CoreSight™ DAP-Lite

Technical Reference Manual

ARM®

ARM DDI 0316D

CoreSight DAP-Lite
Technical Reference Manual

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
06 January 2006	A	Non-Confidential	First release.
14 December 2006	B	Non-Confidential	Second release.
19 October 2007	C	Non-Confidential	Alignment with *ARM Debug Interface v5 Architecture Specification*. Additional corrections and enhancements.
01 May 2008	D	Non-Confidential	Fourth release.

Proprietary Notice

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com

Contents
CoreSight DAP-Lite Technical Reference Manual

Preface
About this book xii
Feedback xvi

Chapter 1 Introduction
1.1 About the DAP-Lite 1-2
1.2 DAP-Lite structure 1-3
1.3 DAP-Lite control flow 1-5
1.4 DAP-Lite block summary 1-6

Chapter 2 Functional Description
2.1 About the Debug Port 2-2
2.2 SWJ-DP 2-3
2.3 JTAG-DP 2-11
2.4 SW-DP 2-13
2.5 Common debug port features and registers 2-19
2.6 Access ports 2-31
2.7 APB-AP 2-32
2.8 APB-Mux 2-41
2.9 ROM table 2-46

2.10 Authentication requirements 2-49
2.11 Clocks and resets 2-50
2.12 Connections to debug components and system interfaces 2-51

Chapter 3 Programmer's Model
3.1 About the programmer's model 3-2

Appendix A DAP-Lite Ports
A.1 CoreSight DAP signals A-2

Appendix B Revisions

Glossary

List of Tables
CoreSight DAP-Lite Technical Reference Manual

	Change History	ii
Table 1-1	DAP-Lite block summary	1-6
Table 2-1	JTAG-DP physical interface	2-12
Table 2-2	JTAG-DP registers	2-12
Table 2-3	Terms used in SW-DP timing	2-16
Table 2-4	Summary of Debug Port registers	2-21
Table 2-5	Identification Code Register bit assignments	2-22
Table 2-6	JEDEC JEP-106 manufacturer ID code, with ARM Limited values	2-23
Table 2-7	Control/Status Register bit assignments	2-24
Table 2-8	AP Select Register bit assignments	2-26
Table 2-9	Wire Control Register bit assignments	2-28
Table 2-10	Turnaround tristate period field bit definitions	2-29
Table 2-11	Wire operating mode bit definitions	2-29
Table 2-12	APB-AP other ports	2-32
Table 2-13	APB-AP registers	2-33
Table 2-14	APB Control/Status Word Register bit assignments	2-35
Table 2-15	APB-AP Transfer Address Register bit assignments	2-37
Table 2-16	ABP-AP Data Read/Write Register bit assignments	2-37
Table 2-17	APB-AP Banked Data Registers bit assignments	2-38
Table 2-18	Debug APB ROM Address Register bit assignments	2-38
Table 2-19	APB-AP Identification Register bit assignments	2-39

Table 2-20 APB-Mux miscellaneous signals 2-42
Table 2-21 ROM table registers 2-46
Table 2-22 ROM table entries bit assignments 2-48
Table A-1 CoreSight DAP signals A-2
Table B-1 Differences between issue C and issue D B-1

