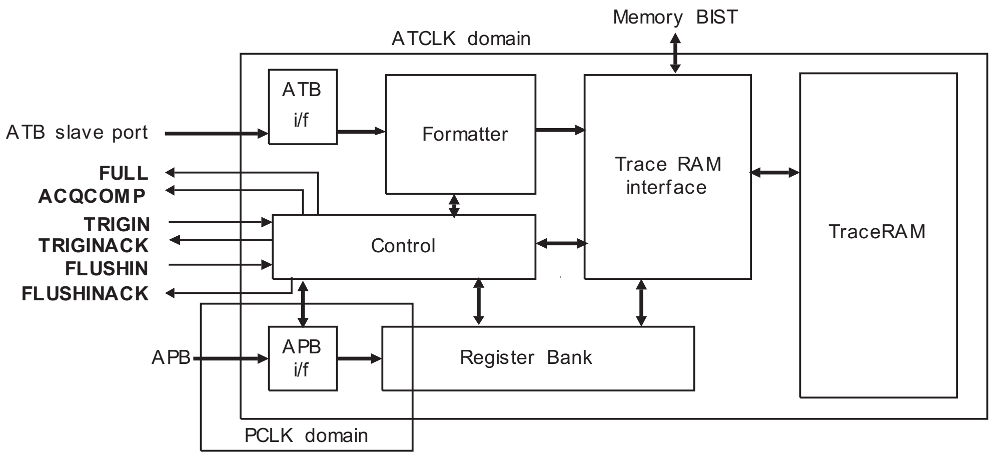


Figure 10-1 ETB block diagram

The ETB accepts trace data from CoreSight trace source components providing an *AMBA Trace Bus* (ATB) write port. The ATB is a common bus used by trace devices to share CoreSight capture resources. See the *CoreSight Architecture Specification* for a detailed description.

The ETB contains the following blocks:

Formatter Inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source after the data is read back out of the ETB.

Control Control registers for trace capture and flushing.

APB interface

Read, write, and data pointers provide access to ETB registers. In addition, the APB interface supports wait states through the use of a **PREADYDBG** signal output by the ETB.

The APB interface is synchronous to the ATB domain.

Register bank

Contains the management, control, and status registers for triggers, flushing behavior, and external control.

Trace RAM interface

Controls reads and writes to the Trace RAM.

Memory BIST interface

Provides test access to the Trace RAM.

10.1.1 ATB interface

The ETB accepts trace data from any CoreSight-compliant component trace source with an ATB master port, such as a trace source or a trace funnel.

10.1.2 ETB triggering and flushing ports

Table 10-1 shows the ETB triggering and flushing ports.

Table 10-1 ETB triggering and flushing ports

Name	Type	Description
TRIGIN	Input	From either a CTI or direct from a trace source. Used to enable the trigger to affect the captured trace stream.
TRIGINACK	Output	Return response to the CTI. Acknowledgement to **TRIGIN**.
FLUSHIN	Input	External control used to assert the ATB signal **AFVALIDS** and drain any historical FIFO information on the bus.
FLUSHINACK	Output	Return acknowledgement to **FLUSHIN**.

10.1.3 ETB status ports

The ETB continuously captures and writes data into RAM when trace capture is enabled and either:

- the trigger counter is static at 0
- the trigger counter has not yet decremented to zero. The trigger counter begins decrementing when a trigger is observed.

A trigger event is denoted when the trigger counter reaches zero, or if the trigger counter is zero when the trigger input is HIGH.

When the trigger counter reaches zero the acquisition complete flag, AcqComp, is activated and trace capture stops. The value loaded into the trigger counter therefore sets the number of data words stored in the trace RAM after a trigger event.

Table 10-2 shows the ETB status ports.

Table 10-2 ETB status ports

Name	Type	Description
ACQCOMP	Output	When HIGH indicates that trace acquisition is complete, that is, the trigger counter has reached zero.
FULL	Output	When HIGH indicates that the ETB RAM has overflowed or wrapped around to address zero.

10.1.4 Memory BIST interface

Table 10-3 shows the Memory BIST interface ports.

Table 10-3 ETB Memory BIST interface ports

Name	Type	Description
MBISTADDR [CSETB_ADDR_WIDTH-1:0]	Input	Address bus for the external BIST controller, active when **MTESTON** is HIGH. **CSETB_ADDR_WIDTH** defines the address bus width used, and therefore the RAM depth supported.
MBISTCE	Input	Active HIGH chip select for external BIST controller, active when **MTESTON** is HIGH.
MBISTDIN[31:0]	Input	Write data bus for external BIST controller, active when **MTESTON** is HIGH.
MBISTDOUT[31:0]	Output	Read data bus for external BIST controller, active when **MTESTON** is HIGH.
MBISTWE	Input	Active HIGH write enable for external BIST controller, active when **MTESTON** is HIGH.
MTESTON	Input	Enable signal for the external BIST controller.

10.2 ETB programmer's model

Table 10-4 shows the ETB registers.

Table 10-4 ETB register summary

Offset	Type	Width	Reset value	Name	Reference Section
`0x004`	RO	32	`0x00000000`	RAM Depth Register, RDP	*ETB RAM Depth Register, RDP, 0x004* on page 10-7
`0x010`	RO	32	`0x00000000`	RAM Read Data Register, RRD	*ETB RAM Read Data Register, RRD, 0x010* on page 10-8
`0x014`	R/W	32	`0x00000000`	RAM Read Pointer Register, RRP	*ETB RAM Read Pointer Register, RRP, 0x014* on page 10-9
`0x00C`	RO	32	`0x00000000`	Status Register, STS	*ETB Status Register, STS, 0x00C* on page 10-7
`0x018`	R/W	32	`0x00000000`	RAM Write Pointer Register, RWP	*ETB RAM Write Pointer Register, RWP, 0x018* on page 10-9
`0x01C`	R/W	32	`0x00000000`	Trigger Counter Register, TRG	*ETB Trigger Counter Register, TRG, 0x01C* on page 10-10
`0x020`	R/W	32	`0x00000000`	Control Register, CTL	*ETB Control Register, CTL, 0x020* on page 10-10
`0x024`	WO	32	`0x00000000`	RAM Write Data Register, RWD	*ETB RAM Write Data Register, RWD, 0x024* on page 10-11
`0x300`	RO	2	`0x0`	Formatter and Flush Status Register, FFSR	*ETB Formatter and Flush Status Register, FFSR, 0x300* on page 10-11
`0x304`	R/W	13	`0x0200`	Formatter and Flush Control Register, FFCR	*ETB Formatter and Flush Control Register, FFCR, 0x304* on page 10-12
`0xEE0`	WO	2	-	Integration Register, ITMISCOP0	*ETB Integration Test Registers* on page 10-14
`0xEE4`	WO	2	-	Integration Register, ITTRFLINACK	*Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4* on page 10-15
`0xEE8`	RO	2	Undefined	Integration Register, ITTRFLIN	*Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8* on page 10-16
`0xEEC`	RO	5	Undefined	Integration Register, ITATBDATA0	*Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC* on page 10-17
`0xEF0`	WO	2	-	Integration Register, ITATBCTR2	*Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 10-17

Table 10-4 ETB register summary (continued)

Offset	Type	Width	Reset value	Name	Reference Section
`0xEF4`	RO	7	Undefined	Integration Register, ITATBCTR1	*Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4* on page 10-18
`0xEF8`	RO	10	Undefined	Integration Register, ITATBCTR0	*Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 10-19
`0xF00`	R/W	1	`0x0`	Integration Mode Control Register	*CoreSight Management Registers summary* on page 2-12
`0xFA0`	R/W	4	`0xF`	Claim Tag Set Register	
`0xFA4`	R/W	4	`0x0`	Claim Tag Clear Register	
`0xFB0`	WO	32	-	Lock Access Register	
`0xFB4`	RO	3	`0x3`	Lock Status Register	
`0xFB8`	RO	8	`0x00`	Authentication Status Register	*ETB CoreSight management registers* on page 10-20
`0xFC8`	RO	32	`0x00`	Device ID	
`0xFCC`	RO	8	`0x21`	Device Type Identifier Register	
`0xFD0`	RO	8	`0x04`	Peripheral ID4	*CoreSight Management Registers summary* on page 2-12
`0xFD4`	RO	8	`0x00` (reserved)	Peripheral ID5	
`0xFD8`	RO	8	`0x00` (reserved)	Peripheral ID6	
`0xFDC`	RO	8	`0x00` (reserved)	Peripheral ID7	
`0xFE0`	RO	8	`0x07`	Peripheral ID0	
`0xFE4`	RO	8	`0xB9`	Peripheral ID1	
`0xFE8`	RO	8	`0x0B`	Peripheral ID2	
`0xFEC`	RO	8	`0x00`	Peripheral ID3	
`0xFF0`	RO	8	`0x0D`	Component ID0	See *CoreSight Management Registers summary* on page 2-12
`0xFF4`	RO	8	`0x90`	Component ID1	
`0xFF8`	RO	8	`0x05`	Component ID2	
`0xFFC`	RO	8	`0xB1`	Component ID3	

10.3 ETB register descriptions

The ETB registers are described in:

- *ETB RAM Depth Register, RDP, 0x004*
- *ETB Status Register, STS, 0x00C*
- *ETB RAM Read Data Register, RRD, 0x010* on page 10-8
- *ETB RAM Read Pointer Register, RRP, 0x014* on page 10-9
- *ETB RAM Write Pointer Register, RWP, 0x018* on page 10-9
- *ETB Trigger Counter Register, TRG, 0x01C* on page 10-10
- *ETB Control Register, CTL, 0x020* on page 10-10
- *ETB RAM Write Data Register, RWD, 0x024* on page 10-11
- *ETB Formatter and Flush Status Register, FFSR, 0x300* on page 10-11
- *ETB Formatter and Flush Control Register, FFCR, 0x304* on page 10-12
- *ETB Integration Test Registers* on page 10-14.

10.3.1 ETB RAM Depth Register, RDP, 0x004

Table 10-5 shows the ETB RAM Depth Register bit assignments.

Table 10-5 ETB RAM Depth Register bit assignments

Bits	Type	Name	Function
[31:0]	RO	RAM Depth Register, RDP	Defines the depth, in words, of the trace RAM. This value is configurable with the RTL, but fixed at synthesis. Supported depth in powers of 2 only. Reset value = Ram Depth which is given by a Verilog tick define.

10.3.2 ETB Status Register, STS, 0x00C

Figure 10-2 shows the ETB Status Register bit assignments.

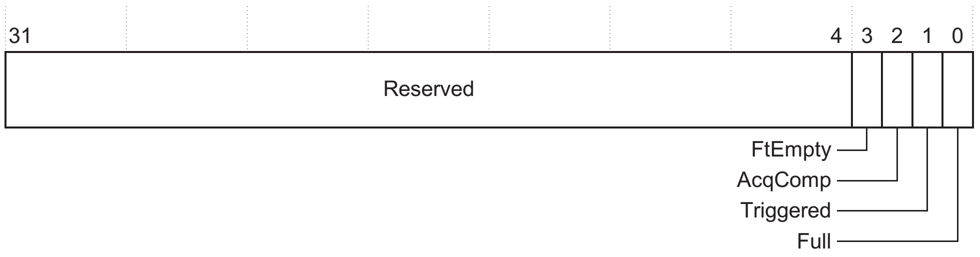


Figure 10-2 ETB Status Register bit assignments

Table 10-6 shows the ETB Status Register bit assignments.

Table 10-6 ETB Status Register bit assignments.

Bits	Type	Name	Function
[31:4]	-	-	Reserved.
[3]	RO	FtEmpty	Formatter pipeline empty. All data stored to RAM.
[2]	RO	AcqComp	Acquisition complete. The acquisition complete flag indicates that capture has been completed when the formatter stops because of any of the methods defined in the Formatter and Flush Control Register, or TraceCaptEn = 0. This also results in FtStopped in the Formatter and Flush Status Register going HIGH.
[1]	RO	Triggered	The Triggered bit is set when a trigger has been observed. This does not indicate that a trigger has been embedded in the trace data by the formatter, but is determined by the programming of the Formatter and Flush Control Register.
[0]	RO	Full	RAM Full. The flag indicates when the RAM write pointer has wrapped around.

10.3.3 ETB RAM Read Data Register, RRD, 0x010

When trace capture is disabled, the contents of the ETB Trace RAM at the location addressed by the RAM Read Pointer Registers are placed in this register. Reading this register increments the RAM Read Pointer Register and triggers a RAM access cycle.

If trace capture is enabled (FtStopped=0, TraceCaptEn=1), and ETB RAM reading is attempted, a read from this register outputs `0xFFFFFFFF` and the RAM Read Pointer Register does not auto-increment.

A constant stream of 1s being output corresponds to a synchronization output in the formatter protocol, which is not applicable to the ETB, and so can be used to signify a read error, when formatting is enabled.

Table 10-7 shows the ETB RAM Read Data Register bit assignments.

Table 10-7 ETB RAM Read Data Register bit assignments.

Bits	Type	Name	Function
[31:0]	RO	RAM Read Data, RRD	Data read from the ETB Trace RAM.

10.3.4 ETB RAM Read Pointer Register, RRP, 0x014

The RAM Read Pointer Register sets the read pointer used to read entries from the Trace RAM over the APB interface. When this register is written to, a RAM access is initiated. The RAM Read Data Register is then updated. The register can also be read to determine the current memory location being referenced.

This register must not be written to when trace capture is enabled (FtStopped=0, TraceCaptEn=1). If access is attempted, the register is not updated.

Table 10-8 shows the ETB RAM Read Pointer Register bit assignments.

Table 10-8 ETB RAM Read Pointer Register bit assignments.

Bits	Type	Name	Function
[CSETB_ADDR_WIDTH-1:0]	R/W	RAM Read Pointer, RRP	Sets the read pointer used to read entries from the Trace RAM over the APB interface.

10.3.5 ETB RAM Write Pointer Register, RWP, 0x018

The RAM Write Pointer Register sets the write pointer used to write entries from the CoreSight bus into Trace RAM. During trace capture the pointer increments when the DataValid flag is asserted by the Formatter. When this register increments from its maximum value back to zero, the Full flag is set.

This register can also be written to over APB to set the pointer for write accesses carried out through the APB interface.

This register must not be written to when trace capture is enabled (FtStopped=0, TraceCaptEn=1). If access is attempted, the register is not updated. The register can also be read to determine the current memory location being referenced.

Table 10-9 shows the ETB RAM Write Pointer Register bit assignments.

Table 10-9 ETB RAM Write Pointer Register bit assignments.

Bits	Type	Name	Function
[CSETB_ADDR_WIDTH-1:0]	R/W	RAM Write Pointer, RWP	Sets the write pointer used to write entries from the CoreSight bus into the Trace RAM

10.3.6 ETB Trigger Counter Register, TRG, 0x01C

The Trigger Counter Register disables write access to the Trace RAM by stopping the Formatter after a defined number of words have been stored following the trigger event. The number of 32-bit words written into the Trace RAM following the trigger event is equal to the value stored in this register+1.

Table 10-10 shows the ETB Trigger Counter Register bit assignments.

Table 10-10 ETB Trigger Counter Register bit assignments.

Bits	Type	Name	Function
[31:CSETB_ADDR_WIDTH]	-	-	Reserved. CSETB_ADDR_WIDTH defines the address bus width, and hence the RAM depth supported
[CSETB_ADDR_WIDTH-1:0]	R/W	Trigger Counter, TRG	Trigger Counter Register. The counter is used as follows: • Trace after The counter is set to a large value, slightly less than the number of entries in the RAM. • Trace before The counter is set to a small value. • Trace about The counter is set to half the depth of the Trace RAM. This register must not be written to when trace capture is enabled (FtStopped=0, TraceCaptEn=1). If a write is attempted, the register is not updated. A read access is permitted with trace capture enabled.

10.3.7 ETB Control Register, CTL, 0x020

Figure 10-3 shows the ETB Control Register bit assignments.

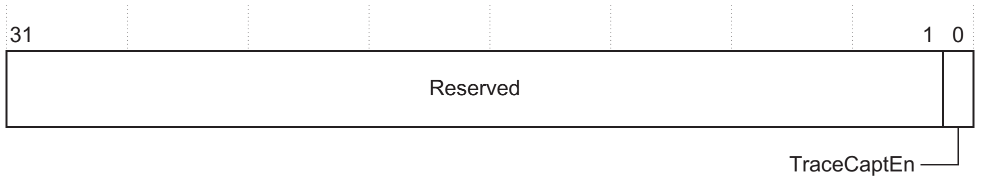


Figure 10-3 ETB Control Register bit assignments

Table 10-11 shows the ETB Control Register bit assignments.

Table 10-11 ETB Control Register bit assignments.

Bits	Type	Name	Function
[31:1]	-	-	Reserved
[0]	R/W	TraceCaptEn	ETB Trace Capture Enable. 1 = enable trace capture 0 = disable trace capture. This is the master enable bit forcing FtStopped HIGH when TraceCaptEn is LOW. When capture is disabled, any remaining data in the ATB formatter is stored to RAM. When all data is stored the formatter outputs FtStopped. Capture is fully disabled, or complete, when FtStopped goes HIGH. See *ETB Formatter and Flush Status Register, FFSR, 0x300*.

10.3.8 ETB RAM Write Data Register, RWD, 0x024

Table 10-12 shows the ETB RAM Write Data Register bit assignments.

Table 10-12 ETB RAM Write Data Register bit assignments

Bits	Type	Name	Function
[31:0]	R/W	RAM Write Data Register, RWD	Data written to the ETB Trace RAM. When trace capture is disabled, the contents of this register are placed into the ETB Trace RAM when this register is written to. Writing to this register increments the RAM Write Pointer Register. If trace capture is enabled, and this register is accessed, then a read from this register outputs `0xFFFFFFFF`. Reads of this register never increment the RAM Write Pointer Register. A constant stream of 1s being output corresponds to a synchronization output from the TPIU. If a write access is attempted, the data is not written into Trace RAM.

10.3.9 ETB Formatter and Flush Status Register, FFSR, 0x300

This register indicates the implemented Trigger Counter multipliers and other supported features of the trigger system.

Figure 10-4 on page 10-12 shows the ETB Formatter and Flush Status Register bit assignments.

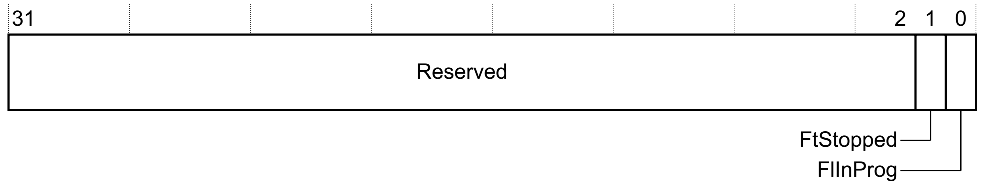


Figure 10-4 ETB Formatter and Flush Status Register bit assignments

Table 10-13 shows the ETB Formatter and Flush Status Register bit assignments.

Table 10-13 ETB Formatter and Flush Status Register bit assignments

Bits	Type	Name	Description
[31:2]	-	-	Reserved RAZ/SBZP.
[1]	RO	FtStopped	Formatter stopped. The formatter has received a stop request signal and all trace data and post-amble has been output. Any further trace data on the ATB interface is ignored and **ATREADYS** goes HIGH.
[0]	RO	FlInProg	Flush In Progress. This is an indication of the current state of **AFVALIDS**.

10.3.10 ETB Formatter and Flush Control Register, FFCR, 0x304

This register indicates the stop, trigger, and flush events. Figure 10-5 shows the ETB Formatter and Flush Control Register bit assignments.

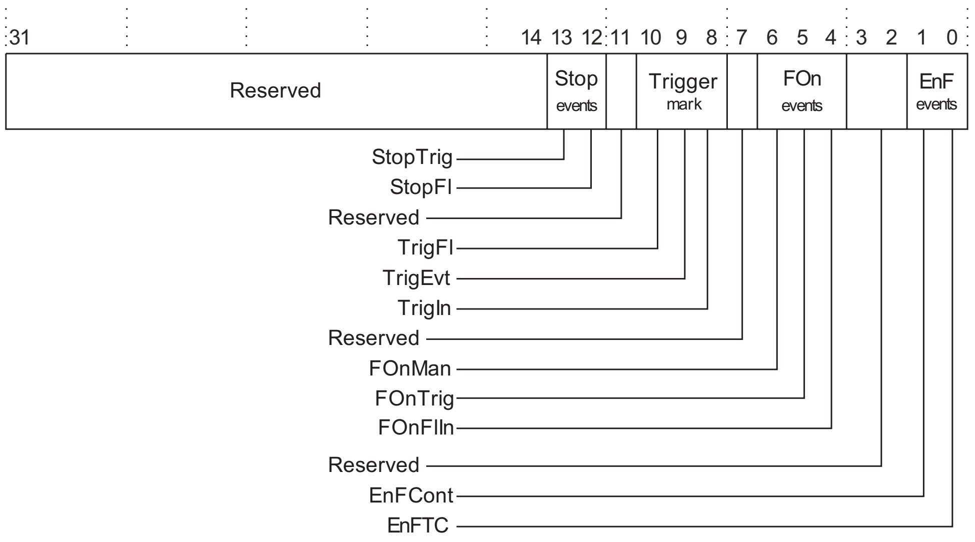


Figure 10-5 ETB Formatter and Flush Control Register bit assignments

Table 10-14 shows the ETB Formatter and Flush Control Register bit assignments.

Table 10-14 ETB Formatter and Flush Control Register bit assignments

Bits	Type	Name	Description
[31:14]	-	-	Reserved RAZ/SBZP.
[13]	R/W	StopTrig	Stop the formatter when a Trigger Event[a] has been observed. Reset to disabled (zero).
[12]	R/W	StopFl	Stop the formatter when a flush has completed (return of **AFREADYS**). This forces the FIFO to drain off any part-completed packets. Setting this bit enables this function but this is clear on reset (disabled).
[11]	-	-	Reserved RAZ/SBZP.
[10]	R/W	TrigFl	Indicate a trigger on Flush completion (**AFREADYS** being returned).
[9]	R/W	TrigEvt	Indicate a trigger on a Trigger Event[a].
[8]	R/W	TrigIn	Indicate a trigger on **TRIGIN** being asserted
[7]	-	-	Reserved RAZ/SBZP.
[6]	R/W	FOnMan	Manually generate a flush of the system. Setting this bit causes a flush to be generated. This is cleared when this flush has been serviced. This bit is clear on reset.
[5]	R/W	FOnTrig	Generate flush using Trigger event. Set this bit to cause a flush of data in the system when a Trigger Event[a] occurs. This bit is clear on reset.
[4]	R/W	FOnFlIn	Generate flush using the **FLUSHIN** interface. Set this bit to enable use of the **FLUSHIN** connection. This bit is clear on reset.
[3:2]	-	-	Reserved RAZ/SBZP.
[1]	R/W	EnFCont	Continuous Formatting. Continuous mode in the ETB corresponds to normal mode with the embedding of triggers. Can only be changed when FtStopped is HIGH. This bit is clear on reset.
[0]	R/W	EnFTC	Enable Formatting. Do not embed Triggers into the formatted stream. Trace disable cycles and triggers are indicated by **TRACECTL**, where fitted. Can only be changed when FtStopped is HIGH. This bit is clear on reset.

a. A Trigger Event is defined as when the Trigger counter reaches zero (where fitted) or, in the case of the trigger counter being zero (or not fitted), when **TRIGIN** is HIGH.

To disable formatting and put the formatter into bypass mode, bits 1 and 0 must be clear. If both bits are set, then the formatter inserts triggers into the formatted stream.All three flush generating conditions can be enabled together. However, if a second or third flush event is generated then the current flush completes before the next flush is serviced. Flush from **FLUSHIN** takes priority over flush from Trigger, which in turn completes before a manually activated flush.All trigger indication conditions can be enabled simultaneously although this can cause the appearance of multiple triggers if flush using trigger is also enabled.

Both 'Stop On' settings can be enabled although if flush on trigger, FOnTrig, is set then none of the flushed data is stored. When the system stops, it returns **ATREADY** and does not store the accepted data packets. This is to stop stalling of any other connected devices using a Trace Replicator.

10.3.11 ETB Integration Test Registers

Integration Test Registers are provided to simplify the process of verifying the integration of the ETB with other devices in a CoreSight system. These registers enable direct control of outputs and the ability to read the value of inputs. You must only use these registers when the Integration Test Control Register bit [0] is set to 1. Before you use these registers, you must disable the trace capture function of the ETB Control Register by setting bit [0] LOW, and setting the FtStopped bit in the Formatter and Flush Status register HIGH.

The registers in the ETB enable the system to set the **FULL** and **ACQCOMP** output pins on the ETB, in addition to the **FLUSHINACK** and **TRIGINACK** output pins. The **FLUSHIN** and **TRIGIN** inputs to the ETB can also be read. The other Integration Test Registers are for testing the integration of the ATB slave interface on the ETB. For details of how to use these signals see the applicable CoreSight DK Integration Manual.

This section describes the following registers:

- *Integration Test Miscellaneous Output Register 0, ITMISCOP0, 0xEE0* on page 10-15
- *Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4* on page 10-15
- *Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8* on page 10-16
- *Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC* on page 10-17
- *Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 10-17
- *Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4* on page 10-18
- *Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 10-19.

Integration Test Miscellaneous Output Register 0, ITMISCOP0, 0xEE0

The Integration Test Miscellaneous Output Register 0 controls the values of some outputs from the ETB. Figure 10-6 shows the bit assignments.

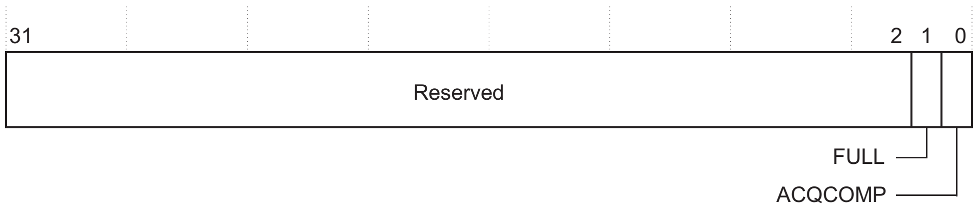


Figure 10-6 Integration Test Miscellaneous Output Register 0 bit assignments

Table 10-15 shows the bit assignments.

Table 10-15 Integration Test Miscellaneous Output Register 0 bit assignments

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	WO	FULL	Set the value of **FULL**
[0]	WO	ACQCOMP	Set the value of **ACQCOMP**

Integration Test Trigger In and Flush In Acknowledge Register, ITTRFLINACK, 0xEE4

The Integration Test Trigger In and Flush In Acknowledge Register enables control of the **TRIGINACK** and **FLUSHINACK** outputs from the ETB. Figure 10-7 shows the bit assignments.

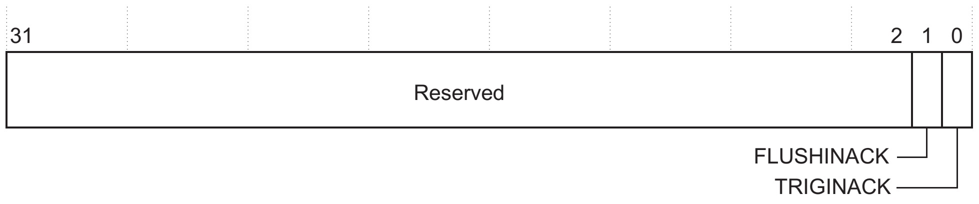


Figure 10-7 Integration Test Trigger In and Flush In Acknowledge Register bit assignments

Table 10-16 shows the bit assignments.

Table 10-16 Integration Test Trigger In and Flush In Acknowledge Register bit assignments

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	WO	FLUSHINACK	Set the value of **FLUSHINACK**
[0]	WO	TRIGINACK	Set the value of **TRIGINACK**

Integration Test Trigger In and Flush In Register, ITTRFLIN, 0xEE8

The Integration Test Trigger In and Flush In Register contains the values of the **FLUSHIN** and **TRIGIN** inputs to the ETB. Figure 10-8 shows the bit assignments.

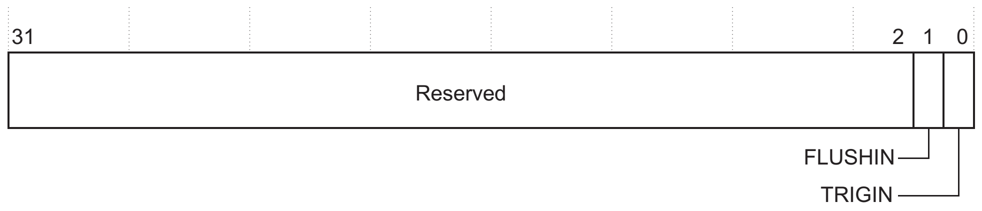


Figure 10-8 Integration Test Trigger In and Flush In Register bit assignments

Table 10-17 shows the bit assignments.

Table 10-17 ETB for CoreSight Integration Register, ITTRFLIN bit assignments

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	RO	FLUSHIN	Read the value of **FLUSHIN**
[0]	RO	TRIGIN	Read the value of **TRIGIN**

Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC

The Integration Test ATB Data Register 0 contains the value of the **ATDATAS** inputs to the ETB. The values are only valid when **ATVALIDS** is HIGH. Figure 10-9 shows the bit assignments.

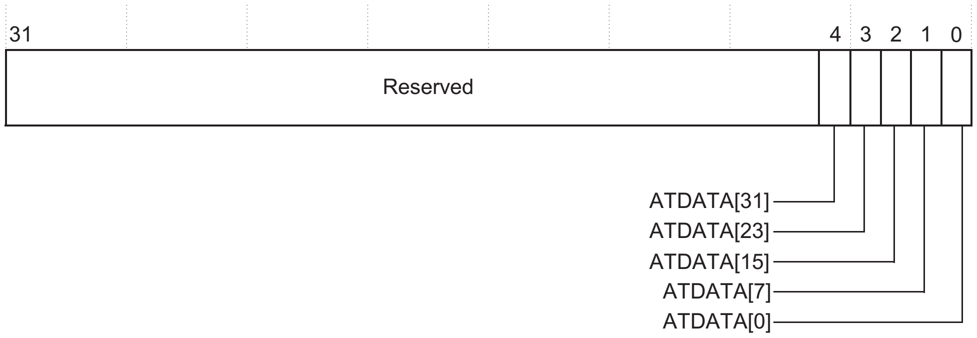


Figure 10-9 Integration Test ATB Data Register 0 bit assignments

Table 10-18 shows the bit assignments.

Table 10-18 Integration Test ATB Data Register 0 bit assignments

Bits	Type	Name	Function
[31:5]	-	-	Reserved
[4]	RO	ATDATA[31]	Read the value of **ATDATAS[31]**
[3]	RO	ATDATA[23]	Read the value of **ATDATAS[23]**
[2]	RO	ATDATA[15]	Read the value of **ATDATAS[15]**
[1]	RO	ATDATA[7]	Read the value of **ATDATAS[7]**
[0]	RO	ATDATA[0]	Read the value of **ATDATAS[0]**

Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0

The Integration Test ATB Control Register 2 enables control of the **ATREADYS** and **AFVALIDS** outputs of the ETB. Figure 10-10 on page 10-18 shows the bit assignments.

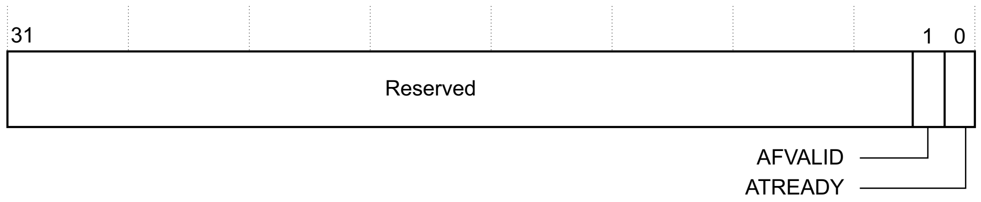


Figure 10-10 Integration Test ATB Control Register 2 bit assignments

Table 10-19 shows the bit assignments.

Table 10-19 Integration Test ATB Control Register 2 bit assignments

Bits	Type	Name	Function
[31:2]	-	-	Reserved
[1]	WO	AFVALID	Set the value of **AFVALIDS**
[0]	WO	ATREADY	Set the value of **ATREADYS**

Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4

The Integration Test ATB Control Register 1 contains the value of the **ATIDS** input to the ETB. This is only valid when **ATVALIDS** is HIGH. Figure 10-11 shows the bit assignments.

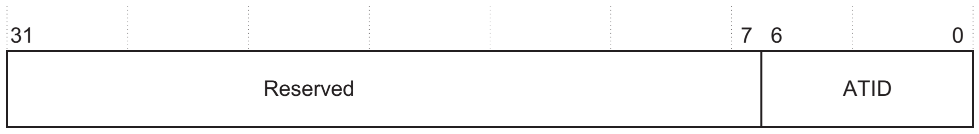


Figure 10-11 Integration Test ATB Control Register 1 bit assignments

Table 10-20 shows the bit assignments.

Table 10-20 Integration Test ATB Control Register 1 bit assignments

Bits	Type	Name	Function
[31:7]	-	-	Reserved
[6:0]	RO	ATID	Read the value of **ATIDS**

Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8

The Integration Test ATB Control Register 0 captures the values of the **ATVALIDS**, **AFREADYS**, and **ATBYTESS** inputs to the ETB. To ensure the integration registers work correctly in a system, the value of **ATBYTESS** is only valid when **ATVALIDS**, bit [0], is HIGH. Figure 10-12 shows the bit assignments.

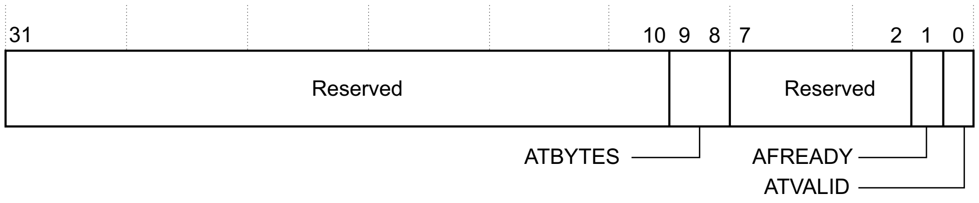


Figure 10-12 Integration Test ATB Control Register 0 bit assignments

Table 10-21 shows the bit assignments.

Table 10-21 Integration Test ATB Control Register 0 bit assignments

Bits	Type	Name	Function
[31:10]	-	-	Reserved
[9:8]	RO	ATBYTES	Read the value of **ATBYTESS**
[7:2]	-	-	Reserved
[1]	RO	AFREADY	Read the value of **AFREADYS**
[0]	RO	ATVALID	Read the value of **ATVALIDS**

10.4 ETB CoreSight management registers

These registers are defined in Chapter 2 *CoreSight Programmer's Model*. This section gives information specific to the ETB programmable registers.

Authentication Status, 0xFB8 [7:0]

Reports the required security level. This is set to `0x00` for functionality not implemented.

Device ID, 0xFC8 The CSTF has the default value `0x00`.

Table 10-22 shows the Device ID bit assignments.

Table 10-22 CSTF Device ID bit assignments

Bits	Value	Description
[31:6]	`0x0000000`	Reserved.
[5]	1'b0	Indicates that the ETB RAM operates synchronously to **ATCLK**.
[4:0]	`5'b0000`	Hidden Level of Input multiplexing. When non-zero this value indicates the type/number of ATB multiplexing present on the input to the ATB. Currently only `0x00` is supported (no multiplexing present). This value is used to assist topology detection of the ATB structure.

Device Type Identifier, 0xFCC [7:0]

`0x21` indicates this device is a trace sink and specifically an ETB.

PartNumber, 0xFE4 [3:0], 0xFE0 [7:4], 0xFE0 [3:0]

Upper, middle, and lower BCD value of Device number. Set to `0x907`.

10.5 ETB clocks, resets, and synchronization

This section describes ETB clocks, resets, and synchronization.

10.5.1 ETB clock domains

The ETB has two clock domains:

PCLKDBG	Drives the APB interface and selected control registers.
ATCLK	Drives the ATB interface, all control logic, and RAM.

10.5.2 ETB resets

ATRESETn resets all of the ETB registers in the **ATCLK** domain. **ATRESETn** must be synchronized externally in the CoreSight infrastructure by a global CoreSight reset signal that asserts **ATRESETn** asynchronously and deasserts **ATRESETn** synchronously with **ATCLK**.

PRESETDBGn resets the APB interface of the ETB. **PRESETDBGn** must be synchronized externally in the CoreSight infrastructure by a global CoreSight reset signal that asserts **PRESETDBGn** asynchronously and deasserts **PRESETDBGn** synchronously with **PCLKDBG**.

10.5.3 ETB synchronization

ATCLK and **PCLKDBG** are synchronous domains.

——— **Note** ———

PCLKDBG must be equivalent to, or an integer division of **ATCLK**.

———————————

10.6 ETB Trace capture and formatting

The formatter inserts the source ID signal **ATIDS[6:0]** into a special format data packet stream to enable trace data to be re-associated with a trace source after data is read back out of the ETB.

10.6.1 Formatter data processing

Figure 10-13 shows the arrangement of four words and organization of the data packets.

When a trace source is changed the appropriate flag bit, F, is set (1 = ID, 0 = Data on the first byte). The second byte is always data. The corresponding bit at the end of the sequence (bits A to J) indicates if this second byte corresponds to the new ID (bit clear) or the previous ID (bit set).

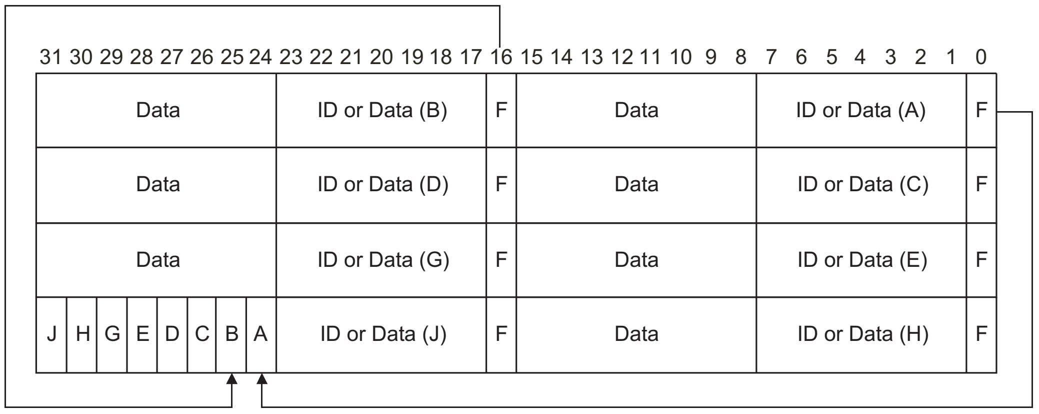


Figure 10-13 Construction of data packets within the formatter

10.6.2 Special Trace Source IDs

The following IDs are set aside for special purposes and must not be used under normal operation:

0x00	This indicates a NULL trace source. This is used to identify unused data packets within the formatter so that incomplete packets can be stored into RAM when data capture ceases.
0x70-0x7C	Reserved.
0x7D	Allocated for indication of a trigger within the trace stream.

0x7E	Reserved.
0x7F	Reserved. This ID must never be used as a trace source ID because it can cause issues with correctly detecting the synchronization packet.

10.6.3 Special modes of operation

The Formatter supports two distinct modes of operation as specified by bits [1:0] in the Formatter and Flush Control Register:

Bypass In this mode, no formatting information is inserted into the trace stream and a raw reproduction of the incoming trace stream is stored. If any bytes remain in the formatter when tracing is stopped, because of non 32-bit **ATDATAS**, then the word stored to RAM is appended with a single logic 1 bit and filled in with zeros. A second word, `0x00000000` is then stored. If no bytes remain in the formatter, when capture is stopped, four additional words are stored, `0x00000001` then three words of `0x00000000`.

When data is later decompressed it is then possible to determine that a post-amble is present by back tracking the trailing zero data at then end of the trace stream until the last single bit at logic 1 is detected. All data preceding this first logic 1 is then treated as decompressible data. When all data has been stored in the RAM, FtStopped in the Formatter and Flush Status Register is set HIGH.

Note

This mode assumes that the source ID is not changing.

Normal Normal mode has the option of embedding triggers into the data packets.

Formatting information is added to indicate the change of source ID along with the associated wrapping additions, as described in *TPIU formatter and FIFO* on page 9-34. If any data remains in the formatter when tracing is stopped then the formatter is filled with NULL packets (ID = `0x00`) to ensure all data is stored in RAM. When all remaining data has been stored in the RAM, FtStopped in the Formatter and Flush Status Register is set HIGH.

A third mode, Continuous, is designed primarily to be used by the TPIU. Continuous mode in the ETB corresponds to normal mode with the embedding of triggers.

10.6.4 Stopping tracing

The stopping of tracing is indicated by FtStopped HIGH in the Formatter and Flush Status Register. This is set by:

- TraceCaptEn = 0 in the Control Register.
- A trigger event occurring, **TRIGIN** goes HIGH and the Trigger Counter Register reaches zero, and StopTrig set HIGH in the Formatter and Flush Control Register (`0x304`).
- A flush completing and StopFl set HIGH in the Formatter and Flush Control Register.

Figure 10-14 on page 10-25 shows a flowchart defining the conditions for stopping trace capture.

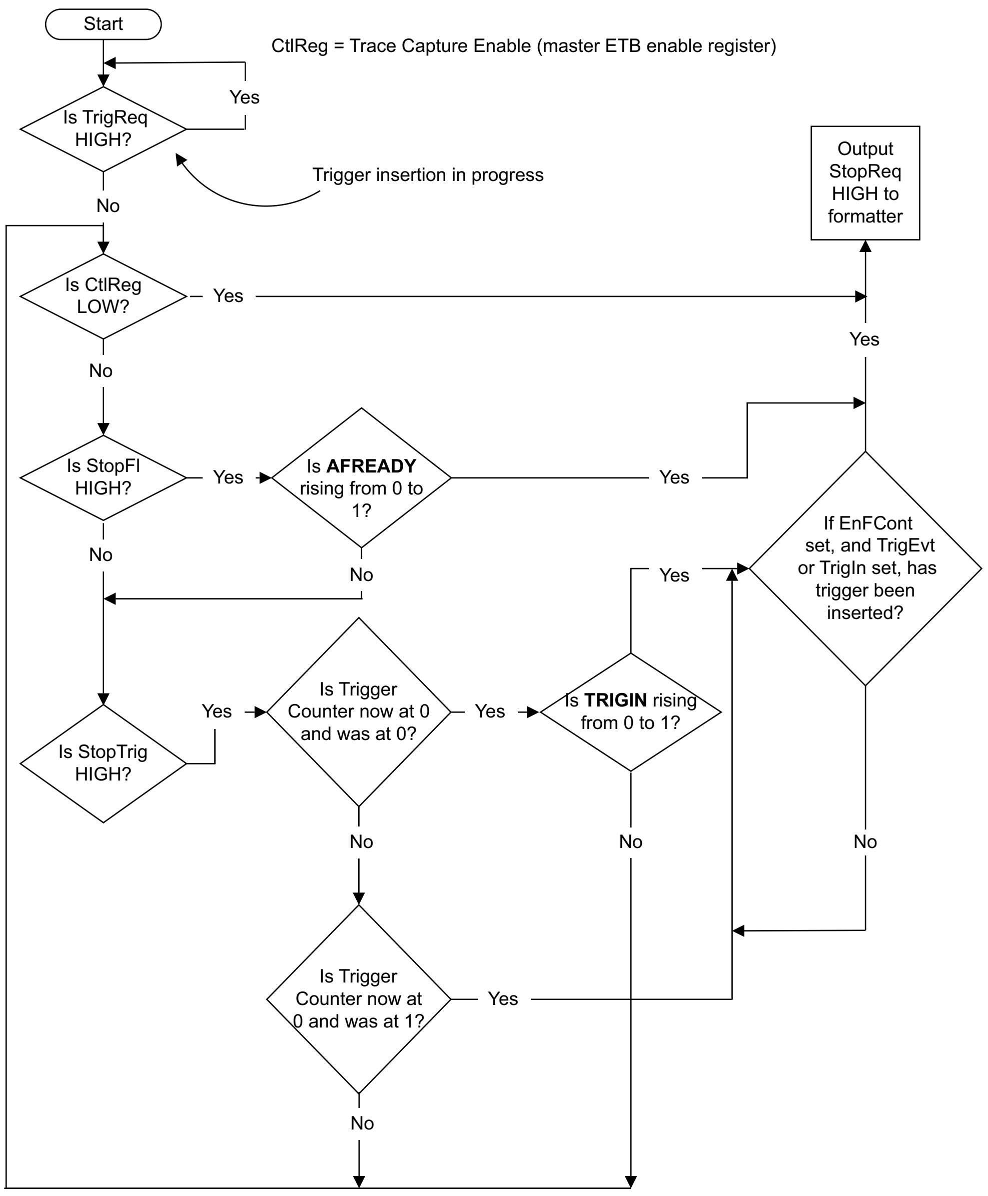

Start
CtlReg = Trace Capture Enable (master ETB enable register)
Is TrigReq HIGH?
Yes
Trigger insertion in progress
No
Output StopReq HIGH to formatter
Is CtlReg LOW?
Yes
No
Yes
Is StopFl HIGH?
Yes
Is **AFREADY** rising from 0 to 1?
Yes
No
No
If EnFCont set, and TrigEvt or TrigIn set, has trigger been inserted?
Yes
Is StopTrig HIGH?
Yes
Is Trigger Counter now at 0 and was at 0?
Yes
Is **TRIGIN** rising from 0 to 1?
No
No
No
Is Trigger Counter now at 0 and was at 1?
Yes
No

Figure 10-14 Conditions for stopping trace capture

StopTrig and StopFl in the Formatter and Flush Control Register can be enabled at the same time. For example, if FOnTrig in the Formatter and Flush Control Register is set to perform a flush on a trigger event, but StopTrig is HIGH, none of the flushed data is stored, because StopTrig is HIGH. If the situation requires that all the flushed data is captured StopTrig must be LOW and StopFl must be HIGH in this situation.

When the formatter stops, **ATREADYS** is output HIGH, to prevent an ATB bus from stalling which is important when a replicator is present, but the received data is ignored.

10.7 Flush assertion

All three flush generating conditions can be enabled together:

- flush from **FLUSHIN**
- flush from Trigger
- manually activated flush.

If more flush events are generated while a flush is in progress, the current flush is serviced before the next flush is started. If the waiting flush signal is deasserted in the meantime it is not serviced, that is, the state pending flushes is only analyzes when the current flush has completed.

Flush from **FLUSHIN** takes priority over Flush from Trigger, which in turn is completed before a manually activated flush. The operation of the flush mechanism is defined in the *CoreSight Architecture Specification*.

Figure 10-15 on page 10-28 shows a flowchart defining the conditions for generating a flush on the ATB interface.

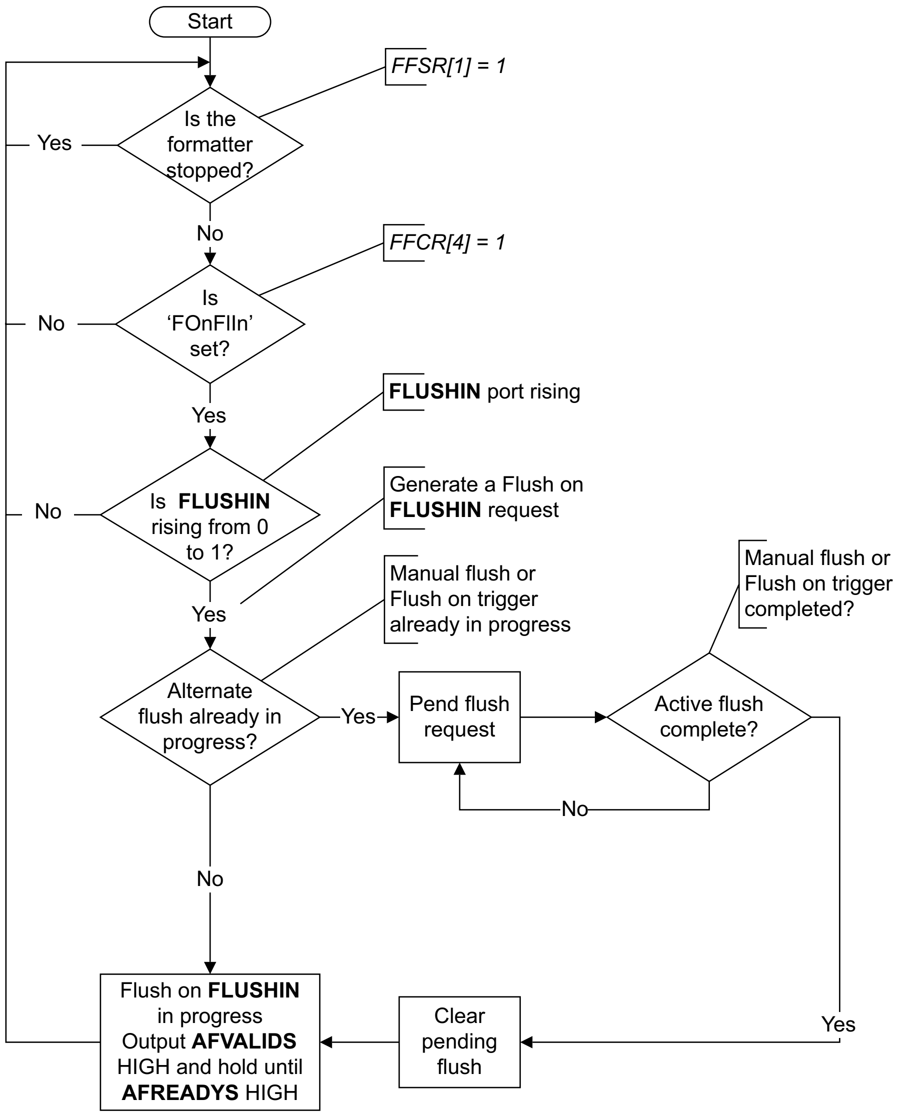

Start
FFSR[1] = 1
Is the formatter stopped?
Yes
No
FFCR[4] = 1
Is 'FOnFlIn' set?
No
Yes
FLUSHIN port rising
Is FLUSHIN rising from 0 to 1?
No
Generate a Flush on FLUSHIN request
Yes
Manual flush or Flush on trigger already in progress
Alternate flush already in progress?
Yes
Pend flush request
Manual flush or Flush on trigger completed?
Active flush complete?
No
No
Flush on FLUSHIN in progress Output AFVALIDS HIGH and hold until AFREADYS HIGH
Clear pending flush
Yes

Figure 10-15 Generation of flush on FLUSHIN

Figure 10-16 on page 10-29 shows generation of flush from a trigger.