List of Figures
CoreSight DAP-Lite Technical Reference Manual

	Key to timing diagram conventions	xiv
Figure 1-1	DAP-Lite structure	1-3
Figure 1-2	DAP-Lite control flow	1-5
Figure 2-1	SWJ-DP external connections	2-4
Figure 2-2	SWJ-DP signal clamping	2-6
Figure 2-3	SWD and JTAG select state diagram	2-8
Figure 2-4	SW-DP acknowledgement timing	2-16
Figure 2-5	SW-DP to DAP bus timing for write	2-17
Figure 2-6	SW-DP to DAP bus timing for read	2-17
Figure 2-7	SW-DP idle timing	2-18
Figure 2-8	Identification Code Register bit assignments	2-22
Figure 2-9	Control/Status Register bit assignments	2-23
Figure 2-10	AP Select Register bit assignments	2-26
Figure 2-11	Wire Control Register bit assignments	2-28
Figure 2-12	APB-AP functional blocks	2-32
Figure 2-13	APB-AP Control/Status Word Register bit assignments	2-34
Figure 2-14	APB-AP Transfer Address Register bit assignments	2-36
Figure 2-15	Debug APB ROM Address Register bit assignments	2-38
Figure 2-16	APB-AP Identification Register bit assignments	2-39
Figure 2-17	APB-Mux block diagram	2-41
Figure 2-18	APB-Mux integrated into the DAP-Lite	2-41

Figure 2-19 APB-Mux domains 2-44
Figure 2-20 APB-Mux power domain separation 2-45
Figure 2-21 Debug trace with a single core 2-51

Preface

This preface introduces the *CoreSight DAP-Lite Technical Reference Manual*. It contains the following sections:

- *About this book* on page xii
- *Feedback* on page xvi.

About this book

This is the *Technical Reference Manual* (TRM) for the CoreSight *Debug Access Port Lite* (DAP-Lite).

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, 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 book is written for the following target audience:

- Hardware and software engineers who want to incorporate a DAP-Lite component into their design and perform debug functions within an ASIC.
- Software engineers writing tools to use the DAP-Lite.

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

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for a high-level view of the DAP-Lite and a description of its features.

Chapter 2 *Functional Description*

Read this chapter for a description of the major components of the DAP-Lite and how they operate.

Chapter 3 *Programmer's Model*

Read this chapter for description of the DAP-Lite registers.

Appendix A *DAP-Lite Ports*

Read this appendix for a description of the DAP-Lite input and output signals.

Appendix B *Revisions*

Read this appendix for a description of the changes specific to this issue of the book.

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

Conventions

Conventions that this book can use are described in:

- *Typographical*
- *Timing diagrams*
- *Signals* on page xiv.

Typographical

The typographical conventions are:

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace` Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`mono`space Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic` Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold` Denotes language keywords when used outside example code.