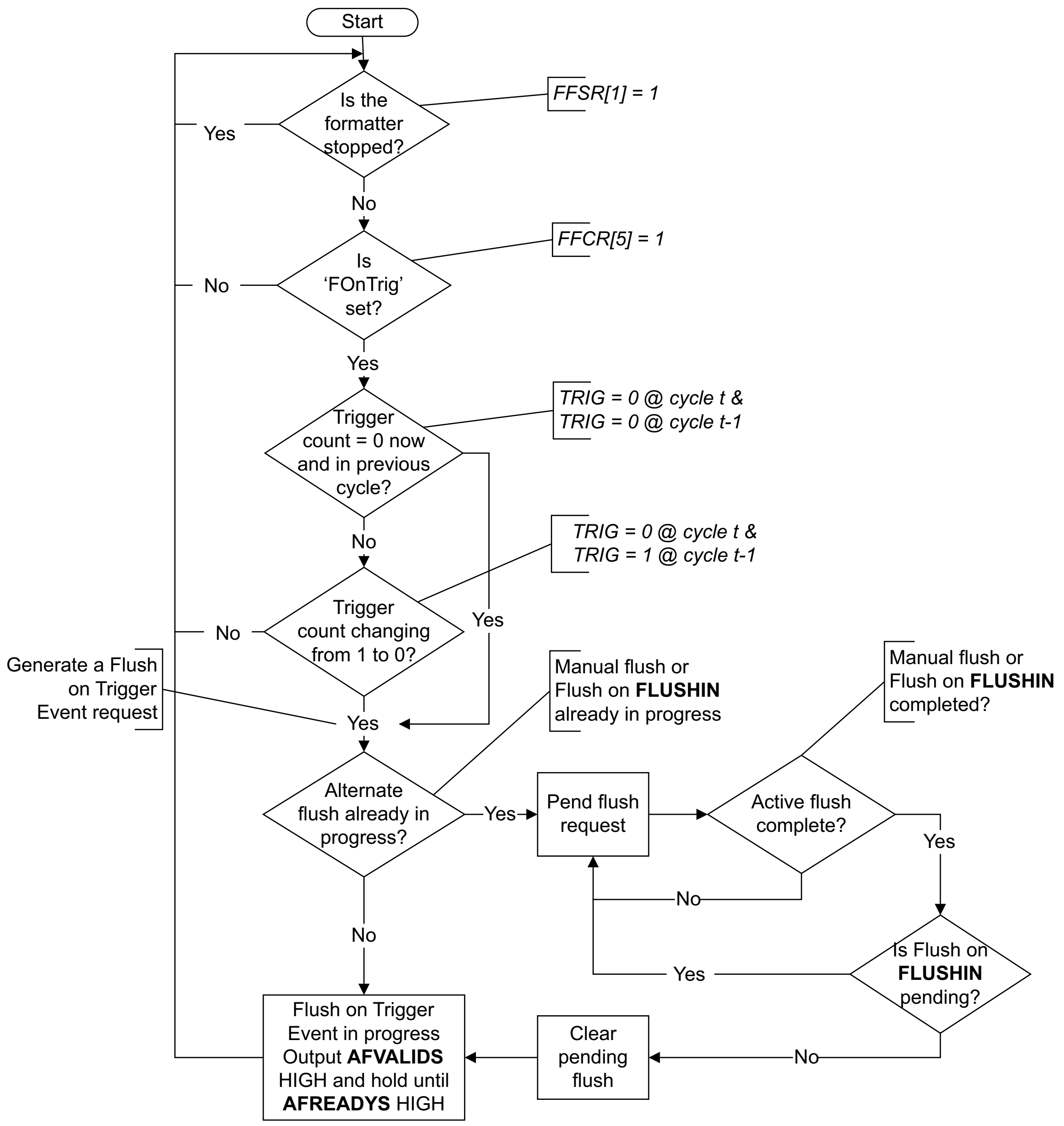

Start
Is the formatter stopped?
FFSR[1] = 1
Yes
No
Is 'FOnTrig' set?
FFCR[5] = 1
No
Yes
Trigger count = 0 now and in previous cycle?
TRIG = 0 @ cycle t &
TRIG = 0 @ cycle t-1
No
Yes
Trigger count changing from 1 to 0?
TRIG = 0 @ cycle t &
TRIG = 1 @ cycle t-1
No
Yes
Generate a Flush on Trigger Event request
Manual flush or Flush on FLUSHIN already in progress
Alternate flush already in progress?
Yes
Pend flush request
Active flush complete?
Manual flush or Flush on FLUSHIN completed?
Yes
No
Is Flush on FLUSHIN pending?
Yes
No
Clear pending flush
Flush on Trigger Event in progress Output AFVALIDS HIGH and hold until AFREADYS HIGH

Figure 10-16 Generation of flush from a trigger event

Figure 10-17 on page 10-30 shows generation of flush from manual.

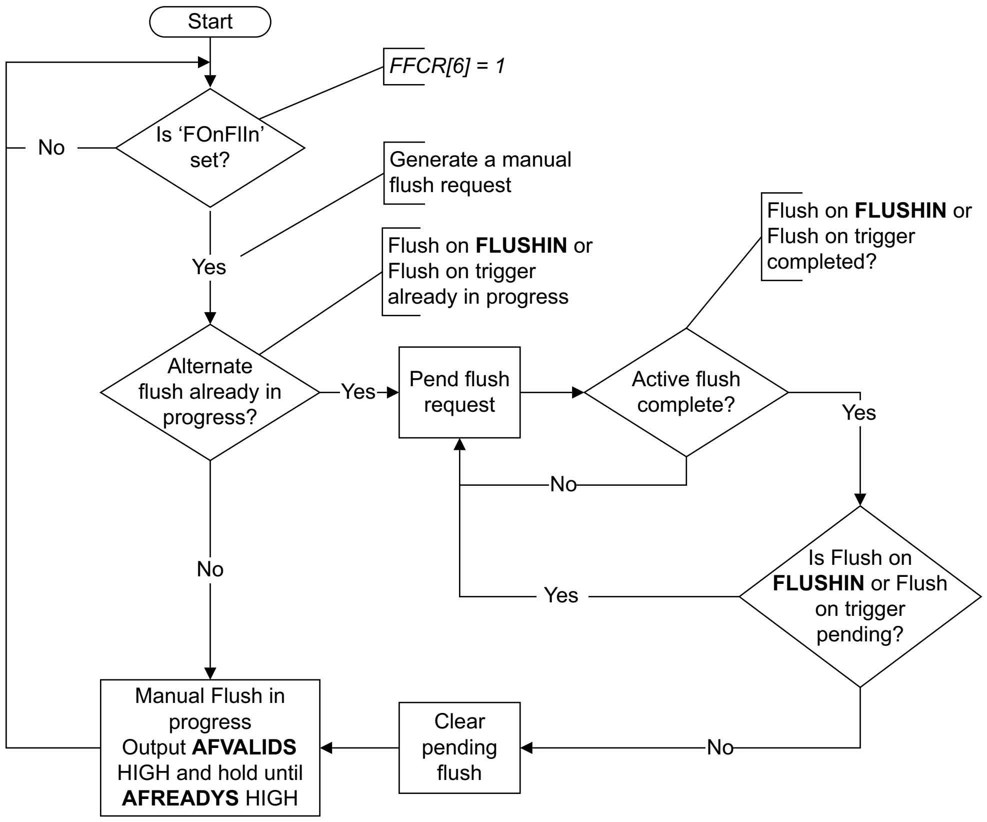

Start
FFCR[6] = 1
Is 'FOnFlIn' set?
No
Generate a manual flush request
Yes
Flush on **FLUSHIN** or Flush on trigger already in progress
Flush on **FLUSHIN** or Flush on trigger completed?
Alternate flush already in progress?
Yes
Pend flush request
Active flush complete?
Yes
No
No
Is Flush on **FLUSHIN** or Flush on trigger pending?
Yes
Manual Flush in progress Output **AFVALIDS** HIGH and hold until **AFREADYS** HIGH
Clear pending flush
No

Figure 10-17 Generation of a flush on manual

10.8 Triggers

A trigger event is defined as when the Trigger Counter reaches zero, or when the trigger counter is zero, when **TRIGIN** is HIGH. All trigger indication conditions, TrigEvt, TrigFl, and TrigIn, in the Formatter and Flush Control Register can be enabled simultaneously. This results in multiple triggers appearing in the trace stream.

The trigger counter register controls how many words are written into the Trace RAM after a trigger event. After the formatter is flushed in normal or continuous mode a complete empty frame is generated. This is a data overhead of seven extra words in the worst case. The trigger counter defines the number of 32-bit words remaining to be stored in the ETB Trace RAM. If the formatter is in bypass mode, a maximum of two additional words are stored for the trace capture post-amble. See *Formatter and Flush Control Register, 0x304* on page 9-16.

Figure 10-18 on page 10-32 shows a flowchart defining the conditions for indicating a trigger in the formatted data. This flowchart only applies for the condition when continuous formatting is enabled (EnFCont HIGH) in the Formatter and Flush Control Register.

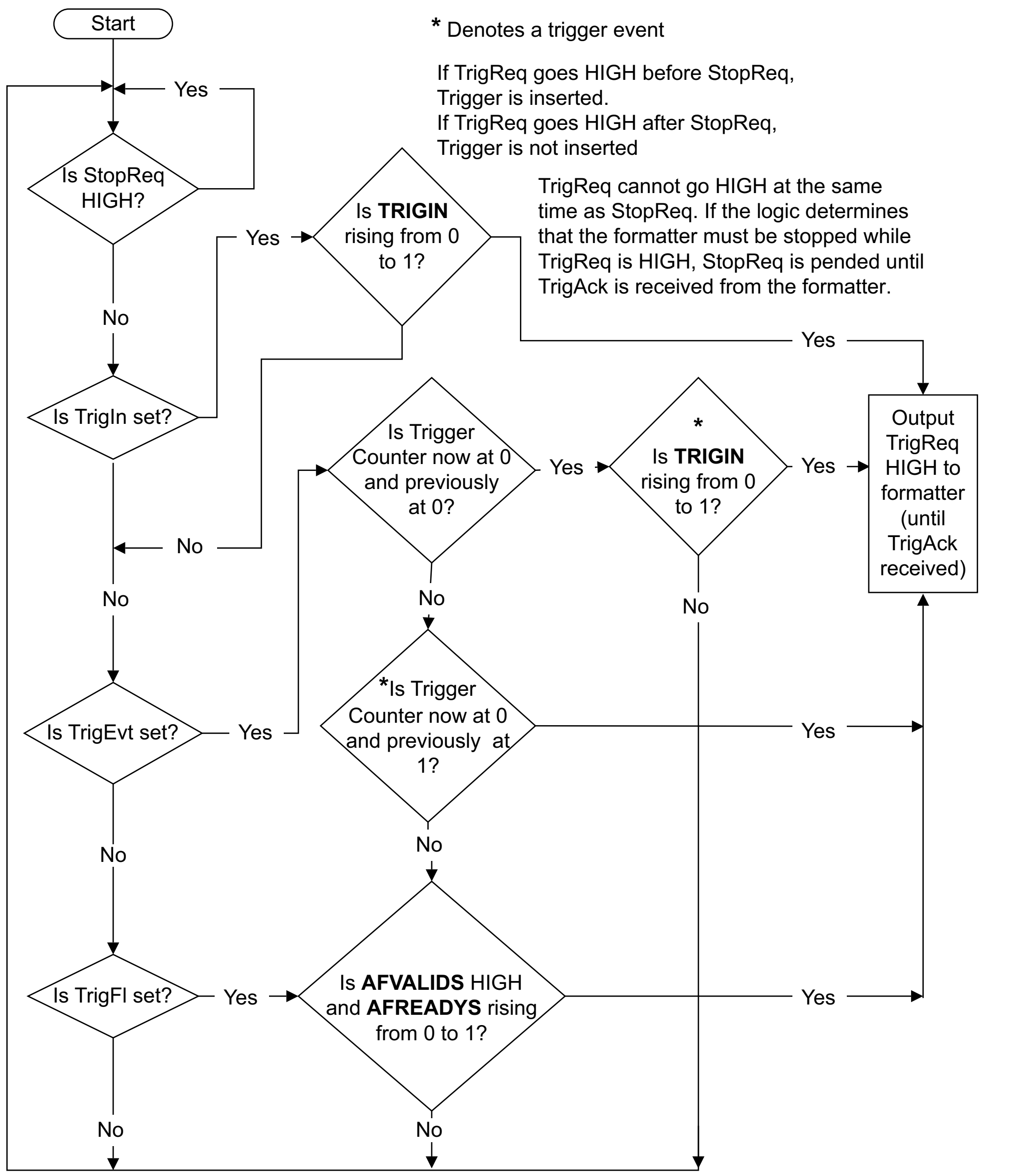

Start
* Denotes a trigger event
If TrigReq goes HIGH before StopReq, Trigger is inserted.
If TrigReq goes HIGH after StopReq, Trigger is not inserted
TrigReq cannot go HIGH at the same time as StopReq. If the logic determines that the formatter must be stopped while TrigReq is HIGH, StopReq is pended until TrigAck is received from the formatter.
Is StopReq HIGH?
Yes
No
Is TrigIn set?
Yes
Is TRIGIN rising from 0 to 1?
Yes
No
No
Is Trigger Counter now at 0 and previously at 0?
Yes
* Is TRIGIN rising from 0 to 1?
Yes
No
Output TrigReq HIGH to formatter (until TrigAck received)
No
Is TrigEvt set?
Yes
*Is Trigger Counter now at 0 and previously at 1?
Yes
No
No
Is TrigFl set?
Yes
Is AFVALIDS HIGH and AFREADYS rising from 0 to 1?
Yes
No
No

Figure 10-18 Generation of a trigger request with continuous formatting enabled

10.9 Write address generation for trace data storage

The RAM Write Pointer is automatically selected for addressing the Trace RAM when trace capture is enabled in the Control Register. Data is written into the Trace RAM when the formatter asserts the DataValid flag on generation of a formatted word. On completion of a write access to the Trace RAM the register is automatically incremented.

The RAM Write Pointer Register must be programmed before trace capture is enabled.

10.10 Trace data storage

Data is received by the ETB ATB Formatter block. When TraceCaptEn is asserted the Formatter starts the process of embedding the source IDs into the data stream, or normal mode, then outputs 32-bit words to be stored in the ETB Trace RAM. When data is ready to be stored in the Trace RAM, data is written to at the address stored in the RAM Write Pointer Register. When the Trigger Counter Register reaches zero, the acquisition complete flag AcqComp is asserted and trace capture is stopped.

10.11 APB configuration and RAM access

This section describes APB configuration and RAM accesses:

- *Read access*
- *Write access*.

10.11.1 Read access

When trace capture is disabled (TraceCaptEn=0), the RAM Read Pointer is used to generate the RAM address for reading data stored in the ETB Trace RAM. Updating the RAM Read Pointer automatically triggers a RAM access to ensure all RAM data present in the RAM Read Data Register is up-to-date.

The RAM Read Pointer is updated either by writing directly to it, or performing a read of the RAM Read Data Register, causing the RAM Read Pointer to be incremented. Therefore, a read of the RAM Read Data Register consequentially causes the next memory location to be read and loaded into the RAM Read Data Register.

10.11.2 Write access

When trace capture is disabled (TraceCaptEn=0), the RAM Write Pointer Register is used to generate the RAM address for writing data from the RAM Read Data Register. Compare this with trace capture is enabled, and trace data storage from the ATB interface. The data to be stored in the RAM Write Data Register is derived from an APB write transfer to the APB interface.

Updating the RAM Read Data Register automatically triggers a RAM access to write the register contents into the Trace RAM at the address present in the RAM Write Pointer Register. On completion of the write cycle the RAM Write Pointer Register is automatically incremented.

Note

A write access to the RAM Write Pointer Register does not initiate a write transfer to the Trace RAM, only a write to the RAM Read Data Register causes a RAM write.

The RAM Write Pointer Register must be programmed before trace capture is re-enabled.

10.12 Trace RAM

See the *CoreSight Components Implementation Guide* for information about Trace RAM and RAM integration.

10.13 Authentication requirements for CoreSight ETBs

CoreSight ETBs do not require any inputs capable of disabling them.

10.13.1 Access sizes

Byte writable RAMs are not supported. Trace storage is only in word accesses, and APB accesses take place through the RAM Write Data Register, again 32 bits. The APB interface has no concept of data size.

10.13.2 BIST interface

The RAM BIST interface connects though the Trace RAM Interface block to provide test access to the Trace RAM. **MTESTON** enables the memory BIST interface and disables all other accesses to and from the RAM.

10.13.3 RAM instantiation

As shown in Figure 10-19, the CSEtbRam block provides the wrapper in which the RAM simulation model/hardened cell can be instantiated. The active level of Chip Enable, Write Enable can be modified in this block.

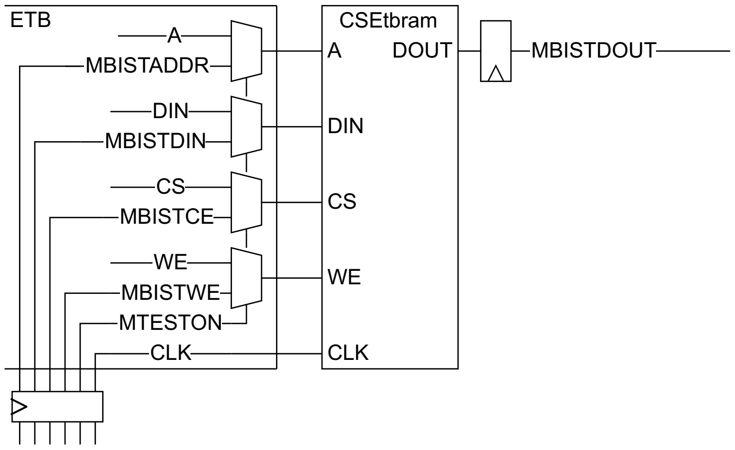


Figure 10-19 ETB trace RAM block wrapper

10.14 ETB configuration options

This section describes:

- *ETM architecture version no longer required*
- *RAM Size Configuration*.

10.14.1 ETM architecture version no longer required

ETM architecture version configuration is no longer required on the Formatter input. The ETB is now trace data agnostic, with formatting carried out in a consistent manner, with optional bypass, as defined in the *CoreSight Architecture Specification*.

10.14.2 RAM Size Configuration

RAM width is 32 bits. This minimizes buffering in the formatter and ensures that the Trace RAM can be used as slow software-accessible memory.

The RAM depth is configurable by setting the address bus width using CSETB_RAM_ADRW, as shown in Table 10-23. Previous implementations of the ETB did not specify a lower and upper range of RAM configurations. ARM currently recommends customers adopt a minimum RAM size of 4KB. The range specified covers both ends of requirements, and meets future trace storage requirements.

Table 10-23 ETB RAM size options

Address range	Words stored	Total storage in KB
[7:0]	256	1
[8:0]	512	2
[9:0]	1024	4
[10:0]	2048	8
[11:0]	4096	16
[12:0]	8192	32
[13:0]	16384	64
[14:0]	32768	128
[15:0]	65536	256
[16:0]	131072	512
[17:0]	262144	1024

The ETB adopts the methodology used to generate and configure the top level CoreSight system to support the configuration and generation of RAM size related parameters, as specified by CSETB_RAM_ADRW which defines the address range

10.15 Comparisons with ETB11

Table 10-24 shows the differences and similarities between ETB11 and CoreSight ETB.

Table 10-24 ETB11 and ETB comparison

ETB11 feature	Changes in ETB	Comments	References
JTAG Port	Does not exist	Access is through a single APB bus for memory access and configuration	See *ATB interface* on page 10-3.
Programmer's Model	Integration into standard PrimeCell peripheral ID register structure	Existing registers maintain their current location where required	*ETB register descriptions* on page 10-7.
		Existing registers are modified to support the peripheral ID register structure where required	*ETB register descriptions* on page 10-7.
AHB Interface	Port is replaced with an AMBA 3 APB interface	Trace RAM is no longer memory-mapped.	See *ATB interface* on page 10-3.
		Access is through a single APB bus for memory access and configuration	
ETM Trace Interface	Replacement with an ATB-compliant trace input	-	See *ETB Trace capture and formatting* on page 10-22.
	Replacement of the Data Formatter block with CoreSight-compliant formatter unit	-	
ETB Trace RAM	Support for 32-bit RAM only	-	-
	Word-width access only required.	-	-
	Direct RAM access removed.	-	-
	Use of read and write register pointers, with auto-incrementing addresses, like the original JTAG access methodology	-	-

Chapter 11
Serial Wire Viewer

This chapter describes the *Serial Wire Viewer* (SWV). It contains the following sections:

11.1 About the Serial Wire Viewer

The *Instrumentation Trace Macrocell* (ITM) and *Serial Wire Output* (SWO) can be used to form a SWV, as shown in Figure 11-1.

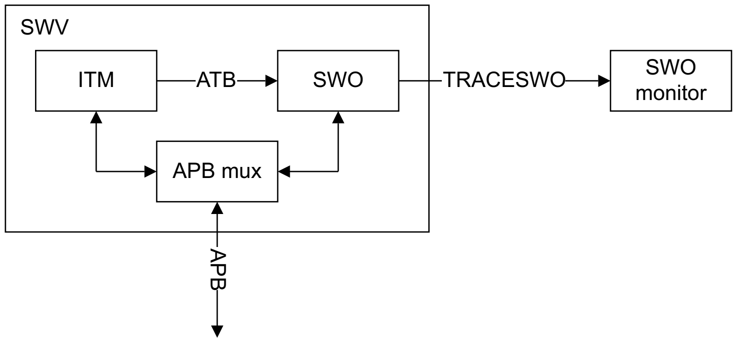


Figure 11-1 ITM and SWO as part of a SWV system

When the ITM and SWO are used as part of the SWV:

- if **PADDRDBG[12]** is set LOW, the ITM registers are accessed
- if **PADDRDBG[12]** is set HIGH, the SWO registers are accessed.

The **ATIDM** and **AFREADYM** outputs from the ITM are left unconnected. These signals do not provide any useful information, because the ITM is directly connected to the SWO. Because of this, it is not necessary to write to the ITM Integration Register, ITATBCTR1, for integration testing.

——— **Note** ———

For more information on the ITM and SWO, see Chapter 13 *Instrumentation Trace Macrocell* and Chapter 12 *Serial Wire Output*.

11.2 SWV ports

The SWV ports are described in:

- *Clocks and resets*
- *Trace interface ports*
- *APB interface* on page 11-4
- *Authentication interface ports* on page 11-4
- *Miscellaneous ports* on page 11-5.

11.2.1 Clocks and resets

Table 11-1 shows the clocks and resets.

Table 11-1 Clocks and resets

Name	Type	Description
PCLKDBG	Input	Debug APB clock, synchronous to **ATCLK**
PCLKENDBG	Input	Debug APB clock enable, tied HIGH if not required
PRESETDBGn	Input	Debug APB reset, active LOW
ATCLK	Input	Trace bus clock
ATCLKEN	Input	Trace bus clock enable, tied HIGH if not required
ATRESETn	Input	Trace bus reset, active LOW

11.2.2 Trace interface ports

Table 11-2 shows the trace interface ports.

Table 11-2 Trace interface ports

Name	Type	Description
TRACECLKIN	Input	Decoupled clock from ATB to enable easy ramping up of the trace port speed. This is typically from a controllable clock source on the processor, but can be driven by an external clock generator if a high speed pin is used. Data changes on the rising edge only.
TRESETn	Input	Reset for TRACECLKIN registers.
TRACESWO	Output	Trace out port over a single pin. Manchester encoded and UART formats are supported.

11.2.3 APB interface

Table 11-3 shows the APB interface ports.

Table 11-3 APB interface ports

Name	Type	Description
PSELDBG	Input	Indicates that a transfer is intended for the ITM interface
PADDRDBG31	Input	Enables lock register bypassing by external debug agents, such as a DAP
PADDRDBG[12:2]	Input	Address bus to select control or stimulus registers
PENABLEDBG	Input	Enable signal to indicate second and subsequent cycles
PWRITEDBG	Input	Write transfer this cycle, and !Read
PWDATADBG[31:0]	Input	Data bus for write transfers
PREADYDBG	Output	Ready signal, pulled LOW to extend transfers
PRDATADBG[31:0]	Output	Data bus for read transfers., only valid on reads when **PREADYDBG** is HIGH

11.2.4 Authentication interface ports

Table 11-4 shows the authentication interface ports.

Table 11-4 Authentication interface ports

Name	Type	Description
DBGEN	Input	Invasive Debug Enable, implies non-invasive
NIDEN	Input	Non-Invasive Debug Enable
SPIDEN	Input	Secure Invasive Debug Enable, implies secure non-invasive
SPNIDEN	Input	Secure Non-Invasive Debug Enable

11.2.5 Miscellaneous ports

Table 11-5 shows the miscellaneous SWV ports.

Table 11-5 Miscellaneous ports

Name	Type	Description
TRIGOUT	Output	Indicates a trigger event, that is, a write to a stimulus register that has a trigger enable against it
TRIGOUTACK	Input	Asynchronous **TRIGOUT** acknowledgement signal
SE	Input	Scan enable.

Chapter 12
Serial Wire Output

This chapter describes the *Serial Wire Output* (SWO). It contains the following sections:

12.1 About the Serial Wire Output

The CoreSight SWO is a trace data drain that acts as a bridge between the on-chip trace data to a data stream that is captured by a *Trace Port Analyzer* (TPA). It is a TPIU-like device that supports a limited subset of the full TPIU functionality, to minimize gate count for a simple debug solution.

Compared to the TPIU, the SWO contains:

- no formatter
- no pattern generator
- an 8-bit ATB input
- no synchronous trace output, that is, no **TRACEDATA**, **TRACECTL**, or **TRACECLK** pins
- no support for flush, because this is not required
- no support for triggering
- no external inputs and outputs (**EXTCTLIN** and **EXTCTLOUT** are not implemented).

Figure 12-1 shows a block diagram of the SWO.

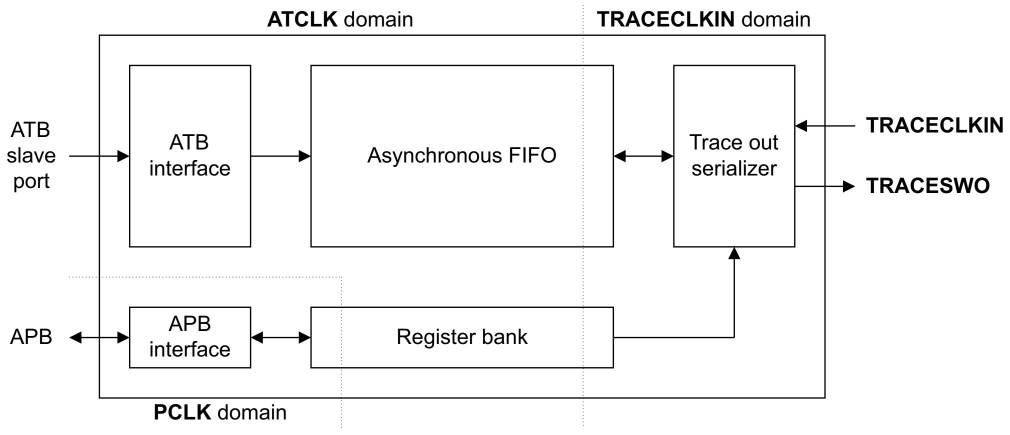


Figure 12-1 SWO block diagram

The CoreSight SWO is designed to be used as part of either:

- a *Serial Wire Viewer* (SWV), as described in Chapter 11 *Serial Wire Viewer*
- a complete CoreSight system, as shown in Figure 12-2 on page 12-3.

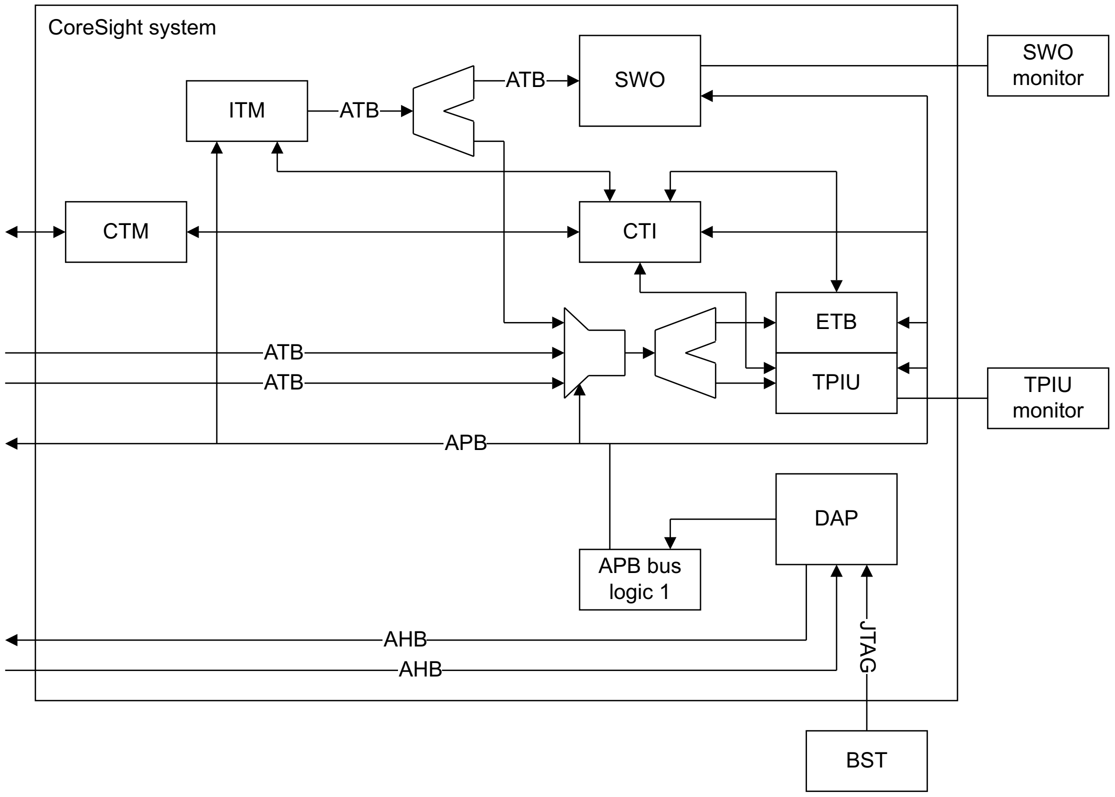


Figure 12-2 SWO as part of a CoreSight system

12.2 SWO ports

The SWO ports are described in:

- *ATB interface*
- *APB interface*
- *Trace out ports* on page 12-5
- *Miscellaneous ports* on page 12-5.

12.2.1 ATB interface

The SWO accepts trace data from a single trace source, the *Instrumentation Trace Macrocell* (ITM), and drains the data out. See the *CoreSight Architecture Specification* for more information. Table 12-1 shows the ATB slave input ports.

Table 12-1 ATB slave input ports

Name	Type	Description
ATCLK	Input	Trace bus clock. Main clock for the internals of the TPIU.
ATCLKEN	Input	ATB clock enable.
ATRESETn	Input	Reset for the **ATCLK** domain.
ATDATAS[7:0]	Input	Trace data 1 byte valid with data always aligned to the LSB.
ATVALIDS	Input	There are valid signals this cycle from the trace source.
ATREADYS	Output	If there is valid data, that is, **ATVALID** HIGH, the data is accepted in this cycle.

12.2.2 APB interface

This is the programming interface for the APB. See the *AMBA 3 APB Protocol* for more information. Table 12-2 shows the APB programming ports.

Table 12-2 APB programming ports

Name	Type	Description
PCLKDBG	Input	Debug APB clock.
PCLKENDBG	Input	Debug APB clock enable.
PRESETDBGn	Input	Debug APB interface reset, including programming registers.
PADDRDBG[11:2]	Input	Programming address. Only a 4k window is required.

Table 12-2 APB programming ports (continued)

Name	Type	Description
PADDRDBG31	Input	HIGH for external accesses to bypass the Lock Access mechanism. Must be wired to **PADDRDBG[31]** in systems.
PSELDBG	Input	This device is currently being accessed.
PENABLEDBG	Input	Indicates second, and subsequent, cycles.
PWRITEDBG	Input	This access is a write transfer.
PWDATADBG[31:0]	Input	Write data bus.
PRDATADBG[31:0]	Output	Read data bus.
PREADYDBG	Output	Used to extend an APB transfer.

12.2.3 Trace out ports

The SWO can output trace data in a number of formats but only one output mechanism is valid at any one time. Table 12-3 shows the trace out ports.

Table 12-3 Trace out ports

Name	Type	Description
TRACECLKIN	Input	Decoupled clock from ATB to enable easy ramping up of the trace port speed. This is typically from a controllable clock source on the processor, but can be driven by an external clock generator if a high speed pin is used. Data changes on the rising edge only.
TRESETn	Input	Reset for TRACECLKIN registers.
TRACESWO	Output	Trace out port over a single pin. Manchester encoded and UART formats are supported.

12.2.4 Miscellaneous ports

Table 12-4 shows the miscellaneous ports.

Table 12-4 Miscellaneous ports

Name	Type	Description
SE	Input	Scan enable.

12.3 SWO programmer's model

This section describes all the visible registers that can be accessed from the APB interface. Table 12-5 shows the SWO programmable registers.

Table 12-5 SWO programmable registers

Offset	Type	Width	Reset value	Name	Description
`0x000`	RO	32	`0x00000000`	Supported Synchronous Port Sizes	See *Trace port control registers* on page 12-11
`0x004`	RO	32	`0x00000000`	Current Synchronous Port Size	
`0x010`	R/W	13	`0x0000`	Current Asynchronous Output Speed Divisors	
`0x0F0`	R/W	2	`0x1`	Selected Pin Protocol	
`0x100`	RO	9	`0x000`	Supported Trigger Modes	See *Trigger registers* on page 12-14
`0x200`	RO	18	`0x00000`	Supported Test Pattern/Modes	See *Test pattern registers* on page 12-16
`0x300`	RO	4	`0x8`	Formatter and Flush Status	See *Formatter and flush registers* on page 12-17
`0x304`	RO	14	`0x0000`	Formatter and Flush Control	
`0xEEC`	RO	2	`0x0`	Integration Test ATB Data Register 0	See *Integration test registers* on page 12-20
`0xEF0`	WO	1	-	Integration Test ATB Control Register 2	
`0xEF8`	RO	1	-	Integration Test ATB Control Register 0	
`0xF00`	R/W	1	`0x0`	Integration Mode Control Register	See *CoreSight defined registers* on page 12-8
`0xFA0`	R/W	4	`0xF`	Claim Tag Set	
`0xFA4`	R/W	4	`0x0`	Claim Tag Clear	
`0xFB0`	WO	32	-	Lock Access	
`0xFB4`	RO	3	`0x3`	Lock Status	
`0xFB8`	RO	8	`0x00`	Authentication status	
`0xFC8`	RO	13	`0x0EA0`	Device ID	
`0xFCC`	RO	8	`0x11`	Device Type Identifier	

Table 12-5 SWO programmable registers (continued)

Offset	Type	Width	Reset value	Name	Description
`0xFD0`	RO	8	`0x04`	Peripheral ID4	See *CoreSight Management Registers summary* on page 2-12
`0xFD4`	RO	8	`0x00`	Peripheral ID5, reserved for future use	
`0xFD8`	RO	8	`0x00`	Peripheral ID6, reserved for future use	
`0xFDC`	RO	8	`0x00`	Peripheral ID7, reserved for future use	
`0xFE0`	RO	8	`0x14`	Peripheral ID0, contains Part Number[7:0]	See on page 12-8*CoreSight defined registers* on page 12-8
`0xFE4`	RO	8	`0xB9`	Peripheral ID1, contains Part Number[12:8]	
`0xFE8`	RO	8	`0x0B`	Peripheral ID2	
`0xFEC`	RO	8	`0x00`	Peripheral ID3	
`0xFF0`	RO	8	`0x0D`	Component ID0	See *CoreSight Management Registers summary* on page 2-12
`0xFF4`	RO	8	`0x90`	Component ID1	
`0xFF8`	RO	8	`0x05`	Component ID2	
`0xFFC`	RO	8	`0xB1`	Component ID3	

12.4 CoreSight defined registers

These registers are described in *CoreSight Management Registers summary* on page 2-12. This section describes the values that are specific to the SWO:

- *Integration Mode Control Register, 0xF00 [0]*
- *Claim Tag Set and Clear Registers, 0xFA0 and 0xFA4*
- *Lock Access Register, 0xFB0 [31:0]*
- *Lock Status Register, 0xFB4 [2:0]* on page 12-9
- *SWO identification registers* on page 12-9.

12.4.1 Integration Mode Control Register, 0xF00 [0]

This register enables the device to switch from a functional mode, or default behavior, into integration mode where the device inputs and outputs can be directly controlled for the purpose of integration testing/topology solving. For the SWO, it is set to `0x1` to enable Integration mode.

12.4.2 Claim Tag Set and Clear Registers, 0xFA0 and 0xFA4

Typically these registers enable the sharing of resources. Because the SWO has no resources to share, all implementations take the minimum size claim tag of 4 bits.

12.4.3 Lock Access Register, 0xFB0 [31:0]

The Lock Access Register is write-only. This register enables write access to component registers. A write of `0xC5ACCE55` enables further write access to this component. An invalid write has the effect of removing write access. This Lock Access Mechanism can be bypassed by setting **PADDRDBG31**, the highest APB Address line, to HIGH. In this case, the entire device can be accessed.

12.4.4 Lock Status Register, 0xFB4 [2:0]

The Lock Status Register indicates the status of the lock control mechanism, and is used in conjunction with the Lock Access Register.

Table 12-6 shows the bit assignments.

Table 12-6 Lock Status Register bit assignments

Bits	Name	Description
[31:3]	-	Reserved RAZ/SBZP.
[2]	8-BIT	Access Lock Register size. This bit reads 0 to indicate a 32-bit register is present.
[1]	STATUS	Lock Status. This bit is HIGH when the device is locked, and LOW when unlocked.
[0]	IMP	Lock mechanism is implemented. This bit always reads 1.

Note

When **PADDRDBG31** is HIGH, the Lock Access Mechanism is bypassed and the Lock Status Register does not reflect the actual status of the Lock Access, and the register reads zero.

12.4.5 SWO identification registers

The identification register values specific to the SWO are:

Authentication Status Register, 0xFB8 [7:0]

This register reports the required security level for operation, and is set to `0x00` for functionality not implemented.

Device ID, 0xFC8

Table 12-7 shows the SWO Device ID Register bit assignments.

Table 12-7 Device ID Register bit assignments

Bits	Value	Description
[31:13]	`0x00000`	Reserved RAZ/SBZP.
[12]	0	STP fitted. In the SWO this is not supported, and reads as 0.
[11]	1	UART/NRZ Serial Wire Output supported. In the SWO this is always supported, and reads as 1.
[10]	1	Manchester Serial Wire Output supported. In the SWO this is always supported, and reads as 1.

Table 12-7 Device ID Register bit assignments (continued)

Bits	Value	Description
[9]	1	Trace clock and data not supported. In the SWO this is never supported, and this bit reads as 1.
[8:6]	`0x2`	FIFO Size (Power of 2). A value of 2 gives a FIFO size of 4 (2^2).
[5]	1	Indicates the relationship between **ATCLK** and **TRACECLKIN**. `0x1` indicates asynchronous. The SWO always assumes the relationship is asynchronous, and this bit reads as 1.
[4:0]	`0x00`	Hidden level of input multiplexing. When non-zero this value indicates the type and number of ATB multiplexing present on the input to the ATB: b1yyyy = Manual selection where [3:0] indicate the number of selectable inputs selectable on **EXTCTLOUT[3:0]**. b01111-b00010 = Reserved. b00001 = Fixed priority done by port number. b00000 = No invisible port multiplexing present. In the SWO there is no invisible port multiplexing, so the only valid value is 5'b00000. This value is used to assist topology detection of the ATB structure.

Device Type Identifier, 0xFCC

A value of `0x11` identifies this device as a trace sink and specifically as a TPIU.

JEP106 Identity, 0xFD0 [3:0], 0xFE8 [2:0], and 0xFE4 [7:4]

JEP106 continuation code, 4-bits, JEP106 Identity code, 7-bits. ARM designed components take the value `0x4`, continuation code, and `0x3B`, identity code.

Part Number, 0xFE4 [3:0], 0xFEO [7:4], and 0xFEO [3:0]

A value of `0x914` identifies the part as a Single Wire Output.

Revision, 0xFE8 [7:4]

The value indicates the revision of the component starting from `0x0`, corresponding to r0p0. The value increments for each part number on a revision, either minor or major.

Customer Modified, 0xFEC [3:0]

This component has no customer modified RTL and takes the value `0x0`.

RevAnd, 0xFEC [7:4]

This is changed from `0x0` if any alterations have to be made to the device during layout.

12.5 Trace port control registers

This section describes the trace port control registers:

- *Supported Synchronous Port Size Register, SPR, 0x000*
- *Current Synchronous Port Size Register, CPR, 0x004*
- *Current Output Divisor Register, CODR, 0x010*
- *Selected Pin Protocol Register, SPPR, 0xF0* on page 12-12.

——— **Note** ———

For more information on the trace port, see *SWO trace port* on page 12-22.

———————

12.5.1 Supported Synchronous Port Size Register, SPR, 0x000

Figure 12-3 shows the Port Size Register layout.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Figure 12-3 Port Size Register layout

——— **Note** ———

This register is only for implementations that implement a synchronous trace port, **TRACEDATA** or **TRACECLK**.

———————

The SWO does not support synchronous mode, so this register reads as zero.

12.5.2 Current Synchronous Port Size Register, CPR, 0x004

The SWO does not support synchronous mode, so this register reads as zero.

12.5.3 Current Output Divisor Register, CODR, 0x010

With this register is possible to scale the baud rate of the SWO output. It divides **TRACECLKIN**, enabling it when the counter reaches the programmed value and thus reducing the baud rate of the **TRACESWO** pin. Its reset value is `0x0000` and the counter is always running. The actual division value for the **TRACECLKIN** frequency will be the value of this register plus one.

Whenever this register is reprogrammed, the internal counter is automatically reset, making the baud-rate change instantaneously.

Note

Changing the prescaler value while the formatter is reported as running, that is, when stopping is supported, can cause unpredictable results.

When the Divisor field has been altered, the capture device must also be adjusted where appropriate to ensure the correct data is captured.

Figure 12-4 shows the Current Output Divisor Register bit assignments.

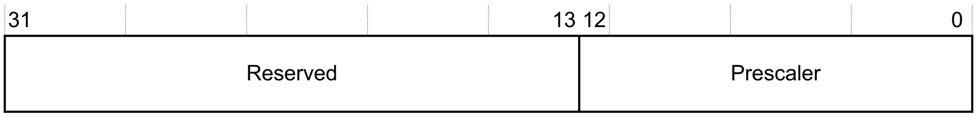


Figure 12-4 Current Output Divisor Register bit assignments

Table 12-8 shows the Current Output Divisor Register bit assignments.

Table 12-8 Current Output Divisor Register bit assignments

Bits	Name	Description
[31:13]	-	Reserved RAZ/SBZP.
[12:0]	Prescaler	13 bit counter. Reset value is `0x0000`.

12.5.4 Selected Pin Protocol Register, SPPR, 0xF0

This register selects which protocol to use for trace outputting. The register is defined in Table 12-9 on page 12-13, and the supported values allowed in the register are determined by bits [11:9] of the Device ID Register, Table 12-7 on page 12-9. When only one protocol is supported on a component, this device is read only and set as the appropriate protocol.

Figure 12-5 shows the Selected Pin Protocol Register bit assignments.

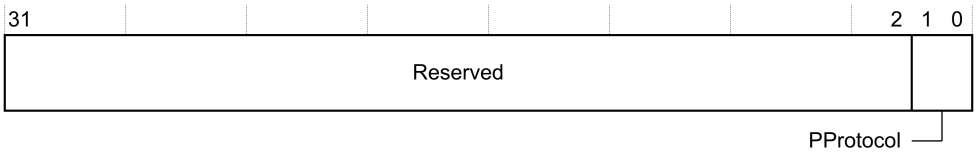


Figure 12-5 Selected Pin Protocol Register bit assignments

Table 12-9 Selected Pin Protocol Register bit assignments

Bits	Name	Description
[31:2]	-	Reserved RAZ/SBZP.
[1:0]	PProtocol	Select the pin protocol where multiple types are supported: `0x1` = Single Wire Output (Manchester). This is the reset value. `0x2` = Single Wire Output (NRZ). The selection of unsupported protocols can result in unpredictable behavior. Values `0x0` and `0x3` are reserved.

12.6 Trigger registers

Triggering is not supported in the SWO. For those trigger registers not present, read attempts return `0x00000000` and writes are ignored.

12.6.1 Supported Trigger Mode Register, STMR, 0x100

This register indicates the implemented Trigger Counter multipliers and other supported features of the trigger system. Because no trigger options are implemented in the SWO, this register must read as zero.

Figure 12-6 shows the Supported Trigger Mode Register bit assignments.

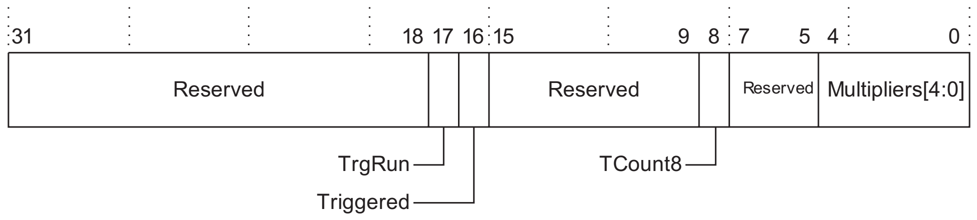


Figure 12-6 Supported Trigger Mode Register bit assignments

Table 12-10 shows the Supported Trigger Mode Register bit assignments.

Table 12-10 Supported Trigger Mode Register bit assignments

Bits	Name	Description
[31:18]		Reserved RAZ/SBZP.
[17]	TrgRun	Trigger Counter running. A trigger has occurred but the counter is not at zero.
[16]	Triggered	Triggered. A trigger has occurred and the counter has reached zero where implemented.
[15:9]		Reserved RAZ/SBZP.
[8]	TCount8	8-bit wide counter register implemented.
[7:5]		Reserved RAZ/SBZP.
[4]	Mult64k	Multiply the Trigger Counter by 65536 supported.
[3]	Mult256	Multiply the Trigger Counter by 256 supported.
[2]	Mult16	Multiply the Trigger Counter by 16 supported.
[1]	Mult4	Multiply the Trigger Counter by 4 supported.
[0]	Mult2	Multiply the Trigger Counter by 2 supported.

12.7 EXTCTL registers

External control inputs and outputs are not implemented in the SWO. Read attempts to these registers, EXTCTLIN and EXTCTLOUT, return `0x00000000` and writes are ignored.

12.8 Test pattern registers

The test pattern generator is not implemented in the SWO. For those test pattern registers not present, read attempts return `0x00000000` and writes are ignored.

12.8.1 Supported Test Patterns and Modes Register, STPR, 0x200

The pattern generator is not implemented in the SWO, so this register always reads zero.

Figure 12-7 shows the Supported Trigger Mode Register bit assignments.

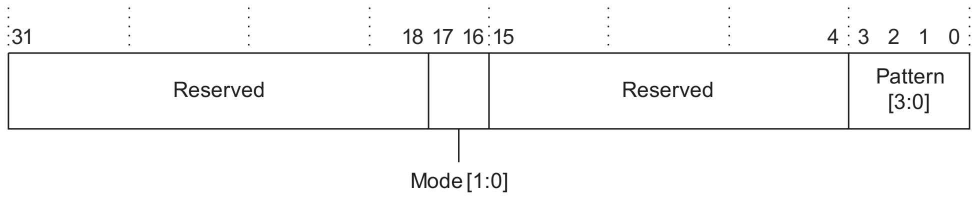


Figure 12-7 Supported Test Pattern and Mode Register bit assignments

Table 12-11 shows the Supported Trigger Mode Register bit assignments.

Table 12-11 Supported Test Pattern and Mode Register bit assignments

Bits	Name	Description
[31:18]	-	Reserved RAZ/SBZP.
[17]	PContEn	Continuous Mode
[16]	PTimeEn	Timed Mode
[15:9]	-	Reserved RAZ/SBZP.
[8]	PPat8	8-bit Programmable Test Pattern
[7:4]	-	Reserved RAZ/SBZP.
[3]	FPatF0	FF/00 Pattern
[2]	FPatA5	AA/55 Pattern
[1]	FPatW1	Walking 0's Pattern
[0]	FPatW0	Walking 1's Pattern

12.9 Formatter and flush registers

This section describes the formatter and flush registers:

- *Formatter and Flush Status Register, FFSR, 0x300*
- *Formatter and Flush Control Register, FFCR, 0x304* on page 12-18.

Note

The SWO does not implement the formatter or support flushing. For those formatter and flush registers not present, read attempts return `0x00000000` and writes are ignored.

12.9.1 Formatter and Flush Status Register, FFSR, 0x300

Figure 12-8 shows the Formatter and Flush Status Register bit assignments.

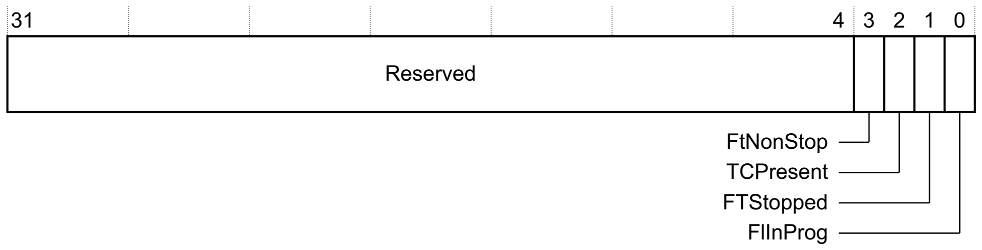


Figure 12-8 Formatter and Flush Status Register bit assignments

Table 12-12 shows the Formatter and Flush Status Register bit assignments.

Table 12-12 Formatter and Flush Status Register bit assignments

Bits	Name	Description
[31:4]	-	Reserved RAZ/SBZP.
[3]	FtNonStop	Formatter cannot be stopped. Settings can be altered without the requirement to stop the formatter, therefore this bit remains HIGH.
[2]	TCPresent	TRACECTL is not present, therefore this bit is tied off to 0.
[1]	FtStopped	Formatter stopped. Because the formatter is not present, this bit always reads 0.
[0]	FlInProg	Flush In Progress. Because flushing is not supported, this bit always reads 0.

12.9.2 Formatter and Flush Control Register, FFCR, 0x304

The formatter is not implemented in the SWO, therefore this register is read only, and always returns zero, representing the only fixed setting the SWO supports.

Figure 12-9 shows the Formatter and Flush Control Register bit assignments.

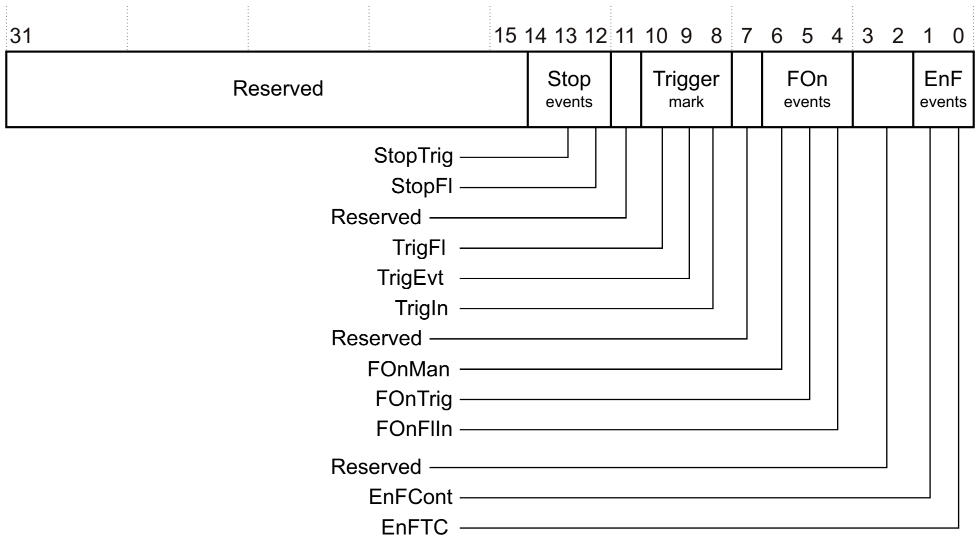


Figure 12-9 Formatter and Flush Control Register bit assignments

Table 12-13 shows the Formatter and Flush Control Register bit assignments.