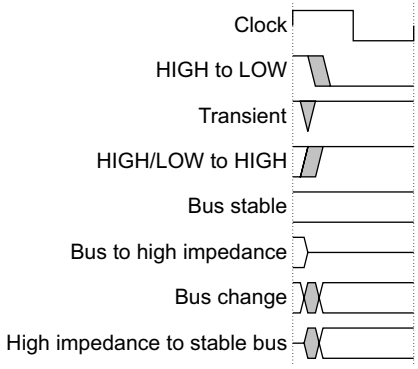
< and > Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>
```

Timing diagrams

The figure named *Key to timing diagram conventions* on page xiv 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
- LOW for active-LOW signals.

Lower-case n At the start or end of a signal name denotes an active-LOW signal.

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

Prefix AR Denotes AXI read address channel signals.

Prefix AW Denotes AXI write address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix P Denotes *Advanced Peripheral Bus* (APB) signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Further reading

This section lists publications by ARM and by third parties.

See `http://infocenter.arm.com/` for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *CoreSight System Design Guide*, ARM DGI 0012
- *CoreSight Architecture Specification*, ARM IHI 0029
- *CoreSight Components Technical Reference Manual*, ARM DDI 0314
- *CoreSight Components Implementation Guide*, ARM DII 0143
- *AMBA® 3 APB Protocol*, ARM IHI 0024
- *ARM Debug Interface v5 Architecture Specification*, ARM IHI 0031
- *RealView ICE User Guide*, ARM DUI 0155.

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

- the product name
- a concise explanation.

Feedback on this book

If you have any comments on this book, send an e-mail to `errata@arm.com`. Give:

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

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the CoreSight DAP-Lite. It contains the following sections:

- *About the DAP-Lite* on page 1-2
- *DAP-Lite structure* on page 1-3
- *DAP-Lite control flow* on page 1-5
- *DAP-Lite block summary* on page 1-6.

1.1 About the DAP-Lite

The *Debug Access Port* (DAP) is a implementation of an *ARM Debug Interface version 5* (ADIv5) comprising a number of components supplied in a single configuration. All the supplied components fit into the various architectural components for *Debug Ports* (DPs), which are used to access the DAP from an external debugger and *Access Ports* (APs), to access on-chip system resources.

The debug port and access ports together are referred to as the DAP.

The DAP-Lite contains the following components:

- *Serial Wire JTAG Debug Port* (SWJ-DP)
- *Advanced Peripheral Bus Access Port* (APB-AP)
- *Advanced Peripheral Bus Multiplexor* (APB-Mux)
- *Read Only Memory* (ROM) table.

Note

The DAP-Lite is a version of the DAP supplied with the CoreSight Design Kit, with fewer features. It does not contain a JTAG Access Port or an AHB Access Port, or any other access ports to directly access the system bus. For more information on the additional features and functionality provided by the DAP, see the *CoreSight Components Technical Reference Manual*.

1.2 DAP-Lite structure

Figure 1-1 shows the structure of the DAP-Lite.

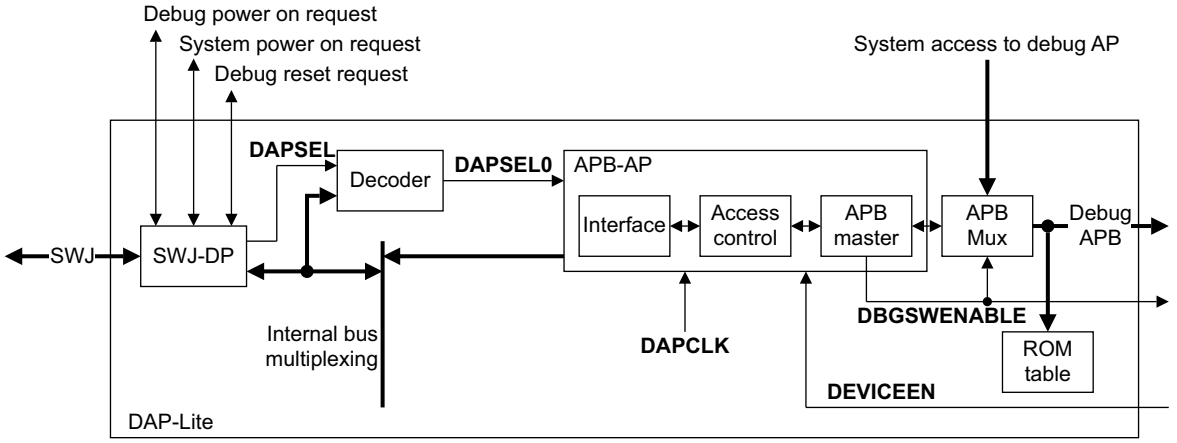


Figure 1-1 DAP-Lite structure

The DAP-Lite 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).
- Debug access using the APB-AP.
- 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. For more information about the ROM Table, see *ROM table* on page 2-46.

The debug port supplied with the DAP-Lite is the SWJ-DP. This is a combined debug port which can communicate in either JTAG or Serial Wire protocols as described in the *ARM Debug Interface v5 Architecture Specification*. It contains two debug ports, the SW-DP and the JTAG-DP that you can select through an interface sequence to move between debug port interfaces.

The access port supplied with the DAP-Lite is the APB-AP. The APB-AP provides an APB master in AMBA v3.0 for access to the Debug APB bus. This is compliant with the MEM-AP with a fixed transfer size of 32-bits.

The DAP-Lite blocks are described in more detail in:

- *About the Debug Port* on page 2-2
- *APB-AP* on page 2-32
- *APB-Mux* on page 2-41
- *ROM table* on page 2-46.

1.3 DAP-Lite control flow

The DAP-Lite, 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 APB-AP through a standard bus interface. 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 1-2 shows the flow of control for the DAP-Lite when used with an off-chip debugging unit such as RealView ICE.

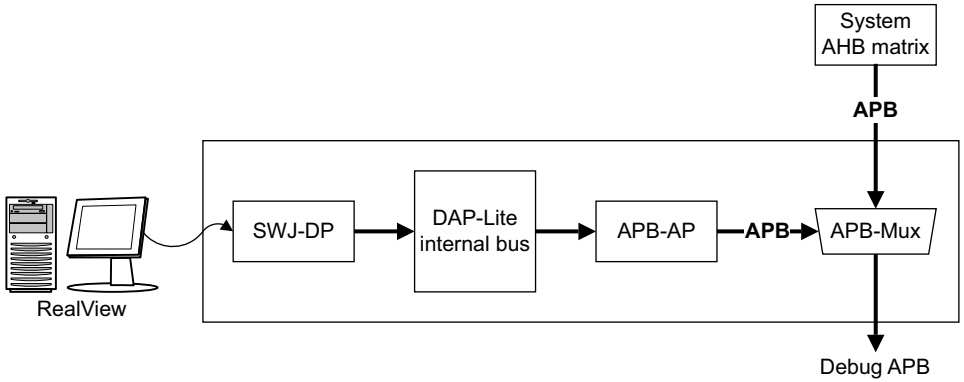


Figure 1-2 DAP-Lite control flow

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