Table 12-13 Formatter and Flush Control Register bit assignments

Bits	Name	Description
[31:15]	-	Reserved RAZ/SBZP.
[14]	StopMan	Manually stop the output of trace.
[13]	StopTrig	Stop the formatter when a Trigger Event is observed.
[12]	StopFl	Stop the formatter when a flush has completed, that is, return of **AFREADY**. This forces the FIFO to drain of any part completed packets.
[11]	-	Reserved RAZ/SBZP.
[10]	TrigFl	Indicate a trigger on Flush completion, that is, **AFREADY** is returned.
[9]	TrigEvt	Indicates a trigger on a Trigger Event.

Table 12-13 Formatter and Flush Control Register bit assignments (continued)

Bits	Name	Description
[8]	TrigIn	Indicates a trigger on **TRIGIN** being asserted.
[7]	-	Reserved RAZ/SBZP.
[6]	FOnMan	Manually generates a flush of the system. Setting this bit causes a flush to be generated. This is cleared when this flush has been serviced. This bit is clear on reset.
[5]	FOnTrig	Generates flush using a trigger event. Set this bit to cause a flush of data in the system when a trigger event occurs.
[4]	FOnFlIn	Generate Flush using the **FLUSHIN** interface. Set this bit to enable use of the **FLUSHIN** connection.
[3:2]	-	Reserved RAZ/SBZP.
[1]	EnFCont	Continuous Formatting, that is, data is always present. Embed in trigger packets and indicate null cycles using sync packets.
[0]	EnFTC	Enable Formatting. Do not embed triggers into the formatted stream. Trace disable cycles and triggers are indicated where possible in the selected pin protocol.

12.10 Integration test registers

This section describes the integration and topology registers that can be present depending on the component configuration:

- *Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC*
- *Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 12-21
- *Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 12-21.

——— **Note** ———

For a tool to reliably use integration registers for enhanced topology detection, it must be aware of the part number because there is no way to distinguish between inputs being LOW, and reserved locations.

When the integration test registers have been used, the device must be reset before any further usage.

Those locations marked as topology bits must be present on SWO implementations to claim CoreSight compliance.

———————————

12.10.1 Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC

Figure 12-10 shows the Integration Test ATB Data Register 0 bit assignments.

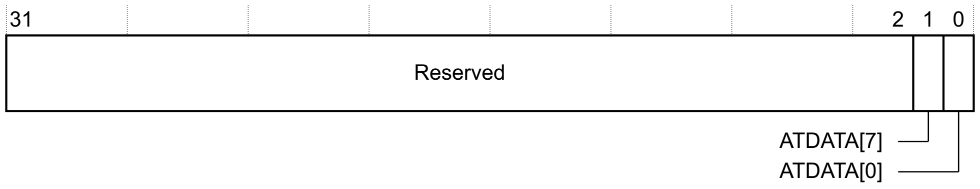


Figure 12-10 Integration Test ATB Data Register 0 bit assignments

Table 12-14 shows the Integration Test ATB Data Register 0 bit assignments.

Table 12-14 Integration Test ATB Data Register 0 bit assignments

Bits	Name	Description
[31:2]	-	Reserved RAZ/SBZP.
[1]	ATDATA[7]	Reads the value of **ATDATAS[7]**
[0]	ATDATA[0]	Reads the value of **ATDATAS[0]**

12.10.2 Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0

Figure 12-11 shows the Integration Test ATB Control Register 2 bit assignments.

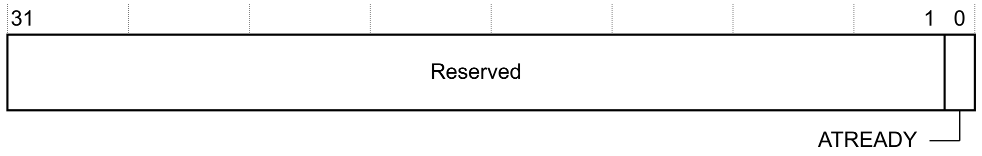


Figure 12-11 Integration Test ATB Control Register 2 bit assignments

Table 12-15 shows the Integration Test ATB Control Register 2 bit assignments.

Table 12-15 Integration Test ATB Control Register 2 bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ/SBZP.
[0]	ATREADY	Sets the value of **ATREADYS**. Topology detection register, always present

12.10.3 Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8

Figure 12-12 shows the Integration Test ATB Control Register 0 bit assignments.

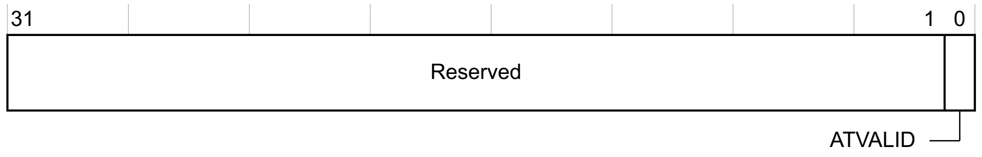


Figure 12-12 Integration Test ATB Control Register 0 bit assignments

Table 12-16 shows the Integration Test ATB Control Register 0 bit assignments.

Table 12-16 Integration Test ATB Control Register 0 bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ/SBZP.
[0]	ATVALID	Reads the value of **ATVALIDS**. Topology detection register, always present

12.11 SWO trace port

This section contains additional information about output port sizes and modes:

- *Supported port sizes*
- *Physical pin protocol*.

12.11.1 Supported port sizes

Both asynchronous port modes only operate over a single pin, **TRACESWO**, and do not require separate clock or control pins. Although data is output over a single pin, every data packet is in 8-bit multiples (bytes) which results in the supported port size register reporting 8-bit wide ports when either the Manchester or UART pin protocols are selected.

12.11.2 Physical pin protocol

For a trace capture device to correctly interpret trace data from a SWO, it must know how to decode where data exists on the various pin protocols. This section describes how logic is interpreted and any wire protocols that must be decoded to establish the underlying transmit data.

Note

In Table 12-17 on page 12-23 and Table 12-18 on page 12-24, the terms HIGH and LOW are relative terms that describe how the pin can appear electrically:

- LOW is typically 0V
- HIGH is equivalent to the supply voltage.

Two types of encoding are used:

- *Manchester encoding* on page 12-23
- *UART encoding* on page 12-24.

Manchester encoding

This protocol is supported if bit[10] of the Device ID Register, `0xFC8`, is set. It is enabled when bits[1:0] of the Selected Pin Protocol Register, `0x00C`, are set to 2'b01. When in Manchester encoding, the SWO outputs up to eight bytes of data between a start and a stop bit.

Figure 12-13 shows how a sequence of bytes is transmitted using the Manchester bit encoding.

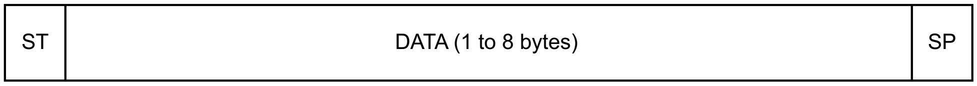


Figure 12-13 SWO Manchester encoded data sequence

Table 12-17 shows the Manchester pin protocol encoding.

Table 12-17 Manchester pin protocol encoding

Pin	Logic '0	'Logic '1	'Idle State (no data)	Valid Data
TRACESWO	"01	""10	"LOW ("00")	Start bit: HIGH-LOW transition/Logic'1', between 1 and 8 bytes of data. Stop bit: Output LOW, not a valid Manchester symbol.

Figure 12-14 shows an example of Manchester encoding, transmitting a bit pattern of 01100101.

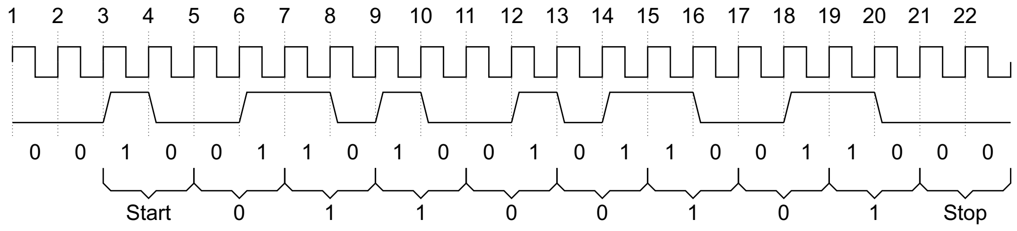


Figure 12-14 Manchester encoding example

UART encoding

This protocol is supported if bit[11] of the Device ID Register, `0xFC8`, is set. It is enabled when bits[1:0] of the Selected Pin Protocol Register, `0x00C`, are set to 2'b10. Data is sent out in packets of ten bits, with start and stop bits, as shown in Figure 12-15 and Table 12-18. Capture devices are expected to operate at the same clocking speed as the **TRACESWO** pin and synchronize by waiting for a start bit.

Figure 12-15 shows a UART encoded data sequence.

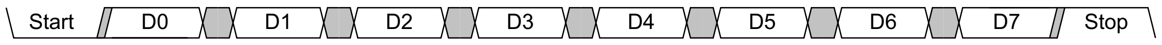


Figure 12-15 UART encoded data sequence

Table 12-18 shows the UART pin protocol encoding.

Table 12-18 UART pin protocol encoding

Pin	Logic '0	'Logic '1	'Idle State (no data)	Valid Data
TRACESWO	LOW	HIGH	HIGH	10 bit sequence: Logic '0', 8 data bits, Logic '1' is shown in Figure 12-15.

Chapter 13
Instrumentation Trace Macrocell

This chapter describes the *Instrumentation Trace Macrocell* (ITM). It contains the following sections:

13.1 About the Instrumentation Trace Macrocell

The CoreSight ITM block is a software application driven trace source. Supporting code generates *SoftWare Instrumentation Trace* (SWIT). In addition, the block provides a coarse-grained timestamp functionality. The main uses for this block are to:

- support `printf` style debugging
- trace OS and application events
- emit diagnostic system information.

Figure 13-1 shows a block diagram of the ITM.

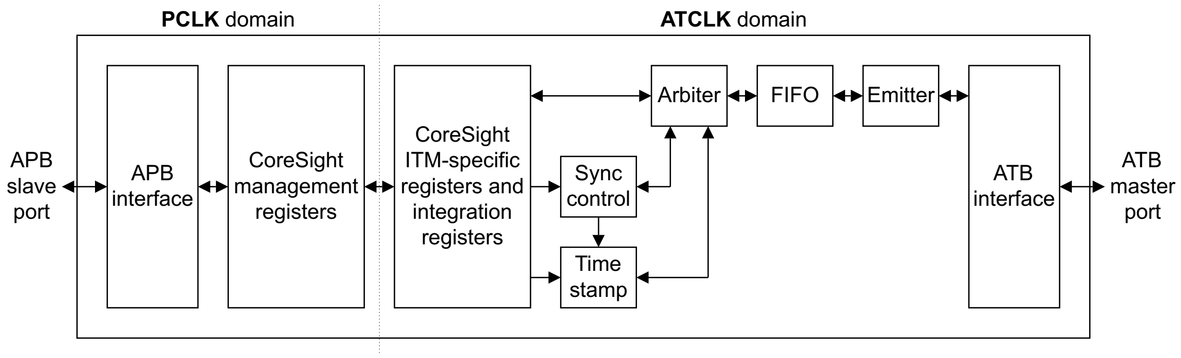


Figure 13-1 ITM block diagram

The ITM contains the following sub-blocks:

Timestamp	Generates timestamp packet.
Sync control	ITM synchronizer.
Arbiter	Arbitrates between synchronous, timestamp, and SWIT packet.
FIFO	ATB *First In First Out* (FIFO).
Emitter	ATB registered emitter.

Data is written to the stimulus registers using the APB interface. See *Stimulus registers* on page 13-14. This data is then transmitted on the ATB interface as SWIT packets as described in *SWIT packet* on page 13-7.

The operation of the ITM is described in more detail in:

- *Interface transfers* on page 13-3
- *Trace packet format* on page 13-3
- *Multiple source arbitration* on page 13-8
- *ITM FIFO* on page 13-8.

13.1.1 Interface transfers

The ITM supports two types of interface transfer:

APB The ITM has an AMBA 3 APB interface which restricts accesses to pure 32-bit transfers. To reduce the data overhead when only writing low values to stimulus registers, the ITM uses leading zero compression on data values.

Stalled transfers are not permitted to registers between `0x000` and `0xDFC`. This ensures that any writes to present or future stimulus registers do not decrease system performance.

ATB ATB trace data is 8 bits wide and, therefore, the **ATBYTES** signal is not required. The flush control, **AFVALIDM**, is not implemented in the ITM.

13.1.2 Trace packet format

Trace data from ITM is output in packets with a byte protocol. These packets are 1-5 bytes in size, comprising:

- one byte header
- 0-4 byte payload.

Note

Synchronization packets differ slightly, and comprise a unique 6-byte sequence for bit and byte synchronization.

Table 13-1 and Table 13-2 on page 13-4 show the trace packet encodings.

Table 13-1 Sync packet encoding

Description	Packet value	Payload	Category	Notes
Synchronization	0h800000000000	None	Synchronization	{47{1'b0}} followed by 1'b1. Matches ETM protocol.

Table 13-2 Trace packet encoding

Description	Header value	Payload	Category	Notes
Overflow	0b01110000	0	Protocol	Overflow

Timestamp	0bCDDD0000	0-4 bytes	Protocol	D = data (!=000) - time
				C = continuation

Reserved	0bCxxx0100	0-4 bytes	Reserved	C = continuation

SWIT	0bBBBBB0SS	1, 2, or 4 bytes	Software source (application)	SS = size (!=00) of payload
				B = SW Source Address

The ITM trace packets are described in:

- *Synchronization packet*
- *Overflow packet*
- *Timestamp packet* on page 13-5
- *SWIT packet* on page 13-7
- *Reserved encodings* on page 13-7.

Synchronization packet

Synchronization packets are unique patterns in the bit-stream that enable capture hardware to identify bit-to-byte alignment. A synchronization packet is only emitted if the interface has been used since the last synchronization packet. Synchronization packets are output by the timestamp packet when the synchronization counter reaches zero. A synchronization packet is forty-seven 1'b0 followed by one 1'b1. Figure 13-2 shows the layout of a synchronization packet.

	MSB ◄							► LSB
Byte 0	0	0	0	0	0	0	0	0
Byte 1	0	0	0	0	0	0	0	0
Byte 2	0	0	0	0	0	0	0	0
Byte 3	0	0	0	0	0	0	0	0
Byte 4	0	0	0	0	0	0	0	0
Byte 5	1	0	0	0	0	0	0	0

Figure 13-2 Synchronization packet layout

Overflow packet

Overflow packets can precede all packets except synchronization packets. Overflow is tracked by each source, either SWIT or timestamp.

SWIT overflow is against all stimulus ports. This happens when data is written to the Stimulus Registers but the FIFO is full. Timestamp overflows on reaching the value `2097151`.

Figure 13-3 shows the layout of an overflow packet.

MSB ◄──► LSB

0	1	1	1	0	0	0	0

Figure 13-3 Overflow packet layout

Note

Before disabling the ITM, tools must ensure the FIFO is not full before writing a dummy packet to a Stimulus Register to flush out any overflow information. They must then wait for the ITMBUSY bit to go LOW again, before finally clearing the ITMEn bit.

Timestamp packet

Timestamp packets encode timestamp information, generic control, and synchronization. The compression scheme uses delta timestamps. Emitting a timestamp zeroes the time counter. Timestamp resolution is related to the clock on the ATB interface. To prevent overflow, the timestamp is emitted at 95% of the counter limit.

The CoreSight ITM implementation has a 21-bit counter, emitting up to TS[20:0], and emitting a full timestamp at 2,000,000 (21'h1E847F) cycles. The architecture allows for a 28-bit counter.

Figure 13-4 shows the layout of an timestamp packet.

MSB ◄──► LSB

Byte 0	C	TC[2]	TC[1]	TC[0]	0	0	0	0
Byte 1	C	TS[6]	TS[5]	TS[4]	TS[3]	TS[2]	TS[1]	TS[0]
Byte 2	C	TS[13]	TS[12]	TS[11]	TS[10]	TS[9]	TS[8]	TS[7]
Byte 3	C	TS[20]	TS[19]	TS[18]	TS[17]	TS[16]	TS[15]	TS[14]
Byte 4	0	TS[27]	TS[26]	TS[25]	TS[24]	TS[23]	TS[22]	TS[21]

Figure 13-4 Timestamp packet layout

In Figure 13-4:

C	Continuation bit, a byte follows. Leading zeroes are implied where C=0, to compress the timestamp packet into the minimum of bytes.
TS [27:0]	Byte packing timestamp information.

TC[2:0] Timestamp control where:

Only one byte output (header only) (C == 0 of Byte 0)

Single byte timestamps are optimized encodings for when there is a timestamp of value 1-6 with no skew.

[b000] = Reserved timestamp control.

[b001-b110] = Timestamp emitted synchronous to ITM data.

[b111] = Overflow ITM.

Two+ bytes (header + payload) (C == 1 of Byte 0)

[b000-b011] = Reserved timestamp control.

[b100] = Timestamp emitted synchronous to ITM data.

[b101] = Timestamp emitted delayed to ITM.

[b110] = Packet emitted delayed.

[b111] = Packet and timestamp emitted delayed.

For example, a timestamp value of 7 is transmitted as two bytes. The value of 7 is transmitted in the second byte and delay information in the first byte.

A packet delay marker is emitted when ITM attempts to write to the FIFO but are blocked because of a priority rule, or the FIFO not accepting the data.

A timestamp delay marker is emitted when a timestamp is ready but the FIFO is unable to accept the packet. A delay is not marked if waiting on higher priority transactions, because the timestamp will continue to count.

The timestamp counter zeroes on a successful write. Timestamps are appended after the source, SWIT. The exception is if a synchronization packet is transmitted after the SWIT or the timestamp counter has overflowed, so the timestamp is preceded by an OVERFLOW or if a full timestamp reference, two million, is reached and a packet is emitted.

For example, a timestamp is configured to count every core clock, with a value of 1001 on the first ITM packet:

```
1000        idle
1001        SWIT
1002        fifo not ready
1003        fifo not ready
1004        TS=1004 with TS delay marked (3 Bytes 0xd0, 0xec, 0x07)
0           SWIT
1           SWIT
2           TS=2 (single byte 0x20)
0           SWIT
1           fifo not ready / ITM packet presented but not accepted
2           SWIT
```

```
3          TS=3 packet delay marked (2 Bytes 0xe0, 0x03)
0          idle
1          idle
2          idle
...
1999997    idle
1999998    idle
1999999    TS=1999999 (2,000,000 ticks)
0          idle
1          idle
...
2097146    fifo not ready
2097147    fifo not ready / ITM packet presented but not accepted
2097148    SWIT packet
2097149    fifo not ready
2097150    fifo not ready
2097151    fifo not ready (2^21 limit)
0          fifo not ready
1          fifo not ready
2          fifo not ready
3          OVERFLOW marked on TS, TS=3, packet delay marked, TS delay marked
           (3 Bytes 0x70, 0xF0, 0x03)
0          fifo not ready
1          fifo not ready
```

SWIT packet

Instrumentation trace originates from a software application writing to stimulus ports of ITM. Figure 13-5 shows the layout of an overflow packet.

MSB ◄──────► LSB

Byte 0	B[4]	B[3]	B[2]	B[1]	B[0]	0	S[1]	S[0]
Byte 1	Payload[7:0]							
Byte 2	Payload[15:8]							
Byte 3	Payload[23:16]							
Byte 4	Payload[31:24]							

Figure 13-5 SWIT packet layout

In Figure 13-5:

- S[1:0] = Payload size. b01=8, b10=16, b11=32, b00=invalid
- B[4:0] = 5 bits of address, **PADDRDBG[7:2]**
- Payload[31:0] = Data written from application.

Reserved encodings

Reserved encodings, at packet b0100, are not implemented in the ITM.

13.1.3 Multiple source arbitration

The ITM block outputs trace from two sources, SWIT and timestamp. When multiple sources are trying to emit data, arbitration is performed as shown in Table 13-3. As seen in *Overflow packet* on page 13-4, overflow is caused by SWIT or timestamp and precedes the source. The synchronization packet has the highest priority and is emitted by the timestamp packet when the synchronization counter reaches zero.

Table 13-3 ITM packet priority levels

Packet	Priority
Synchronization	Priority level 0, highest priority
Overflow	Priority level 1
SWIT	Priority level 2
Timestamp	Priority level 3, lowest priority

Note

SWIT has a higher priority than timestamp to ensure that SWIT output is always available and that timestamps are emitted after a run.

13.1.4 ITM FIFO

The ITM FIFO is 10 bytes. Data is output from the FIFO over an 8-bit ATB interface.

13.2 ITM ports

The ITM ports are described in:

- *Clocks and resets*
- *ITM stimulus and control interface*
- *ATB interface ports* on page 13-10
- *Authentication interface ports* on page 13-10
- *Miscellaneous ports* on page 13-11.

13.2.1 Clocks and resets

Table 13-4 shows the ITM clocks and resets.

Table 13-4 Clocks and resets

Name	Type	Description
PCLKDBG	Input	Debug APB clock, synchronous to **ATCLK**
PCLKENDBG	Input	Debug APB clock enable, tied HIGH if not required
PRESETDBGn	Input	Debug APB reset, active LOW
ATCLK	Input	Trace bus clock
ATCLKEN	Input	Trace bus clock enable, tied HIGH if not required
ATRESETn	Input	Trace bus reset, active LOW

13.2.2 ITM stimulus and control interface

Table 13-5 shows the ITM interface ports in the **PCLKDBG** domain.

Table 13-5 ITM stimulus and control interface ports

Name	Type	Description
PSELDBG	Input	Indicates that a transfer is intended for the ITM interface
PADDRDBG31	Input	Enables lock register bypassing by external debug agents, such as a DAP
PADDRDBG[11:2]	Input	Address bus to select control or stimulus registers
PENABLEDBG	Input	Enable signal to indicate second and subsequent cycles
PWRITEDBG	Input	Write transfer this cycle, and !Read

Table 13-5 ITM stimulus and control interface ports (continued)

Name	Type	Description
PWDATADBG[31:0]	Input	Data bus for write transfers
PREADYDBG	Output	Ready signal, pulled LOW to extend transfers
PRDATADBG[31:0]	Output	Data bus for read transfers., only valid on reads when **PREADYDBG** is HIGH

13.2.3 ATB interface ports

Table 13-6 shows the ITM trace interface ports in the **ATCLK** domain.

Table 13-6 ATB interface ports

Name	Type	Description
ATREADYM	Input	Indicates that the ITM ATB interface can drive a valid transfer
ATVALIDM	Output	Indicates whether the current transfer is valid
ATDATAM[7:0]	Output	The data output from the FIFO
ATIDM[6:0]	Output	ITM Trace ID
AFREADYM	Output	Flush complete, all historical data output

13.2.4 Authentication interface ports

The signals in Table 13-7 are required for disallowing trace generation when not permitted.

Table 13-7 Authentication interface ports

Name	Type	Description
DBGEN	Input	Invasive Debug Enable, implies non-invasive
NIDEN	Input	Non-Invasive Debug Enable
SPIDEN	Input	Secure Invasive Debug Enable, implies secure non-invasive
SPNIDEN	Input	Secure Non-Invasive Debug Enable

13.2.5 Miscellaneous ports

Table 13-8 shows the miscellaneous ITM ports.

Table 13-8 Miscellaneous ports

Name	Type	Description
TRIGOUT	Output	Indicates a trigger event, that is, a write to a stimulus register that has a trigger enable against it
TRIGOUTACK	Input	Asynchronous **TRIGOUT** acknowledgement signal
SE	Input	Scan enable.

13.3 ITM programmer's model

Table 13-9 shows the ITM programmable registers.

Table 13-9 ITM programmable registers

Offset	Type	Width	Reset value	Name	Description
`0x000-0x07C`	R/W	32	`0x00000000`	Stimulus Port Register 0 to 31	See *Stimulus registers* on page 13-14
`0xE00`	R/W	32	`0x00000000`	Trace Enable Register	See *Trace registers* on page 13-15
`0xE20`	R/W	32	`0x00000000`	Trace Trigger Register	
`0xE80`	R/W	32	`0x00000004`	Control Register	See *Control registers* on page 13-17
`0xE90`	R/W	12	`0x00000400`	Sync Control Register	
`0xEE4`	RO	1	`0x00000000`	Integration Test Trigger Out Acknowledge Register	See *Integration test registers* on page 13-20
`0xEE8`	WO	1	-	Integration Test Trigger Out Register	
`0xEEC`	WO	2	-	Integration Test ATB Data Register 0	
`0xEF0`	R0	1	`0x00000000`	Integration Test ATB Control Register 2	
`0xEF4`	WO	7	-	Integration Test ATB Control Register 1	
`0xEF8`	WO	2	-	Integration Test ATB Control Register 0	
`0xF00-0xFFC`	-	-	-	CoreSight Management Registers	See *CoreSight defined registers* on page 13-13

13.4 CoreSight defined registers

These registers are described in *CoreSight Management Registers summary* on page 2-12. This section describes the values that are specific to the ITM.

13.4.1 Integration Mode Control Register, 0xF00

This register enables the component to switch from a functional mode, the default behavior, to integration mode. The ITM must be disabled and be in the idle state before setting the Integration Mode.

13.4.2 Claim Tag Set Register, 0xFA0, and Claim Tag Clear Register, 0xFA4

The ITM implements an 8-bit claim tag.

13.4.3 Lock Access Register, 0xFB0, and Lock Status Register, 0xFB4

The ITM implements a 32-bit Lock Access Register:

- Bypassed with **PADDRDBG31** HIGH.
- Only protect writes to registers inside `0xE00-0xFFC`. Bypass lock mechanism to writes in `0x000-0xDFC`.

13.4.4 Authentication Status Register, 0xFB8

Authentication Status Register == `8'b1S001N00` where S is secure non-invasive debug state and N is non-secure, non-invasive debug.

13.4.5 Device Configuration Register, 0xFC8

This register specifies the number of stimulus registers implemented. DEVID = `0x20`.

13.4.6 Device Type Identifier Register, 0xFCC

This register specifies that the device is a Trace Source and the stimulus is derived from bus activity. DEVTYPE = `0x43`.

13.4.7 Peripheral ID Registers, 0xFDC-0xFE0

The part number [11:0] of the CoreSight ITM is `0x913`. This device uses only 4KB of memory.

13.4.8 Component ID Registers (RO 0xFFC to 0xFF0)

The component class is `0x9`.

13.5 Stimulus registers

Each of the 32 stimulus ports is represented by a virtual address. A write to one of these locations causes data to be written into the FIFO if the corresponding bit in the Trace Enable Register is set and ITM is enabled. Reading from any of the stimulus ports returns the FIFO status (`notFull(1) / Full(0)`) only if the ITM is enabled. This enables more efficient core register allocation because the stimulus address has already been generated.

The ITM transmits SWIT packets using leading zero compression. Packets can be 8, 16, or 32 bits. See *SWIT packet* on page 13-7.

The bank of 32 registers is split into a low-16 (0 to 15) and a high-16 (16 to 31). Writes to the high-16 are discarded by the ITM whenever secure non-invasive trace is disabled, regardless of how the Trace Enable Register bits [31:16] are set. Both the high-16 and low-16 are be disabled when non-invasive trace is disabled. When an input is disabled it must not alter the interface response and must always return an OK without stalling.

Note

Any instrumented secure code must use the upper 16 (16-31) register locations that are masked against secure non-invasive enable. You must allow both secure and non-secure transactions to the ITM.

The polled FIFO interface does not provide an atomic *Read Modify Write* (RMW), so an exclusive monitor must be used if a polled `printf` is used concurrently with ITM usage by interrupts or other threads. The following polled code guarantees stimulus is not lost by polled access to the ITM:

```
    ; R0 = FIFO-full/exclusive status
    ; R1 = base of ITM stimulus ports
    ; R2 = value to write
retry
    LDREX R0,[R1,#??]       ; read FIFO status and request excl lock
    CZBEQ R0,retry          ; FIFO not ready, try again
    STREX R0,R2,[R1,#??]    ; store if FIFO !Full and excl lock
    CZBNE R0,retry          ; excl lock failed, try again
```

If multiple writes are performed to the ITM when the FIFO is full, the transaction is discarded although no error or stall must be generated on the interface.

These registers must not be blocked by the Lock Access Register.

Writes to registers to `0x080-0xDFC` result in no SWIT packet being transmitted. Reads from these registers return a `0x0`. This address space is reserved for future stimulus registers.

Note

The DEVID specifies how many stimulus registers are implemented.

13.6 Trace registers

This section describes the trace registers:

- *Trace Enable Register, TER, 0xE00*
- *Trace Trigger Register, TTR, 0xE20*.

13.6.1 Trace Enable Register, TER, 0xE00

Each bit location corresponds to a virtual stimulus register; when a bit is set, a write to the appropriate stimulus location results in a packet being generated, except when the FIFO is full. Reset to disable all locations is `0x00000000`.

The setting of the bits [31:16] is ignored if secure trace is disabled. Stimulus ports 16-31 are disabled when the ability to perform secure non-invasive debug is removed.

The setting of the bits [31:0] is ignored if non-secure trace is disabled. Stimulus ports 0-31 are disabled when the ability to perform non-secure non-invasive debug is removed.

Table 13-10 shows the Trace Enable Register bit assignments.

Table 13-10 Trace Enable Register bit assignments

Bits	Name	Description
[31:0]	Trace Enable Register	Bit mask to enable tracing on ITM stimulus ports.

13.6.2 Trace Trigger Register, TTR, 0xE20

Each bit in the register represents one of the virtual stimulus registers. When the bit is set, a pulse is sent on **TRIGOUT** when writing to the corresponding Stimulus Register. The pulse is held until **TRIGOUTACK** is returned, which can result in masking of multiple triggers with one acknowledgement. Triggers are not generated if trace is disabled. When secure non-invasive debug, or secure trace, is disabled, no triggers are generated for Stimulus Registers 16 to 31. If non-secure non-invasive debug, or trace, is disabled, no triggers are generated. The register is zero on reset, and the default is no triggers generated.

Table 13-11 shows the Trace Trigger Register bit assignments.

Table 13-11 Trace Trigger Register bit assignments

Bits	Name	Description
[31:0]	Trigger Mask	Bit mask to enable trigger generation, **TRIGOUT**, on selected writes to the Stimulus Registers

Figure 13-6 shows two writes to the stimulus port. It shows the timing for generating **TRIGOUT** and deasserting **TRIGOUT** with **TRIGOUTACK**. **PENABLE** and **PREADY** show when a write happens to the Stimulus Register.

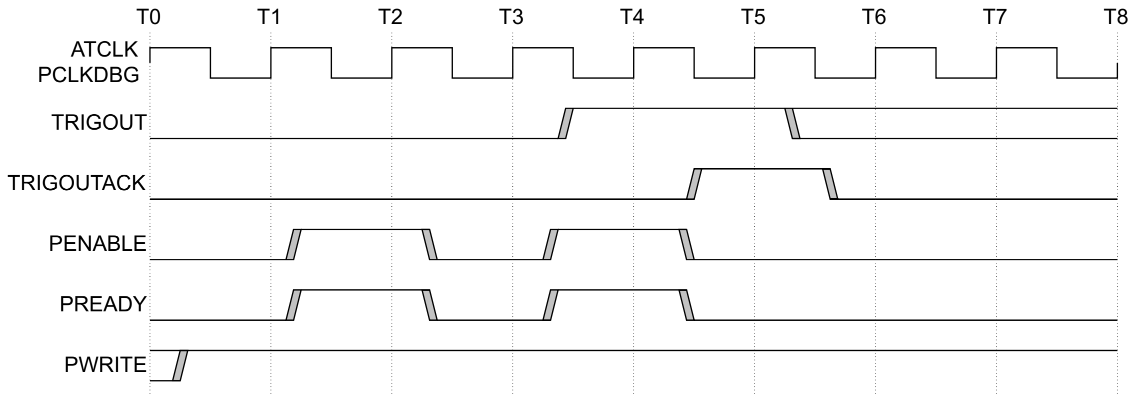


Figure 13-6 TRIGOUT and TRIGOUTACK operation

The write at T0-T2 causes the **TRIGOUT**. This is acknowledged at T5. A write at T2-T4 generates another **TRIGOUT**. If the **TRIGOUTACK** stays HIGH beyond T6, this **TRIGOUT** is also acknowledged at T6. In the example **TRIGOUTACK** goes LOW before T6, so another **TRIGOUTACK** is required.

13.7 Control registers

This section describes the control registers:

- *Control Register, CR, 0xE80*
- *Synchronization Control Register, SCR, 0xE90* on page 13-18.

13.7.1 Control Register, CR, 0xE80

This is the control word used to configure and control transfers through the APB interface.

Figure 13-7 shows the Control Register bit assignments.

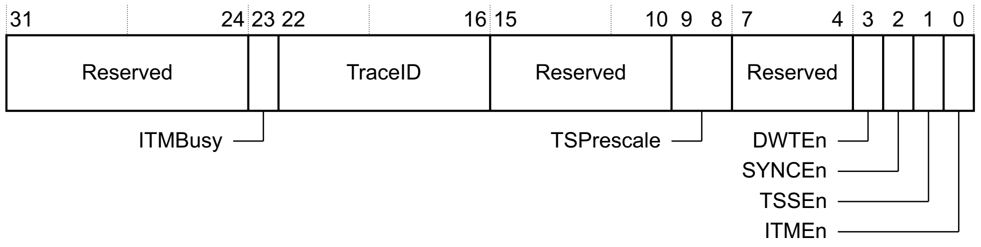


Figure 13-7 Control Register bit assignments

Table 13-12 shows the Control Register bit assignments.

Table 13-12 Control Register bit assignments

Bits	Name	Description
[31:24]	-	Reserved RAZ/SBZP
[23]	ITMBusy	ITM is transmitting trace and FIFO is not empty
[22:16]	TraceID	**ATIDM[6:0]** value
[15:10]	-	Reserved RAZ/SBZP
[9:8]	TSPrescale[a]	Timestamp Prescaler, 0b00=/1, 0b01=/4, 0b10=/16, 0b11=/64
[7:4]	-	Reserved RAZ/SBZP
[3]	DWTEn[b]	Enable DWT input port

Table 13-12 Control Register bit assignments (continued)

Bits	Name	Description
[2]	SYNCEn[c]	Enable sync packets
[1]	TSSEn	Enable timestamps, delta
[0]	ITMEn[d]	Enable ITM stimulus, also acts as a global enable

a. The TSPrescale bit is reset to `0x0`. This means that the timestamp counts on every **ATCLK** tick. However, you can program the TSPrescale bit so that the timestamp counts once every 4, 16, or 64 **ATCLK** ticks.
b. *Data Watchpoint and Trace* (DWT) is not used in the CoreSight ITM, so the DWTEn bit is always `0x0`.
c. The SYNCEn bit is always `0x1` and is enabled.
d. It is recommended that the ITMEn bit is cleared and waits for the ITMBusy bit to be cleared, before changing any fields in the Control Register, otherwise the behavior can be unpredictable. See *Overflow packet* on page 13-4 for information about disabling the ITM overflow packets.

13.7.2 Synchronization Control Register, SCR, 0xE90

Synchronization packets must be output on a regular basis to provide decompression tools markers in the trace stream. The first synchronization packet is output when the ITM is enabled by setting bit [0] in the Control Register. Subsequent packets are output when the counter reaches zero, that is, decrement on data entered into the FIFO. The Synchronization Control Register represents the reloaded count value when the counter reaches zero, not the current count. This register must only be changed when the ITM is disabled.

Note

A value of `0x000` written to the Synchronization Control Register causes unpredictable results. This value must only be used when a synchronization counter is not implemented. This register always reads as zero if no synchronization counter is implemented. A counter can be detected by attempting to write a non-zero value to the register and reading it back. The default value is `0x400`.

Figure 13-8 shows the Synchronization Control Register bit assignments.

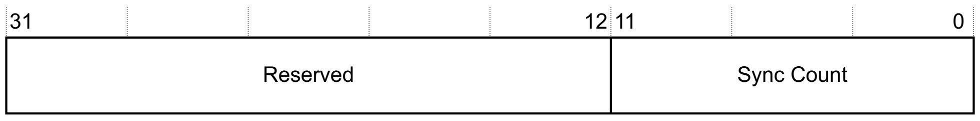


Figure 13-8 Synchronization Control Register bit assignments

Table 13-13 shows the Synchronization Control Register bit assignments.

Table 13-13 Synchronization Control Register bit assignments

Bits	Name	Description
[31:12]	-	Reserved RAZ/SBZP
[11:0]	Sync Count	Counter value for time between synchronization markers

13.8 Integration test registers

This section describes the integration test registers:

- *Integration Test Trigger Out Acknowledge Register, ITTRIGOUTACK, 0xEE4*
- *Integration Test Trigger Out Register, ITTRIGOUT, 0xEE8* on page 13-21
- *Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC* on page 13-21
- *Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0* on page 13-22
- *Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4* on page 13-22
- *Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8* on page 13-23

——— **Note** ———

Reading the integration test registers when not in integration mode can cause unpredictable behavior.

———————————

13.8.1 Integration Test Trigger Out Acknowledge Register, ITTRIGOUTACK, 0xEE4

Figure 13-9 shows the Integration Test Trigger Out Acknowledge Register bit assignments.

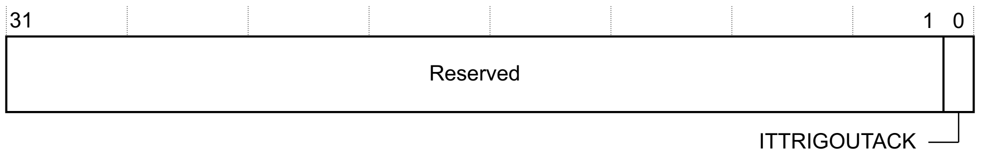


Figure 13-9 Integration Test Trigger Out Acknowledge Register bit assignments

Table 13-14 shows the Integration Test Trigger Out Acknowledge Register bit assignments.

Table 13-14 Integration Test Trigger Out Acknowledge Register bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ/SBZP
[0]	ITTRIGOUTACK	Read the value of **TRIGOUTACK**

13.8.2 Integration Test Trigger Out Register, ITTRIGOUT, 0xEE8

Figure 13-10 shows the Integration Test Trigger Out Register bit assignments.

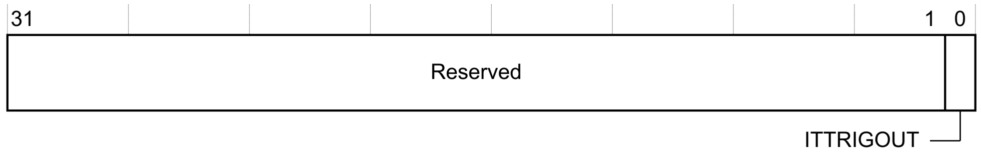


Figure 13-10 Integration Test Trigger Out Register bit assignments

Table 13-15 shows the Integration Test Trigger Out Register bit assignments.

Table 13-15 Integration Test Trigger Out Register bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ/SBZP
[0]	ITTRIGOUT	Set the value of **TRIGOUT**

13.8.3 Integration Test ATB Data Register 0, ITATBDATA0, 0xEEC

Figure 13-11 shows the Integration Test ATB Data Register 0 bit assignments.

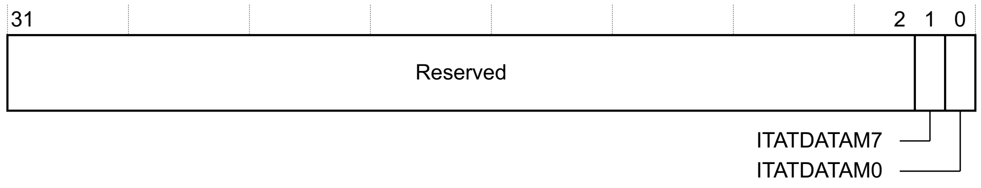


Figure 13-11 Integration Test ATB Data Register 0 bit assignments

Table 13-16 shows the Integration Test ATB Data Register 0 bit assignments.

Table 13-16 Integration Test ATB Data Register 0 bit assignments

Bits	Name	Description
[31:2]	-	Reserved RAZ/SBZP
[1]	ITATDATAM7	Set the value of **ATDATAM[7]**
[0]	ITATDATAM0	Set the value of **ATDATAM[0]**

13.8.4 Integration Test ATB Control Register 2, ITATBCTR2, 0xEF0

Figure 13-12 shows the Integration Test ATB Control Register 2 bit assignments.

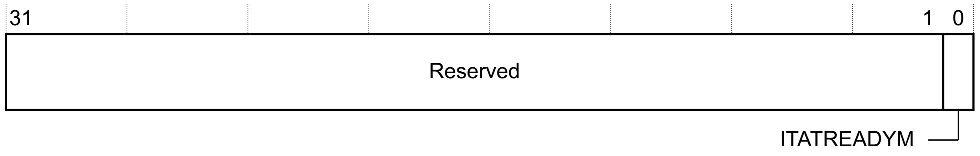


Figure 13-12 Integration Test ATB Control Register 2 bit assignments

Table 13-17 shows the Integration Test ATB Control Register 2 bit assignments.

Table 13-17 Integration Test ATB Control Register 2 bit assignments

Bits	Name	Description
[31:1]	-	Reserved RAZ/SBZP
[0]	ITATREADYM	Read the value of **ATREADYM**

13.8.5 Integration Test ATB Control Register 1, ITATBCTR1, 0xEF4

Figure 13-13 shows the Integration Test ATB Control Register 1 bit assignments.

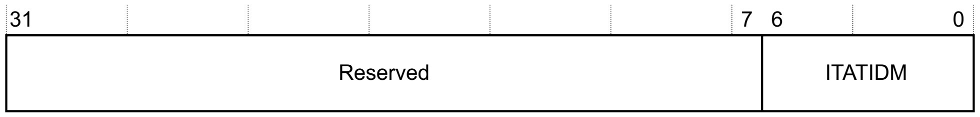


Figure 13-13 Integration Test ATB Control Register 1 bit assignments

Table 13-18 shows the Integration Test ATB Control Register 1 bit assignments.

Table 13-18 Integration Test ATB Control Register 1 bit assignments

Bits	Name	Description
[31:7]	-	Reserved RAZ/SBZP
[6:0]	ITATIDM	Set the value of **ATIDM[6:0]**

13.8.6 Integration Test ATB Control Register 0, ITATBCTR0, 0xEF8

Figure 13-14 shows the Integration Test ATB Control Register 0 bit assignments.

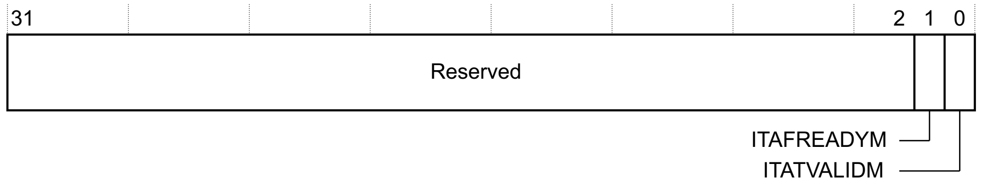


Figure 13-14 Integration Test ATB Control Register 0 bit assignments

Table 13-19 shows the Integration Test ATB Control Register 0 bit assignments.

Table 13-19 Integration Test ATB Control Register 0 bit assignments

Bits	Name	Description
[31:2]	-	Reserved RAZ/SBZP
[1]	ITAFREADYM	Set the value of **ATREADY**
[0]	ITATVALIDM	Set the value of **ATVALIDM**

Appendix A
CoreSight Port List

This appendix describes the CoreSight port and interface signals. It contains the following sections:

A.1 Clock domains

In addition to the names of clock domains there are the following entries in the port lists:

- N/A, not applicable. This is used for the clock signals.
- None. This signal is an asynchronous input. This input can be from any clock domain because the block has internal synchronizers for this signal.
- xx/None. This signal can be from clock domain 'xx' or it can be asynchronous. The signal has bypassable synchronizers it depends on which configuration the designer chooses.

A.2 CoreSight DAP signals

Table A-1 shows the CoreSight *Debug Access Port* (DAP) signals.

Table A-1 CoreSight DAP signals

Name	Type	Description	Clock domain
CDBGPWRUPACK	Input	Debug Power Domain power-up acknowledge SWJ-DP	None
CDBGPWRUPREQ	Output	Debug Power Domain power-up request SWJ-DP	None
CDBGRSTACK	Input	Debug reset acknowledge from reset controller SWJ-DP	None
CDBGRSTREQ	Output	Debug reset request to reset controller SWJ-DP	None
CSRTCK[7:0]	Input	Return **DBGCLK** from JTAG slaves JTAG-AP	None
CSTCK[7:0]	Output	DBGCLK to JTAG slaves JTAG-AP	PCLKDBG
CSTDI[7:0]	Output	TDI to JTAG slaves JTAG-AP	PCLKDBG
CSTDO[7:0]	Input	**TDO** from JTAG slaves JTAG-AP	PCLKDBG
CSTMS[7:0]	Output	TMS to JTAG slaves JTAG-AP	PCLKDBG
CSYSPWRUPACK	Input	System Power Domain power-up acknowledge SWJ-DP	None
CSYSPWRUPREQ	Output	System Power Domain power-up request SWJ-DP	None
DAPABORT	Output	DAP Abort, to support Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPADDR[31:0]	Output	DAP Address, to support Cortex-M3 processor DAPBUS exported interface	DAPCLK

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
DAPENABLE	Output	DAP Enable Transaction, to support Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPRDATACM3[31:0]	Input	DAP Read Data from Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPREADYCM3	Input	DAP Data Bus Ready from Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPSELCM3	Output	DAP transaction select to Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPSLVERRCM3	Input	AP slave error response from Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPWDATA[31:0]	Output	DAP Write Data, to support Cortex-M3 processor DAPBUS exported interface	DAPCLK
DAPWRITE	Output	DAP Bus Write, to support Cortex-M3 processor DAPBUS exported interface	DAPCLK
DBGEN	Input	Invasive Debug enable Enables AHB transfers AHB-AP	None
DEVICEEN	Input	Enable access to Debug APB from the DAP APB-AP	None
HADDRM[31:0]	Output	AHB master address AHB-AP	HCLK
HBSTRBM[3:0]	Output	AHB master byte lane strobes AHB-AP	HCLK
HBURSTM[2:0]	Output	AHB master burst type, always SINGLE AHB-AP	HCLK
HCLK	Input	AHB clock AHB-AP	N/A
HCLKEN	Input	AHB clock enable term AHB-AP	HCLK

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
HLOCKM	Output	AHB master requires locked access to the bus, always zero AHB-AP	HCLK
HPROTM[6:0]	Output	AHB master privilege information AHB-AP	HCLK
HRDATAM[31:0]	Input	AHB master read data AHB-AP	HCLK
HREADYM	Input	AHB ready response to master port AHB-AP	HCLK
HRESETn	Input	AHB reset AHB-AP	HCLK
HRESPM[1:0]	Input	AHB transfer response to master port AHB-AP	HCLK
HSIZEM[2:0]	Output	AHB master transfer size AHB-AP	HCLK
HTRANSM[1:0]	Output	AHB master transfer type, IDLE or NONSEQ AHB-AP	HCLK
HWDATAM[31:0]	Output	AHB master write data AHB-AP	HCLK
HWRITEM	Output	AHB master transfer direction AHB-AP	HCLK
JTAGNSW	Output	HIGH if JTAG selected, LOW if SWD selected	SWCLKTCK
JTAGTOP	Output	JTAG state machine is in one of the top four modes: • Test-Logic-Reset • Run-Test/Idle • Select-DR-Scan • Select-IR-Scan.	SWCLKTCK
nCDBGPWRDN	Input	Debug infrastructure power-down control	None
nCSOCPWRDN	Input	External system, SOC domain, power-down control	None
nCSTRST[7:0]	Output	**nTRST** to JTAG slaves JTAG-AP	PCLKDBG

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
nPOTRST	Input	Power-on reset SWJ-DP	SWCLKTCK
nSRSTOUT[7:0]	Output	Subsystem reset to JTAG slaves JTAG-AP	PCLKDBG
nTDOEN	Output	TAP Data Out Enable SWJ-DP	SWCLKTCK
nTRST	Input	TAP Reset, Asynchronous SWJ-DP	SWCLKTCK
PADDRDBG[31:2]	Output	Debug APB address bus APB-Mux	PCLKDBG
PADDRSYS[30:2]	Input	System APB address bus APB-Mux	PCLKSYS
PCLKDBG	Input	Debug APB clock	N/A
PCLKENDBG	Input	Debug APB clock enable	PCLKDBG
PCLKENSYS	Input	System APB clock enable APB-Mux	PCLKSYS
PCLKSYS	Input	System APB clock, typically **HCLK** APB-Mux	PCLKSYS
PENABLEDBG	Output	Debug APB enable signal, indicates second and subsequent cycles APB-Mux	PCLKDBG
PENABLESYS	Input	System APB enable signal, indicates second and subsequent cycles APB-Mux	PCLKSYS
PORTCONNECTED[7:0]	Input	JTAG ports are connected, static signals JTAG-AP	PCLKDBG
PORTENABLED[7:0]	Input	JTAG ports are active JTAG-AP	PCLKDBG
PRDATADBG[31:0]	Input	Debug APB read data bus	PCLKDBG

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
PRDATASYS[31:0]	Output	System APB read data bus APB-Mux	PCLKSYS
PREADYDBG	Input	Debug APB ready signal	PCLKDBG
PREADYSYS	Output	System APB ready signal APB-Mux	PCLKSYS
PRESETDBGn	Input	Debug APB reset	PCLKDBG
PRESETSYSn	Input	System APB reset APB-Mux	PCLKSYS
PSELDBG	Output	Debug APB select. LOW when accessing the DAP ROM APB-Mux	PCLKDBG
PSELSYS	Input	System APB select APB-Mux	PCLKSYS
PSLVERRDBG	Input	Debug APB transfer error signal	PCLKDBG
PSLVERRSYS	Output	System APB transfer error signal APB-Mux	PCLKSYS
PWDATADBG[31:0]	Output	Debug APB write data bus APB-Mux	PCLKDBG
PWDATASYS[31:0]	Input	System APB Write data bus APB-Mux	PCLKSYS
PWRITEDBG	Output	Debug APB write transfer APB-Mux	PCLKDBG
PWRITESYS	Input	System APB write transfer APB-Mux	PCLKSYS
RSTBYPASS	Input	**nPOTRST** reset bypass for DFT SWJ-DP	N/A
SE	Input	Scan Enable	None
SPIDEN	Input	Enables secure/privileged debug, TrustZone enable AHB-AP	None

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
SRSTCONNECTED[7:0]	Input	JTAG ports support Subsystem Reset, static value JTAG-AP	PCLKDBG
SWCLKTCK	Input	Serial Wire Clock and TAP Clock SWJ-DP	N/A
SWDITMS	Input	Serial Wire Data Input and TAP Test Mode Select SWJ-DP	SWCLKTCK
SWDO	Output	Serial Wire Data Output SWJ-DP	SWCLKTCK
SWDOEN	Output	Serial Wire Data Output Enable SWJ-DP	SWCLKTCK
TDI	Input	TAP Data In SWJ-DP	SWCLKTCK
TDO	Output	TAP Data Out SWJ-DP	SWCLKTCK

A.3 CoreSight ECT signals

This section describes CoreSight *Embedded Cross Trigger* (ECT), signals in the following sections:

- *CoreSight CTI signals*
- *CoreSight CTM signals* on page A-10.

A.3.1 CoreSight CTI signals

Table A-2 shows the CoreSight *Cross Trigger Interface* (CTI) signals.