The DAP-Lite internal bus implements memory mapped accesses to the components that are connected using the parallel address buses for read and write data. The debug port, SWJ-DP, is the bus master that initiates transactions on the DAP-Lite 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-Lite, both the bus master (debug port) and the slaves (access ports) contain registers. This DAP-Lite 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. The APB-AP can translate register interactions into transfers on the memory structure to which it is connected, the Debug APB in this case.

1.4 DAP-Lite block summary

Table 1-1 shows the main DAP-Lite blocks.

Table 1-1 DAP-Lite block summary

Block name	Description	Block version	Block revision
DAPAPBAP	APB Access Port	r0p1	1
DAPAPBMUX	APB Multiplexor	r0p1	-
DAPROM	ROM Table	r0p0	-
DAPSWJDP	Serial Wire and JTAG Debug Port:	r0p2	-
	• DAPSWDP	r0p2	2
	• DAPJTAGDP	r0p4	4

Note

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

Chapter 2
Functional Description

This chapter describes the major components of the CoreSight DAP-Lite, and how they operate. It contains the following sections:

- *About the Debug Port* on page 2-2
- *SWJ-DP* on page 2-3
- *JTAG-DP* on page 2-11
- *SW-DP* on page 2-13
- *Common debug port features and registers* on page 2-19
- *Access ports* on page 2-31
- *APB-AP* on page 2-32
- *APB-Mux* on page 2-41
- *ROM table* on page 2-46
- *Authentication requirements* on page 2-49
- *Clocks and resets* on page 2-50
- *Connections to debug components and system interfaces* on page 2-51.

2.1 About the Debug Port

The debug port is the host tools interface to access the DAP-Lite. This interface controls any access ports provided within the DAP-Lite. The DAP-Lite supports a combined debug port which includes both 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 2-11,
- The SW-DP provides a two-pin (clock + data) interface to the DAP-Lite. For more information, see *SW-DP* on page 2-13.

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 2-3.

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.

2.2 SWJ-DP

The SWJ-DP is a combined JTAG-DP and SW-DP that enables you to connect either a *Serial Wire Debug* (SWD) or JTAG probe 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 use an autodetect mechanism that switches between JTAG-DP and SW-DP depending on which probe is connected. A special sequence on the **SWDIOTMS** pin is used to switch between JTAG-DP and SW-DP. When the switching sequence has been transmitted to the SWJ-DP, it behaves as a dedicated JTAG-DP or SW-DP depending upon which sequence had been performed.