Table A-2 CoreSight CTI signals

Name	Type	Description	Clock domain
CIHSBYPASS[`ECTCHANNELWIDTH-1:0]	Input	Channel interface H/S bypass	CTICLK
CISBYPASS	Input	Channel interface sync bypass	CTICLK
CTIAPBSBYPASS	Input	Between APB and CTI clock	CTICLK
CTICHIN[`ECTCHANNELWIDTH-1:0]	Input	Channel In	CTICLK/None
CTICHOUTACK[`ECTCHANNELWIDTH-1:0]	Input	Channel Out acknowledge	CTICLK/None
CTICLK	Input	CTI Clock	N/A
CTICLKEN	Input	CTI Clock Enable	CTICLK
CTITRIGIN[`ECTTRIGGERWIDTH-1:0]	Input	Trigger In	CTICLK/None
CTITRIGOUTACK[`ECTTRIGGERWIDTH-1:0]	Input	Trigger Out acknowledge	CTICLK/None
DBGEN	Input	Invasive Debug enable	None
nCTIRESET	Input	Reset	CTICLK
NIDEN	Input	Noninvasive debug enable	None
PADDRDBG[11:2]	Input	Debug APB address bus	PCLKDBG
PADDRDBG31	Input	Debug APB programming origin, HIGH for off-chip	PCLKDBG
PCLKDBG	Input	Debug APB clock	N/A
PCLKENDBG	Input	Debug APB clock enable	PCLKDBG

Table A-2 CoreSight CTI signals (continued)

Name	Type	Description	Clock domain
PENABLEDBG	Input	Debug APB enable signal, indicates second and subsequent cycles	PCLKDBG
PRESETDBGn	Input	Debug APB reset	PCLKDBG
PSELDBG	Input	Debug APB component select	PCLKDBG
PWDATADBG[31:0]	Input	Debug APB write data bus	PCLKDBG
PWRITEDBG	Input	Debug APB write transfer	PCLKDBG
SE	Input	Scan Enable	None
TIHSBYPASS[ECTTRIGGERWIDTH-1:0]	Input	Trigger interface H/S bypass, static value	CTICLK
TINIDENSEL[ECTCHANNELWIDTH-1:0]	Input	Mask when **NIDEN** is LOW, static value	CTICLK
TISBYPASSACK[ECTTRIGGERWIDTH-1:0]	Input	Trigger Out ACK Sync bypass, static value	CTICLK
TISBYPASSIN[ECTTRIGGERWIDTH-1:0]	Input	Trigger In Sync bypass, static value	CTICLK
TODBGENSEL[ECTCHANNELWIDTH-1:0]	Input	Mask when **DBGEN** is LOW, static value	CTICLK
ASICCTL[7:0]	Output	External mux control	CTICLK
CTICHINACK[ECTCHANNELWIDTH-1:0]	Output	Channel In acknowledge	CTICLK
CTICHOUT[ECTCHANNELWIDTH-1:0]	Output	Channel Out	CTICLK
CTITRIGINACK[ECTTRIGGERWIDTH-1:0]	Output	Trigger In acknowledge	CTICLK
CTITRIGOUT[ECTTRIGGERWIDTH-1:0]	Output	Trigger Out	CTICLK
PRDATADBG[31:0]	Output	Debug APB read data bus	PCLKDBG
PREADYDBG	Output	Debug APB Ready signal	PCLKDBG

A.3.2 CoreSight CTM signals

Table A-3 on page A-11 shows the CoreSight *Cross Trigger Matrix* (CTM) signals.

Table A-3 CoreSight CTM signals

Name	Type	Description	Clock domain
CIHSBYPASS0[ECTCHANNELWIDTH-1:0]	Input	Handshaking bypass port 0	CTMCLK
CIHSBYPASS1[ECTCHANNELWIDTH-1:0]	Input	Handshaking bypass port 1	CTMCLK
CIHSBYPASS2[ECTCHANNELWIDTH-1:0]	Input	Handshaking bypass port 2	CTMCLK
CIHSBYPASS3[ECTCHANNELWIDTH-1:0]	Input	Handshaking bypass port 3	CTMCLK
CISBYPASS0	Input	Sync bypass for port 0	CTMCLK
CISBYPASS1	Input	Sync bypass for port 1	CTMCLK
CISBYPASS2	Input	Sync bypass for port 2	CTMCLK
CISBYPASS3	Input	Sync bypass for port 3	CTMCLK
CTMCHIN0[ECTCHANNELWIDTH-1:0]	Input	Channel In port 0	CTMCLK
CTMCHIN1[ECTCHANNELWIDTH-1:0]	Input	Channel In port 1	CTMCLK/None
CTMCHIN2[ECTCHANNELWIDTH-1:0]	Input	Channel In port 2	CTMCLK/None
CTMCHIN3[ECTCHANNELWIDTH-1:0]	Input	Channel In port 3	CTMCLK/None
CTMCHOUTACK0[`ECTCHANNELWIDTH-1:0]	Input	Channel Out ACK port 0	CTMCLK/None
CTMCHOUTACK1[`ECTCHANNELWIDTH-1:0]	Input	Channel Out ACK port 1	CTMCLK/None
CTMCHOUTACK2[`ECTCHANNELWIDTH-1:0]	Input	Channel Out ACK port 2	CTMCLK/None
CTMCHOUTACK3[`ECTCHANNELWIDTH-1:0]	Input	Channel Out ACK port 3	CTMCLK/None
CTMCLK	Input	Clock	N/A
nCTMRESET	Input	Reset	CTMCLK
SE	Input	Scan Enable	None
CTMCHINACK0[ECTCHANNELWIDTH-1:0]	Output	Channel IN ACK port 0	CTMCLK
CTMCHINACK1[ECTCHANNELWIDTH-1:0]	Output	Channel IN ACK port 1	CTMCLK
CTMCHINACK2[ECTCHANNELWIDTH-1:0]	Output	Channel IN ACK port 2	CTMCLK
CTMCHINACK3[ECTCHANNELWIDTH-1:0]	Output	Channel IN ACK port 3	CTMCLK
CTMCHOUT0[ECTCHANNELWIDTH-1:0]	Output	Channel OUT port 0	CTMCLK
CTMCHOUT1[ECTCHANNELWIDTH-1:0]	Output	Channel OUT port 1	CTMCLK
CTMCHOUT2[ECTCHANNELWIDTH-1:0]	Output	Channel OUT port 2	CTMCLK
CTMCHOUT3[ECTCHANNELWIDTH-1:0]	Output	Channel OUT port 3	CTMCLK

A.4 CoreSight replicator signals

Table A-4 shows the CoreSight replicator signals.

Table A-4 CoreSight replicator signals

Name	Type	Description	Clock domain
AFREADYS	Input	ATB Data flush complete for the master port	ATCLK
AFVALIDM0	Input	ATB Data flush request for the master port 0	ATCLK
AFVALIDM1	Input	ATB Data flush request for the master port 1	ATCLK
ATBYTESS[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port	ATCLK
ATCLK	Input	ATB clock	N/A
ATCLKEN	Input	ATB clock enable	ATCLK
ATDATAS[31:0]	Input	ATB Trace data on the slave port	ATCLK
ATIDS[6:0]	Input	ATB ID for current trace data on slave port	ATCLK
ATREADYM0	Input	ATB transfer ready on master port 0	ATCLK
ATREADYM1	Input	ATB transfer ready on master port 1	ATCLK
ATRESETn	Input	ATB Reset for the ATCLK domain	ATCLK
ATVALIDS	Input	ATB Valid signals present on slave port	ATCLK
SE	Input	Scan Enable	None
AFREADYM0	Output	ATB Data flush complete for the master port 0	ATCLK
AFREADYM1	Output	ATB Data flush complete for the master port 1	ATCLK
AFVALIDS	Output	ATB Data flush request for the master port	ATCLK
ATBYTESM0[1:0]	Output	ATB Number of valid bytes, LSB aligned, on the master port	ATCLK
ATBYTESM1[1:0]	Output	ATB Number of valid bytes, LSB aligned, on the master port	ATCLK
ATDATAM0[31:0]	Output	ATB Trace data on the master port 0	ATCLK
ATDATAM1[31:0]	Output	ATB Trace data on the master port 1	ATCLK
ATIDM0[6:0]	Output	ATB ID for current trace data on master port 0	ATCLK
ATIDM1[6:0]	Output	ATB ID for current trace data on master port 1	ATCLK
ATREADYS	Output	ATB transfer ready on slave port	ATCLK
ATVALIDM0	Output	ATB Valid signals present on master port 0	ATCLK
ATVALIDM1	Output	ATB Valid signals present on master port 1	ATCLK

A.5 CoreSight synchronous bridge signals

Table A-5 shows the CoreSight synchronous bridge signals.

Table A-5 CoreSight synchronous bridge signals

Name	Type	Description	Clock domain
AFREADYS	Input	ATB data flush complete for the master port	ATCLK
AFVALIDM	Input	ATB data flush request for the master port	ATCLK
ATBYTESS[1:0]	Input	ATB number of valid bytes, LSB aligned, on the slave port	ATCLK
ATCLK	Input	ATB clock	N/A
ATCLKEN	Input	ATB clock enable	ATCLK
ATDATAS[31:0]	Input	ATB trace data on the slave port	ATCLK
ATIDS[6:0]	Input	ATB ID for current trace data on slave port	ATCLK
ATREADYM	Input	ATB transfer ready on master port	ATCLK
ATRESETn	Input	ATB reset for the ATCLK domain	ATCLK
ATVALIDS	Input	ATB valid signals present on slave port	ATCLK
SE	Input	Scan Enable	None
AFREADYM	Output	ATB data flush complete for the master port	ATCLK
AFVALIDS	Output	ATB data flush request for the master port	ATCLK
ATBYTESM[1:0]	Output	ATB number of valid bytes, LSB aligned, on the master port	ATCLK
ATDATAM[31:0]	Output	ATB trace data on the master port	ATCLK
ATIDM[6:0]	Output	ATB ID for current trace data on master port	ATCLK
ATREADYS	Output	ATB transfer ready on slave port	ATCLK
ATVALIDM	Output	ATB valid signals present on master port	ATCLK

A.6 CoreSight trace funnel signals

Table A-6 shows the CoreSight trace funnel signals.

Table A-6 CoreSight trace funnel signals

Name	Type	Description	Clock domain
AFREADYS0	Input	ATB Data flush complete for the slave port 0	ATCLK
AFREADYS1	Input	ATB Data flush complete for the slave port 1	ATCLK
AFREADYS2	Input	ATB Data flush complete for the slave port 2	ATCLK
AFREADYS3	Input	ATB Data flush complete for the slave port 3	ATCLK
AFREADYS4	Input	ATB Data flush complete for the slave port 4	ATCLK
AFREADYS5	Input	ATB Data flush complete for the slave port 5	ATCLK
AFREADYS6	Input	ATB Data flush complete for the slave port 6	ATCLK
AFREADYS7	Input	ATB Data flush complete for the slave port 7	ATCLK
AFVALIDM	Input	ATB Data flush request for the master port	ATCLK
ATBYTESS0[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 0	ATCLK
ATBYTESS1[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 1	ATCLK
ATBYTESS2[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 2	ATCLK
ATBYTESS3[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 3	ATCLK
ATBYTESS4[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 4	ATCLK
ATBYTESS5[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 5	ATCLK
ATBYTESS6[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 6	ATCLK
ATBYTESS7[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port 7	ATCLK
ATCLK	Input	ATB clock	N/A
ATCLKEN	Input	ATB clock enable	ATCLK
ATDATAS0[31:0]	Input	ATB Trace data on the slave port 0	ATCLK
ATDATAS1[31:0]	Input	ATB Trace data on the slave port 1	ATCLK
ATDATAS2[31:0]	Input	ATB Trace data on the slave port 2	ATCLK
ATDATAS3[31:0]	Input	ATB Trace data on the slave port 3	ATCLK
ATDATAS4[31:0]	Input	ATB Trace data on the slave port 4	ATCLK
ATDATAS5[31:0]	Input	ATB Trace data on the slave port 5	ATCLK

Table A-6 CoreSight trace funnel signals (continued)

Name	Type	Description	Clock domain
ATDATAS6[31:0]	Input	ATB Trace data on the slave port 6	ATCLK
ATDATAS7[31:0]	Input	ATB Trace data on the slave port 7	ATCLK
ATIDS0[6:0]	Input	ATB ID for current trace data on slave port 0	ATCLK
ATIDS1[6:0]	Input	ATB ID for current trace data on slave port 1	ATCLK
ATIDS2[6:0]	Input	ATB ID for current trace data on slave port 2	ATCLK
ATIDS3[6:0]	Input	ATB ID for current trace data on slave port 3	ATCLK
ATIDS4[6:0]	Input	ATB ID for current trace data on slave port 4	ATCLK
ATIDS5[6:0]	Input	ATB ID for current trace data on slave port 5	ATCLK
ATIDS6[6:0]	Input	ATB ID for current trace data on slave port 6	ATCLK
ATIDS7[6:0]	Input	ATB ID for current trace data on slave port 7	ATCLK
ATREADYM	Input	ATB transfer ready on master port	ATCLK
ATRESETn	Input	ATB Reset for the ATCLK domain	ATCLK
ATVALIDS0	Input	ATB Valid signals present on slave port 0	ATCLK
ATVALIDS1	Input	ATB Valid signals present on slave port 1	ATCLK
ATVALIDS2	Input	ATB Valid signals present on slave port 2	ATCLK
ATVALIDS3	Input	ATB Valid signals present on slave port 3	ATCLK
ATVALIDS4	Input	ATB Valid signals present on slave port 4	ATCLK
ATVALIDS5	Input	ATB Valid signals present on slave port 5	ATCLK
ATVALIDS6	Input	ATB Valid signals present on slave port 6	ATCLK
ATVALIDS7	Input	ATB Valid signals present on slave port 7	ATCLK
PADDRDBG[11:2]	Input	Debug APB address bus	PCLKDBG
PADDRDBG31	Input	Debug APB programming origin, HIGH for off-chip	PCLKDBG
PCLKDBG	Input	Debug APB clock	N/A
PCLKENDBG	Input	Debug APB clock enable	PCLKDBG
PENABLEDBG	Input	Debug APB enable signal, indicates second and subsequent cycles	PCLKDBG
PRESETDBGn	Input	Debug APB reset	PCLKDBG
PSELDBG	Input	Debug APB component select	PCLKDBG

Table A-6 CoreSight trace funnel signals (continued)

Name	Type	Description	Clock domain
PWDATADBG[31:0]	Input	Debug APB write data bus	PCLKDBG
PWRITEDBG	Input	Debug APB write transfer	PCLKDBG
SE	Input	Scan Enable	None
AFREADYM	Output	ATB Data flush complete for the master port	ATCLK
AFVALIDS0	Output	ATB Data flush request for the slave port 0	ATCLK
AFVALIDS1	Output	ATB Data flush request for the slave port 1	ATCLK
AFVALIDS2	Output	ATB Data flush request for the slave port 2	ATCLK
AFVALIDS3	Output	ATB Data flush request for the slave port 3	ATCLK
AFVALIDS4	Output	ATB Data flush request for the slave port 4	ATCLK
AFVALIDS5	Output	ATB Data flush request for the slave port 5	ATCLK
AFVALIDS6	Output	ATB Data flush request for the slave port 6	ATCLK
AFVALIDS7	Output	ATB Data flush request for the slave port 7	ATCLK
ATBYTESM[1:0]	Output	ATB Number of valid bytes, LSB aligned, on the master port	ATCLK
ATDATAM[31:0]	Output	ATB Trace data on the master port	ATCLK
ATIDM[6:0]	Output	ATB ID for current trace data on master port	ATCLK
ATREADYS0	Output	ATB transfer ready on slave port 0	ATCLK
ATREADYS1	Output	ATB transfer ready on slave port 1	ATCLK
ATREADYS2	Output	ATB transfer ready on slave port 2	ATCLK
ATREADYS3	Output	ATB transfer ready on slave port 3	ATCLK
ATREADYS4	Output	ATB transfer ready on slave port 4	ATCLK
ATREADYS5	Output	ATB transfer ready on slave port 5	ATCLK
ATREADYS6	Output	ATB transfer ready on slave port 6	ATCLK
ATREADYS7	Output	ATB transfer ready on slave port 7	ATCLK
ATVALIDM	Output	ATB Valid signals present on master port	ATCLK
PRDATADBG[31:0]	Output	Debug APB read data bus	PCLKDBG
PREADYDBG	Output	Debug APB Ready signal	PCLKDBG

A.7 CoreSight TPIU signals

Table A-7 shows the CoreSight *Trace Port Interface Unit* (TPIU) signals.

Table A-7 CoreSight TPIU signals

Name	Type	Description	Clock domain
AFREADYS	Input	ATB Data flush complete for the master port	ATCLK
ATBYTESS[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port	ATCLK
ATCLK	Input	ATB clock	N/A
ATCLKEN	Input	ATB clock enable	ATCLK
ATDATAS[31:0]	Input	ATB Trace data on the slave port	ATCLK
ATIDS[6:0]	Input	ATB ID for current trace data on slave port	ATCLK
ATRESETn	Input	ATB Reset for the ATCLK domain	ATCLK
ATVALIDS	Input	ATB Valid signals present on slave port	ATCLK
EXTCTLIN[7:0]	Input	External control input	ATCLK
FLUSHIN	Input	Flush input from the CTI	ATCLK/None
PADDRDBG[11:0]	Input	Debug APB address bus	PCLKDBG
PADDRDBG31	Input	Debug APB programming origin, HIGH for off-chip	PCLKDBG
PCLKDBG	Input	Debug APB clock	N/A
PCLKENDBG	Input	Debug APB clock enable	PCLKDBG
PENABLEDBG	Input	Debug APB enable signal, indicates second and subsequent cycles	PCLKDBG
PRESETDBGn	Input	Debug APB reset	PCLKDBG
PSELDBG	Input	Debug APB component select	PCLKDBG
PWDATADBG[31:0]	Input	Debug APB write data bus	PCLKDBG
PWRITEDBG	Input	Debug APB write transfer	PCLKDBG
SE	Input	Scan Enable	None
TPCTL	Input	Tie-off to report presence of **TRACECTL**, static value	ATCLK
TPMAXDATASIZE[4:0]	Input	Tie-off to report maximum number of pins on **TRACEDATA**, static value	ATCLK

Table A-7 CoreSight TPIU signals (continued)

Name	Type	Description	Clock domain
TRACECLKIN	Input	Trace clock	N/A
TRESETn	Input	Trace clock reset	TRACECLKIN
TRIGIN	Input	Trigger input from the CTI	ATCLK/None
AFVALIDS	Output	ATB Data flush request for the master port	ATCLK
ATREADYS	Output	ATB transfer ready on slave port	ATCLK
EXTCTLOUT[7:0]	Output	External control output	ATCLK
FLUSHINACK	Output	Flush input acknowledgement	ATCLK
PRDATADBG[31:0]	Output	Debug APB read data bus	PCLKDBG
PREADYDBG	Output	Debug APB Ready signal	PCLKDBG
TRACECLK	Output	Exported Trace Port Clock, **TRACECLKIN** divided by 2	TRACECLKIN
TRACECTL	Output	Trace Port Control	TRACECLKIN
TRACEDATA[31:0]	Output	Trace Port Data	TRACECLKIN
TRIGINACK	Output	Trigger input acknowledgement	ATCLK

A.8 CoreSight ETB signals

Table A-8 shows the CoreSight *Embedded Trace Bus* (ETB) signals.

Table A-8 CoreSight ETB signals

Name	Type	Description	Clock domain
AFREADYS	Input	ATB Data flush complete for the master port	ATCLK
ATBYTESS[1:0]	Input	ATB Number of valid bytes, LSB aligned, on the slave port	ATCLK
ATCLK	Input	ATB clock	N/A
ATCLKEN	Input	ATB clock enable	ATCLK
ATDATAS[31:0]	Input	ATB Trace data on the slave port	ATCLK
ATIDS[6:0]	Input	ATB ID for current trace data on slave port	ATCLK
ATRESETn	Input	ATB Reset for the ATCLK domain	ATCLK
ATVALIDS	Input	ATB Valid signals present on slave port	ATCLK
FLUSHIN	Input	Flush input from the CTI	ATCLK/None
MBISTADDR[CSETB_RAM_ADRW-1:0]	Input	Memory BIST address	ATCLK
MBISTCE	Input	Memory BIST chip enable	ATCLK
MBISTDIN[31:0]	Input	Memory BIST data in	ATCLK
MBISTWE	Input	Memory BIST write enable	ATCLK
MTESTON	Input	Memory BIST test is enabled	ATCLK
PADDRDBG[11:2]	Input	Debug APB address bus	PCLKDBG
PADDRDBG31	Input	Debug APB programming origin, HIGH for off-chip	PCLKDBG
PCLKDBG	Input	Debug APB clock	N/A
PCLKENDBG	Input	Debug APB clock enable	PCLKDBG
PENABLEDBG	Input	Debug APB enable signal, indicates second and subsequent cycles	PCLKDBG
PRESETDBGn	Input	Debug APB reset	PCLKDBG

Table A-8 CoreSight ETB signals (continued)

Name	Type	Description	Clock domain
PSELDBG	Input	Debug APB component select	PCLKDBG
PWDATADBG[31:0]	Input	Debug APB write data bus	PCLKDBG
PWRITEDBG	Input	Debug APB write transfer	PCLKDBG
SE	Input	Scan Enable	
TRIGIN	Input	Trigger input from the CTI	ATCLK/None
ACQCOMP	Output	Trace acquisition complete	ATCLK
AFVALIDS	Output	ATB Data flush request for the master port	ATCLK
ATREADYS	Output	ATB transfer ready on slave port	ATCLK
FLUSHINACK	Output	Flush input acknowledgement	ATCLK
FULL	Output	CSETB RAM overflow or wrapped around	ATCLK
MBISTDOUT[31:0]	Output	Memory BIST data out	ATCLK
PRDATADBG[31:0]	Output	Debug APB read data bus	PCLKDBG
PREADYDBG	Output	Debug APB Ready signal	PCLKDBG
TRIGINACK	Output	Trigger input acknowledgement	ATCLK

A.9 Internal DAP bus interface

Table A-9 shows the DAP internal interface signals. Signal direction is with respect to the AHB-AP. All of these signals are in the DAPCLK domain.

Table A-9 DAP port signals

Name	Direction from DP	Description
DAPRESETn	Input[a]	DAP Reset, active LOW Power on Reset. This must be the same as **PRESETDBGn**.
DAPCLK	Input[a]	DAP internal APB Clock. This must be the same as **PCLKDBG**. See Appendix B *CoreSight Components and Clock Domains* for a list of clocks.
DAPCLKEN	Input[a]	Enable term for DAPCLK domain. This must be the same as **PCLKENDBG**.
DAPABORT	Output	Abort the current transfer. DAPREADY is returned HIGH by the currently selected access port.
DAPSLVERR	Input	Access port slave error response.
DAPADDR[31:0]	Output	DAP address bus. See *DAP access port address decoding* on page 3-7 for a description of the encoding.
DAPSEL	Output	Select signal generated from the DAP decoder to each access port. This signal indicates that the slave component is selected and a data transfer is required. There is a **DAPSELx** signal for each slave. **DAPSELx** is not generated by the driving debug port and originates from the address decoder that monitors **DAPADDR**[31:24].
DAPENABLE	Output	The enable signal is used to indicate the second and subsequent cycles of a DAP transfer from debug port to access port.
DAPWRITE	Output	When HIGH indicates a DAP write access from debug port to access port and when LOW a read access.
DAPWDATA[31:0]	Output	The write bus is driven by the debug port block during write cycles, when **DAPWRITE** is HIGH.
DAPRDATA[31:0]	Input	The read bus is driven by the selected access port during read cycles, when **DAPWRITE** is LOW. It is sampled by the debug port on **DAPREADY** HIGH.
DAPREADY	Input	The ready signal is used by the access port to extend a DAP transfer by holding it LOW.

a. Always an input to debug ports and access ports.

Appendix B
CoreSight Components and Clock Domains

This appendix lists CoreSight components and their clock domains. It contains the following section:

- *CoreSight components and clock domains* on page B-2.

B.1 CoreSight components and clock domains

Table B-1 shows the components and clock domains.

Table B-1 CoreSight components and clock domains

CoreSight component	Clock domains	Comments
Debug Access Port (DAP)		
JTAG-DP	**DAPCLK**	Debug Access Port clock. Always equivalent to **PCLKDBG**.
	TCK	JTAG-DP interface.
JTAG-AP	**DAPCLK**	Debug Access Port clock. Always equivalent to **PCLKDBG**.
AHB-AP	**DAPCLK**	Debug Access Port clock. Always equivalent to **PCLKDBG**.
	HCLK	System AHB clock.
APB-AP	**DAPCLK**	Debug Access Port clock. Always equivalent to **PCLKDBG**. APB-AP requires **DAPCLK = PCLKDBG**.
	PCLKDBG	CoreSight Debug APB clock.
APB-Mux	**PCLKSYS**	System bus clock, **HCLK** for example.
	PCLKDBG	CoreSight Debug APB clock.
ROM table	-	-
Embedded Cross Trigger		
CTI	**CTICLK**	Cross Trigger Interface clock.
	PCLKDBG	CoreSight Debug APB clock.
CTM	**CTMCLK**	Cross Trigger Matrix clock.
CoreSight Sources		
AHB Trace Macrocell (HTM)	Separate documentation, see *Further reading* on page xxvii. **HTMCLK**	
	PCLKDBG	CoreSight Debug APB clock.
	ATCLK	AMBA trace bus clock.

Table B-1 CoreSight components and clock domains (continued)

CoreSight component	Clock domains	Comments
ETMs for core trace: CoreSight ETM9 CoreSight ETM11	Separate documentation, see *Further reading* on page xxvii. **ETMCLK**	
	PCLKDBG	CoreSight Debug APB clock.
	ATCLK	AMBA trace bus clock.
One-to-one ATB bridge	**ATCLK**	AMBA trace bus clock.
Replicator	**ATCLK**	AMBA trace bus clock.
Trace Funnel	**ATCLK**	Always synchronous, and greater than or equal to **PCLKDBG**.
	PCLKDBG	CoreSight Debug APB clock.
Trace Port Interface Unit	**PCLKDBG**	CoreSight Debug APB clock.
	ATCLK	Always synchronous, and greater than or equal to **PCLKDBG**.
	TRACECLKIN	CoreSight Trace Port Interface Unit off-chip trace data clock.
Embedded Trace Buffer	**PCLKDBG**	CoreSight Debug APB clock.
	ATCLK	CoreSight AMBA Trace Bus clock. Always synchronous, and greater than or equal to **PCLKDBG**.

Appendix C
Serial Wire Debug and JTAG Trace Connector

This appendix describes the SWD and JTAG trace connector. It contains the following sections:

C.1 About the SWD and JTAG trace connector

The SWD and JTAG trace connector is used for debug targets requiring JTAG, SWD, SWO, and low bandwidth trace connectivity.

This appendix describes 10-way and 20-way connectors that are mounted on debug target boards. These are specified as 0.050 inch pitch two-row pin headers, Samtec FTSH or equivalent. See `www.samtec.com`.

The connectors are grouped into compatible sets according to the functions supported. Some targets support communication by both SWD and JTAG using the SWJ-DP block to switch between protocols.

C.2 Pinout details

The connector pin layouts are described in:

- *Combined pin names*
- *10-way connector pinouts* on page C-5
- *20-way connector pinouts including trace* on page C-6
- *20-way connector pinouts for legacy JTAG connections* on page C-10.

Note

The equivalent pin numbers on a CoreSight-compatible Mictor connector pinout are shown in the tables. This enables you to build a target where the debug communication signals are brought in parallel to both connectors so that the target can be debugged using either physical connector.

C.2.1 Combined pin names

Table C-1 shows a summary of the pin names.

Table C-1 Summary of pin names

Pin number	Combined pin name	Pin number	Combined pin name
1	VTref	11	Gnd/TgtPwr+Cap
2	**TMS/SWDIO**	12	**TraceClk/RTCK**
3	GND	13	Gnd/TgtPwr+Cap
4	**TCK/SWCLK**	14	**TraceD0/SWO**
5	GND	15	GND
6	**TDO/SWO/EXTa/TraceCtl**	16	**TraceD1/nTRST**
7	Key	17	GND
8	**TDI/EXTb/TraceCtl**	18	**TraceD2/TrigIn**
9	GNDDetect	19	GND
10	**TraceCtl/nRESET**	20	**TraceD3/TrigOut**

See Table C-6 on page C-12 for a generic description of the pins.

The following sections describe the use of pins 6, 8, 11, and 13:

- *EXTa and EXTb pins*
- *GND/TgtPwr+Cap pins*.

EXTa and EXTb pins

Some pins on the connector have functions that are only used for certain connection layouts. Where a pin is not required for a particular debug communication protocol, it can be reused for a user-defined target function. These pins are labeled **EXTa** and **EXTb** in the tables in this appendix. You must select any alternate functions carefully, and also consider the effects of connecting debug equipment capable of communicating using multiple protocols.

GND/TgtPwr+Cap pins

There are two usage options for these pins:

Target boards

Standard target boards can connect these two pins directly to signal ground, GND.

Some special target boards, typically those for evaluation or demonstration purposes, can use these pins to supply power to the target board. In this case, the target board must include capacitors between each of the pins and signal ground. The capacitors must be situated extremely close to the connector so that they maintain an effective AC ground, that is, high frequency signal return path. Typical values for the capacitors are 10nF.

Debug equipment

Debug communication equipment designed to work with special valuation or demonstration target boards provides operating current, typically up to 100mA, at a target-specific supply voltage, for example, 3.3V, 5V, or 9V. ARM Limited recommends that the debug equipment includes some protection if you connect it to standard target boards that have connected these pins directly to GND, for example, a current limiting circuit. This debug equipment must include capacitors between each of these power pins and signal ground. The capacitors must be situated extremely close to the connector so that they maintain an effective AC ground, that is, high frequency signal return path. Typical values for the capacitors are 10nF.

Standard debug communication equipment can connect these pins directly to GND. It is also possible for these pins to have only a high frequency signal return path to ground, using 10nF capacitors. This option is also compatible in the unlikely case where a target board has a connection between the debug connector **TgtPwr** pins, but is powered from another source.

C.2.2 10-way connector pinouts

There are two types of the pinout for a 10-pin connector, one supporting communication using SWD, and one using JTAG, and these are arranged to facilitate dynamic switching between the protocols.

SWD is the preferred protocol for debugging because it provides more data bandwidth over fewer pins, therefore freeing some for use by application functions. JTAG can be used where the target is communicating with a tool chain that does not support SWD, or with test tools performing board-level boundary scan testing, where it might be acceptable to sacrifice the functional pins multiplexed with JTAG.

Table C-2 shows the 10-way header for targets using SWD or JTAG for debug communication, and includes an optional *Serial Wire Output* (SWO) signal for conveying application and instrumentation trace.

Table C-2 10-way connector for SWD or JTAG systems

Pin name for SWD	Pin number		Pin name for JTAG	Pin number	
	10-way	Mictor		10-way	Mictor
VTref	1	12	VTref	1	12
SWDIO	2	17	**TMS**	2	17
GND	3	-	GND	3	-
SWCLK	4	15	**TCK**	4	15
GND	5	-	GND	5	-
SWO	6	11	**TDO**	6	11
Key	7	-	Key	7	-
NC/EXTb	8	19	**TDI**	8	19
GNDDetect	9	-	GNDDetect	9	-
nRESET	10	9	**nRESET**	10	9

The SWD layout is typically used in a CoreSight system that uses a SWJ-DP operating in SWD mode.

The JTAG layout is typically used in a CoreSight system including a JTAG-DP, or one with a SWJ-DP operating JTAG mode, possibly because it is cascaded with other JTAG TAPs.

——— **Note** ———

A target board can use this connector for performing board-level boundary scan but then switch its SWJ-DP into SWD mode for debugging according to the layout shown in Table C-2 on page C-5. This frees up pins 6 and 8 for either application functions or SWO.

You do not have to choose the switching mode at the time of chip or board development. The connector can be switched and the target board operated in either SWD or JTAG mode.

———————————

C.2.3 20-way connector pinouts including trace

20-way connectors include support for a narrow trace port, up to four data bits, operating at moderate speed, up to 100 MSamples/sec. They are described in:

- *Connector pinouts for trace systems using TraceCtl*
- *Connector pinouts for future systems* on page C-8.

Connector pinouts for trace systems using TraceCtl

Most existing trace systems operate their CoreSight TPIU in normal mode or bypass mode, or use a dedicated trace port that is equivalent to bypass mode. All of these require a **TraceCtl** signal for identifying trigger events and idle trace samples.

There are several possible pin locations for the **TraceCtl** signal according to which other signals a particular target requires. Trace collection equipment can be configured to accept any of these.

Table C-3 shows the 20-way header for targets using SWD or JTAG for debug communication, which might require both **nRESET**, pin 10, and an optional **SWO** signal for conveying application and instrumentation trace.

Table C-3 20-way connector for SWD or JTAG systems

Pin name for SWD with nRESET	Pin number		Pin name for SWD	Pin number		Pin name for JTAG	Pin number	
	20-way	Mictor		20-way	Mictor		20-way	Mictor
VTref	1	12	VTref	1	12	VTref	1	12
SWDIO	2	17	**SWDIO**	2	17	**TMS**	2	17
GND	3	-	GND	3	-	GND	3	-
SWCLK	4	15	**SWCLK**	4	15	**TCK**	4	15
GND	5	-	GND	5	-	GND	5	-
SWO/EXTa	6	(11)	**SWO/EXTa**	6	11	**TDO**	6	11
Key	7	-	Key	7	-	Key	7	-
TraceCtl	8	36	**NC/EXTb**	8	19	**TDI**	8	19
GNDDetect	9	-	GNDDetect	9	-	GNDDetect	9	-
nRESET	10	9	**TraceCtl**	10	36	**TraceCtl**	10	36
Gnd/TgtPwr +Cap	11	-	Gnd/TgtPwr +Cap	11	-	Gnd/TgtPwr +Cap	11	-
TraceClk	12	6	**TraceClk**	12	6	**TraceClk**	12	6
Gnd/TgtPwr +Cap	13	-	Gnd/TgtPwr +Cap	13	-	Gnd/TgtPwr +Cap	13	-
TraceD0	14	38	**TraceD0**	14	38	**TraceD0**	14	38
GND	15	-	GND	15	-	GND	15	-
TraceD1	16	28	**TraceD1**	16	28	**TraceD1**	16	28
GND	17	-	GND	17	-	GND	17	-
TraceD2	18	26	**TraceD2**	18	26	**TraceD2**	18	26
GND	19	-	GND	19	-	GND	19	-
TraceD3	20	24	**TraceD3**	20	24	**TraceD3**	20	24

The SWD layout with **nRESET** is typically used in a CoreSight system that uses a SWJ-DP operating in SWD mode, where it is necessary to operate that trace port in normal or bypass mode.

The SWD layout is typically used in a CoreSight system that uses a SWJ-DP operating in SWD mode, where it is necessary to operate that trace port in normal or bypass mode, but also necessary to communicate with the same target using JTAG.

The JTAG layout is typically used in a CoreSight system including a JTAG-DP, where it is necessary to operate that trace port in normal or bypass mode, or a system with a SWJ-DP operating in JTAG mode, possibly because it is cascaded with other JTAG TAPs.

——— **Note** ———

A target board can use this layout for performing board-level boundary scan, but then switch its SWJ-DP into SWD mode for debugging according to the layout shown in Table C-3 on page C-7. This frees up pins 6 and 8 for either application functions or SWO.

You do not have to choose the switching mode at the time of chip or board development. The connector can be switched and the target board operated in either SWD or JTAG mode.

———————————

Connector pinouts for future systems

Table C-4 on page C-9 shows the 20-way header for targets using SWD or JTAG for debug communication, and includes an optional **SWO** signal for conveying application/instrumentation trace. Alternatively, a target trace port operating in CoreSight normal or bypass modes might convey the TraceCtl signal on pin 6.

Both pin 6 and pin 8 in the SWD layout are shown with alternative extra signals, **EXTa** and **EXTb**. This enables flexibility to communicate other signals on these pins. For example, future target systems and trace equipment might convey two further trace data signals on these pins.

Table C-4 20-way connector for future SWD or JTAG systems

Pin name for SWD	Pin number		Pin name for JTAG	Pin number	
	20-way	Mictor		20-way	Mictor
VTref	1	12	VTref	1	12
SWDIO	2	17	**TMS**	2	17
GND	3	-	GND	3	-
SWCLK	4	15	**TCK**	4	15
GND	5	-	GND	5	-
SWO/EXTa/TraceCtl	6	11	**TDO**	6	11
Key	7	-	Key	7	-
NC/EXTb	8	(19)	**TDI**	8	19
GNDDetect	9	-	GNDDetect	9	-
nRESET	10	9	**nRESET**	10	9
Gnd/TgtPwr+Cap	11	-	Gnd/TgtPwr+Cap	11	-
TraceClk	12	6	**TraceClk**	12	6
Gnd/TgtPwr+Cap	13	-	Gnd/TgtPwr+Cap	13	-
TraceD0	14	38	**TraceD0**	14	38
GND	15	-	GND	15	-
TraceD1	16	28	**TraceD1**	16	28
GND	17	-	GND	17	-
TraceD2	18	26	**TraceD2**	18	26
GND	19	-	GND	19	-
TraceD3	20	24	**TraceD3**	20	24

The SWD layout is typically used in a CoreSight system that uses a SWJ-DP operating in SWD mode.

The JTAG layout is typically used in a CoreSight system including a JTAG-DP, or one with a SWJ-DP operating JTAG mode, possibly because it is cascaded with other JTAG TAPs. This layout s the recommended debug connection for a processor built with support for instruction trace, that is, including an ETM.

Note

A target board can use this layout for performing board-level boundary scan but then switch its SWJ-DP into SWD mode for debugging according to the layout shown in Table C-4 on page C-9. This frees up pins 6 and 8 for either application functions or SWO.

You do not have to choose the switching mode at the time of chip or board development. The connector can be switched and the target board operated in either SWD or JTAG mode.

C.2.4 20-way connector pinouts for legacy JTAG connections

Table C-5 shows the 20-way header for targets using legacy JTAG for debug communication. It includes support for adaptive clocking where the communication rate is throttled by the debug equipment waiting for each **TCK** change to be reflected from the target on **RTCK**. It includes an optional **SWO** signal for conveying application and instrumentation trace.

Table C-5 20-way connector for legacy JTAG connections

Pin name	Pin number	
	20-way	Mictor
VTref	1	12
TMS	2	17
GND	3	-
TCK	4	15
GND	5	-
TDO	6	11
Key	7	-

Table C-5 20-way connector for legacy JTAG connections (continued)

Pin name	Pin number	
	20-way	Mictor
TDI	8	19
GNDDetect	9	-
nRESET	10	9
Gnd/TgtPwr+Cap	11	-
RTCK	12	13
Gnd/TgtPwr+Cap	13	-
SWO	14	(38)
GND	15	-
nTRST	16	21
GND	17	-
DBGRQ/TrigIn	18	7
GND	19	-
DBGACK/TrigOut	20	8

C.3 Signal definitions

Table C-6 shows the generic trace connector signal definitions.

Table C-6 Generic signal definitions

Signal	Type	Description
VTref	Output	Debug and trace port reference voltage.
TMS	Input	Standard JTAG pin.
TCK	Input	Standard JTAG pin.
TDO	Output	User-definable JTAG pin.
Key	-	Key pin. Removed from target connector, or physically blocked on debug connector.
TDI	Input	User-definable JTAG pin.
GndDetect	-	Can be used by target system for debugger presence detection.
nRESET	-	Target reset signal.
TraceClk	Output	Clock trace pin.
TraceD[n]	Output	Trace data pins.

Note

See *Combined pin names* on page C-3 for information on how these signals are used in the SWD and JTAG trace connector.

Appendix D
Deprecated SWJ-DP Switching Sequences

This appendix describes the switching sequences used in earlier versions of the *Serial Wire JTAG Debug Port* (SWJ-DP). It contains the following section:

- *About the deprecated SWJ-DP switching sequences* on page D-2.

D.1 About the deprecated SWJ-DP switching sequences

An earlier version of the SWJ-DP uses different select sequences for switching between JTAG and SWD, and between SWD and JTAG.

The earlier version can be identified as follows:

- the Version field in the DeviceID code of JTAG-DP is `0x1`
- the Version field in the DeviceID code of SW-DP being `0x0`.

Table D-1 shows the deprecated switching sequences.

Table D-1 Deprecated switching sequences

Deprecated switching sequence	MSB first	LSB first
JTAG to SWD	0110110110110111 or 16'h6DB7	16'hEDB6
SWD to JTAG	0111010101110101 or 16'h7575	16'hAEAE

Glossary

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

Abort

A mechanism that indicates to a core that it must halt execution of an attempted illegal memory access. An abort can be caused by the external or internal memory system as a result of attempting to access invalid instruction or data memory. An abort is classified as either a Prefetch or Data Abort, and an internal or External Abort.

See also Data Abort, External Abort and Prefetch Abort.

Advanced eXtensible Interface (AXI)

This is a bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)

The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Advanced Microcontroller Bus Architecture (AMBA)

AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

Advanced Peripheral Bus (APB)

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

See also Advanced High-performance Bus.

AHB *See* Advanced High-performance Bus.

AHB Access Port (AHB-AP)

An optional component of the DAP that provides an AHB interface to an SoC.

AHB-AP *See* AHB Access Port.

AHB-Lite AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.

AMBA *See* Advanced Microcontroller Bus Architecture.

AMBA Trace Bus (ATB)

A bus used by trace devices to share CoreSight capture resources.

APB *See* Advanced Peripheral Bus.

Application Specific Integrated Circuit (ASIC)

An integrated circuit that has been designed to perform a specific application function. It can be custom-built or mass-produced.

Architecture	The organization of hardware and/or software that characterizes a processor and its attached components, and enables devices with similar characteristics to be grouped together when describing their behavior, for example, Harvard architecture, instruction set architecture, ARMv6 architecture.
ASIC	*See* Application Specific Integrated Circuit.
ATB	*See* AMBA Trace Bus.
ATB bridge	A synchronous ATB bridge provides a register slice to facilitate timing closure through the addition of 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 is intended to support connection of components with ATB ports residing in different clock domains.
ATPG	*See* Automatic Test Pattern Generation.
Automatic Test Pattern Generation (ATPG)	The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.
AXI	*See* Advanced eXstensible Interface.
Banked registers	Those physical registers whose use is defined by the current processor mode. The banked registers are r8 to r14.
Beat	Alternative word for an individual transfer within a burst. For example, an INCR4 burst comprises four beats. *See also* Burst.
Boundary scan chain	A boundary scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.
Breakpoint	A breakpoint is a mechanism provided by debuggers to identify an instruction at which program execution is to be halted. Breakpoints are inserted by the programmer to enable inspection of register contents, memory locations, variable values at fixed points in the program execution to test that the program is operating correctly. Breakpoints are removed after the program is successfully tested. *See also* Watchpoint.

Burst	A group of transfers to consecutive addresses. Because the addresses are consecutive, there is no requirement to supply an address for any of the transfers after the first one. This increases the speed at which the group of transfers can occur. Bursts over AHB buses are controlled using the **HBURST** signals to specify if transfers are single, four-beat, eight-beat, or 16-beat bursts, and to specify how the addresses are incremented. *See also* Beat.
Byte	An 8-bit data item.
Byte lane strobe	An AHB signal, **HBSTRB**, that is used for unaligned or mixed-endian data accesses to determine which byte lanes are active in a transfer. One bit of **HBSTRB** corresponds to eight bits of the data bus.
Clock gating	Gating a clock signal for a macrocell with a control signal and using the modified clock that results to control the operating state of the macrocell.
Cold reset	Also known as power-on reset. Starting the processor by turning power on. Turning power off and then back on again clears main memory and many internal settings. Some program failures can lock up the processor and require a cold reset to enable the system to be used again. In other cases, only a warm reset is required. *See also* Warm reset.
Coprocessor	A processor that supplements the main processor. It carries out additional functions that the main processor cannot perform. Usually used for floating-point math calculations, signal processing, or memory management.
Core	A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.
Core reset	*See* Warm reset.
Cross Trigger Interface (CTI)	Part of an Embedded Cross Trigger device. The CTI provides the interface between a core/ETM and the CTM within an ECT.
Cross Trigger Matrix (CTM)	The CTM combines the trigger requests generated from CTIs and broadcasts them to all CTIs as channel triggers within an Embedded Cross Trigger device.
CTI	*See* Cross Trigger Interface.
CTM	*See* Cross Trigger Matrix.
CoreSight	The infrastructure for monitoring, tracing, and debugging a complete system on chip.

DBGTAP	*See* Debug Test Access Port.
Debug Access Port (DAP)	A TAP block that acts as an AMBA (AHB or AHB-Lite) master for access to a system bus. The DAP is the term used to encompass a set of modular blocks that support system wide debug. The DAP is a modular component, intended to be extendable to support optional access to multiple systems such as memory mapped AHB and CoreSight APB through a single debug interface.
Debugger	A debugging system that includes a program, used to detect, locate, and correct software faults, together with custom hardware that supports software debugging.
Debug Test Access Port (DBGTAP)	The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **DBGTDI**, **DBGTDO**, **DBGTMS**, and **TCK**. The optional terminal is **TRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.
ECT	*See* Embedded Cross Trigger.
Embedded Cross Trigger (ECT)	The ECT is a modular component to support the interaction and synchronization of multiple triggering events with an SoC.
EmbeddedICE logic	An on-chip logic block that provides TAP-based debug support for ARM processor cores. It is accessed through the TAP controller on the ARM core using the JTAG interface.
EmbeddedICE-RT	The JTAG-based hardware provided by debuggable ARM processors to aid debugging in real-time.
Embedded Trace Buffer	The 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 core, outputs instruction and data trace information on a trace port. The ETM provides processor driven trace through a trace port compliant to the ATB protocol.
ETB	*See* Embedded Trace Buffer.
ETM	*See* Embedded Trace Macrocell.
Event	1 (Simple) An observable condition that can be used by an ETM to control aspects of a trace. 2 (Complex) A boolean combination of simple events that is used by an ETM to control aspects of a trace.

External Abort
An indication from an external memory system to a core that it must halt execution of an attempted illegal memory access. An External Abort is caused by the external memory system as a result of attempting to access invalid memory.

See also Abort, Data Abort and Prefetch Abort.

Formatter
The formatter is an internal input block in the ETB and TPIU that embeds the trace source ID within the data to create a single trace stream.

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

Halfword
A 16-bit data item.

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.

Illegal instruction
An instruction that is architecturally Undefined.

Implementation-defined
Means that the behavior is not architecturally defined, but should be defined and documented by individual implementations.

Implementation-specific
Means that the behavior is not architecturally defined, and does not have to be documented by individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

Imprecise tracing
A filtering configuration where instruction or data tracing can start or finish earlier or later than expected. Most 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.

Intelligent Energy Management (IEM)
A technology that enables dynamic voltage scaling and clock frequency variation to be used to reduce power consumption in a device.

Internal scan chain	A series of registers connected together to form a path through a device, used during production testing to import test patterns into internal nodes of the device and export the resulting values.
Joint Test Action Group (JTAG)	The name of the organization that developed standard IEEE 1149.1. This standard defines a boundary-scan architecture used for in-circuit testing of integrated circuit devices. It is commonly known by the initials JTAG.
JTAG	*See* Joint Test Action Group.
JTAG Access Port (JTAG-AP)	An optional component of the DAP that provides JTAG access to on-chip components, operating as a JTAG master port to drive JTAG chains throughout a SoC.
JTAG-AP	*See* JTAG Access Port.
JTAG Debug Port (JTAG-DP)	An optional external interface for the DAP that provides a standard JTAG interface for debug access.
JTAG-DP	*See* JTAG Debug Port.
Macrocell	A complex logic block with a defined interface and behavior. A typical VLSI system comprises several macrocells (such as a processor, an ETM, and a memory block) plus application-specific logic.
Microprocessor	*See* Processor.
Monitor debug-mode	One of two mutually exclusive debug modes. In Monitor debug-mode the processor enables a software abort handler provided by the debug monitor or operating system debug task. When a breakpoint or watchpoint is encountered, this enables vital system interrupts to continue to be serviced while normal program execution is suspended. *See also* Halt mode.
Multi-ICE	A JTAG-based tool for debugging embedded systems.
Multi-layered	An AMBA scheme to break a bus into segments that are controlled in access. This enables local masters to reduce lock overhead.
Multi master	An AMBA bus sharing scheme (not in AMBA Lite) where different masters can gain a bus lock (Grant) to access the bus in an interleaved fashion.
Power-on reset	*See* Cold reset.

Prefetch Abort	An indication from a memory system to a core that it must halt execution of an attempted illegal memory access. 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.
Processor	A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.
Read	Reads are defined as memory operations that have the semantics of a load. That is, the ARM instructions LDM, LDRD, LDC, LDR, LDRT, LDRSH, LDRH, LDRSB, LDRB, LDRBT, LDREX, RFE, STREX, SWP, and SWPB, and the Thumb instructions LDM, LDR, LDRSH, LDRH, LDRSB, LDRB, and POP. Java instructions that are accelerated by hardware can cause a number of reads to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.
RealView ICE	A system for debugging embedded processor cores using a JTAG interface.
Replicator	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	A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.
SBO	*See* Should Be One.
SBZ	*See* Should Be Zero.
SBZP	*See* Should Be Zero or Preserved.
Scan chain	A scan chain is made up of serially-connected devices that implement boundary scan technology using a standard JTAG TAP interface. Each device contains at least one TAP controller containing shift registers that form the chain connected between **TDI** and **TDO**, through which test data is shifted. Processors can contain several shift registers to enable you to access selected parts of the device.
SCREG	The currently selected scan chain number in an ARM TAP controller.
Serial Wire Debug Port	An optional external interface for the DAP that provides a serial-wire bidirectional debug interface.

Should Be One (SBO)	Should be written as 1 (or all 1s for bit fields) by software. Writing a 0 produces Unpredictable results.
Should Be Zero (SBZ)	Should be written as 0 (or all 0s for bit fields) by software. Writing a 1 produces Unpredictable results.
Should Be Zero or Preserved (SBZP)	Should be written as 0 (or all 0s for bit fields) by software, or preserved by writing the same value back that has been previously read from the same field on the same processor.
SW-DP	*See* Serial Wire Debug Port.
TAP	*See* Test access port.
TCD	*See* Trace Capture Device.
Test Access Port (TAP)	The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **TRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.
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 driver	A Remote Debug Interface target that controls a piece of trace hardware. That is, the trigger macrocell, trace macrocell, and trace capture tool.
Trace funnel	A device that combines multiple trace sources onto a single bus.
Trace hardware	A term for a device that contains an Embedded Trace Macrocell.
Trace port	A port on a device, such as a processor or ASIC, used to output trace information.
Trace Port Analyzer (TPA)	A hardware device that captures trace information output on a trace port. This can be a low-cost product designed specifically for trace acquisition, or a logic analyzer.
Trace Port Interface Unit (TPIU)	The TPIU is used to drain trace data and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.

Undefined	Indicates an instruction that generates an Undefined instruction trap. See the *ARM Architecture Reference Manual* for more details on ARM exceptions.
UNP	*See* Unpredictable.
Unpredictable	Means that the behavior of the ETM cannot be relied upon. 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 ETM is capable of causing the core to enter debug state, and external outputs may be used for other purposes.
Unpredictable	For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.
Warm reset	Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.
Watchpoint	A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to allow inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.
Word	A 32-bit data item.
Write	Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH. Java instructions that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.
Write buffer	A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.