Note

For more information about the programming capabilities and features of the SWJ-DP, see *JTAG-DP* on page 2-11 and *SW-DP* on page 2-13.

Figure 2-1 on page 2-4 shows the external connections to the SWJ-DP.

The SWJ-DP is described in more detail in:

- *Structure of the SWJ-DP* on page 2-4
- *Operation of the SWJ-DP* on page 2-4
- *JTAG and SWD interface* on page 2-5
- *Clock, reset and power domain support* on page 2-6
- *SWD and JTAG selection mechanism* on page 2-7.

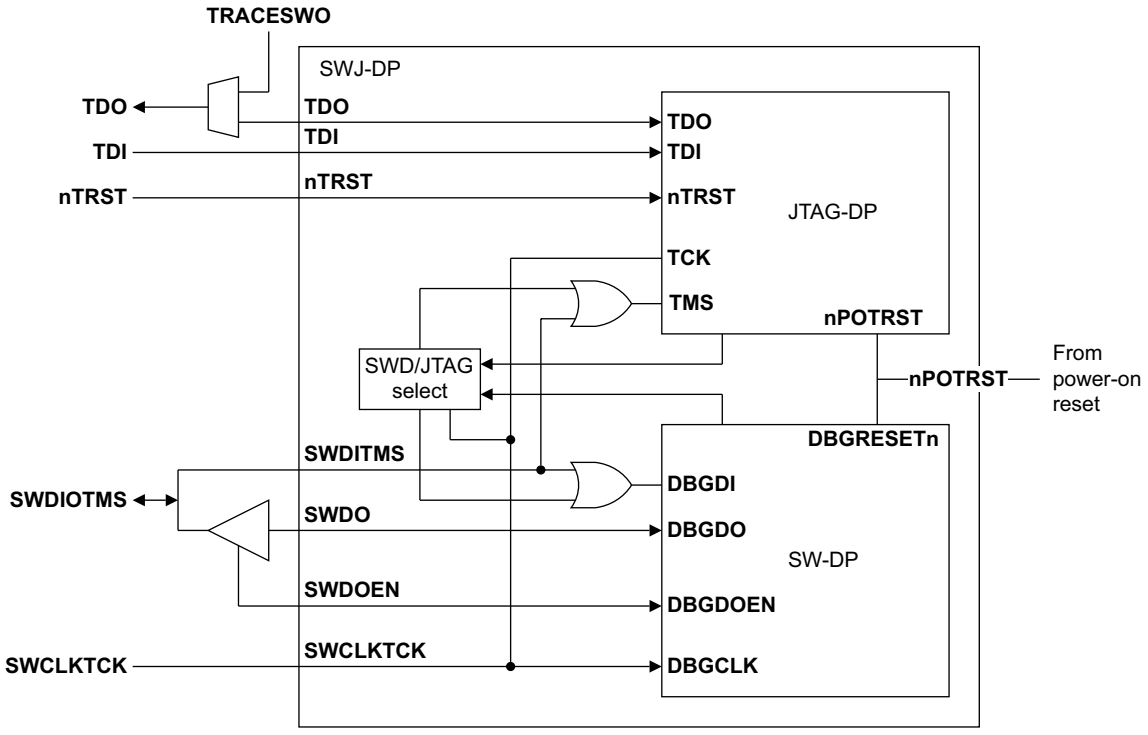


Figure 2-1 SWJ-DP external connections

2.2.1 Structure of the SWJ-DP

The SWJ-DP consists of a wrapper around the JTAG-DP and SW-DP. It selects JTAG or SWD as the connection mechanism and enables either JTAG-DP or SW-DP as the interface to the DAP.

2.2.2 Operation of the SWJ-DP

SWJ-DP enables you to design an *Application Specific Integrated Circuit* (ASIC) that you can use 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 you must use a JTAG debug interface. These enable:

- inclusion in an existing scan chain, usually on-chip TAPs used for test or other purposes.
- the device to be cascaded with legacy devices that use JTAG for debug

- use of existing debug hardware with the corresponding test TAPs, for example in *Automatic Test Equipment* (ATE).

You can connect an ASIC fitted with SWJ-DP support to legacy JTAG equipment without making any changes. If an SWD tool is available, only two pins are required, instead of the usual four used for JTAG. You can therefore use the other two pins for something else.

You can only use these two pins if there is no conflict with their use in JTAG mode. 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. You cannot use the other function while the ASIC is 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 can 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 supplied by external tools.

2.2.3 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, **SWCLK**, which can be input or output from the device.

The block level interface has two pins for data plus an output enable that 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 2-3 on page 2-8. In particular, **TMS** must be a bidirectional pin to support the bidirectional **SWDIO** pin in SWD mode. When SWD mode is used, the **TDO** pin is expected to be re-used for *Serial Wire Output* (SWO). You can use the **TDI** pin as an alternative input function.

Note

If you require SWO functionality in JTAG mode, you must have as a dedicated pin.

2.2.4 Clock, reset and power domain support

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 help provide separate power domains, it is possible to partition the RTL of SWJ-DP to enable an always on domain and debug domain as described in the *ARM Debug Interface v5 Architecture Specification*. Figure 2-2 shows the RTL structure to support power domain structure.

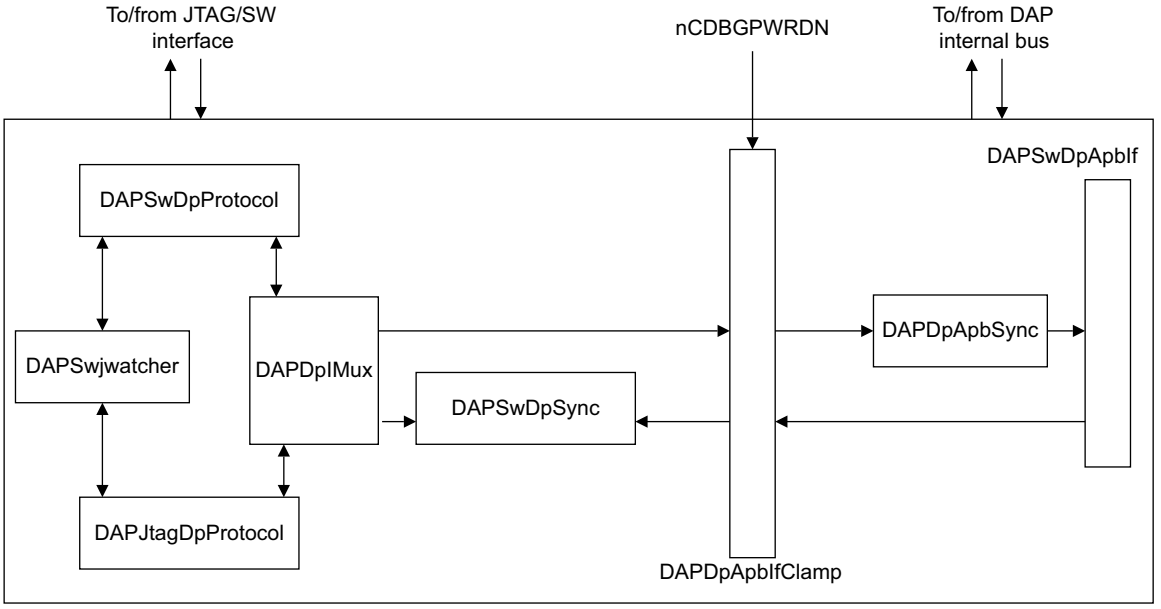


Figure 2-2 SWJ-DP signal clamping

2.2.5 SWD and JTAG selection mechanism

SWJ-DP enables either an 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 selection sequence on the **SWDIOTMS** pin:

- A 16-bit sequence is used to switch from JTAG to SWD operation
- Another 16-bit sequence is used to switch from SWD to JTAG.

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

Another status output, **JTAGTOP**, indicates the state of the JTAG-DP TAP controller. These states are:

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

This signal can be used 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 2-3 on page 2-8 is a simplified state diagram that shows how the watcher transitions between sleeping, detecting, and selection states.

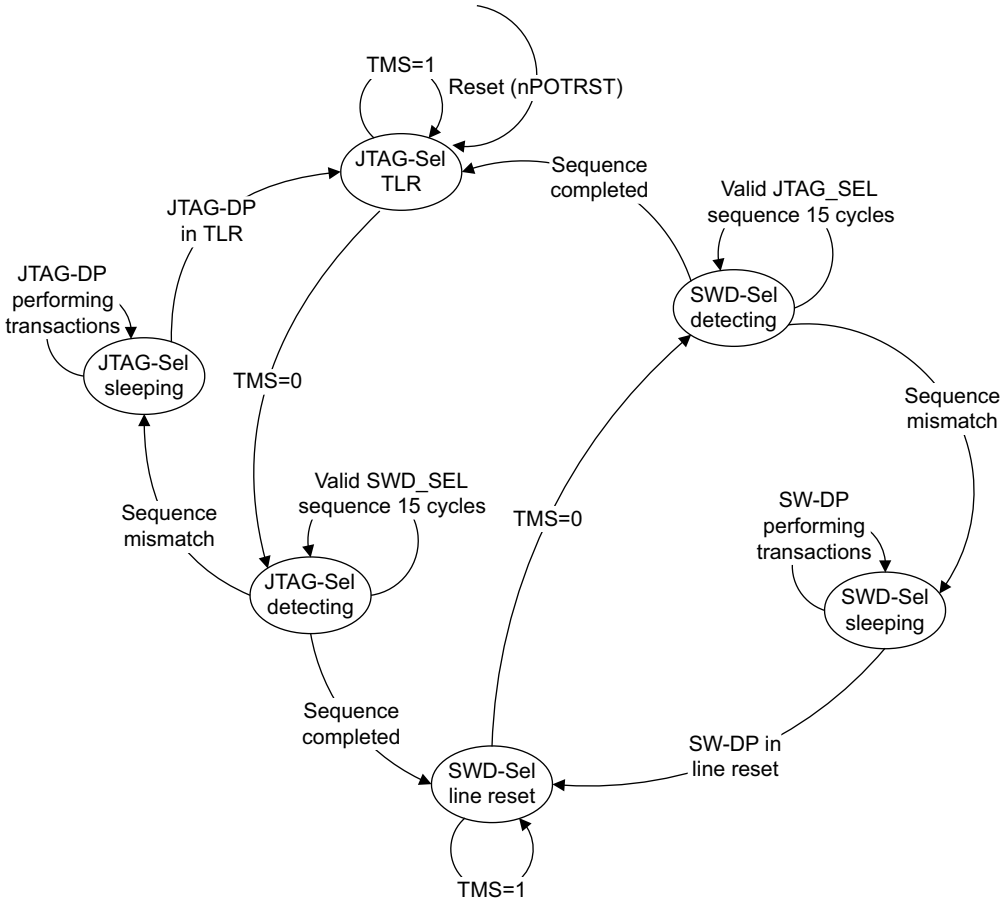


Figure 2-3 SWD and JTAG select state diagram

SWJ-DP switching sequences

The SWJ-DP switching sequences are described in:

- *JTAG to SWD switching*
- *SWD to JTAG switching* on page 2-9.

JTAG to SWD switching

To switch SWJ-DP from JTAG to SWD operation:

1. Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS**=1. This ensures that both SWD and JTAG are in their reset states

2. Send the 16-bit JTAG-to-SWD select sequence on **SWDIOTMS**
3. 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.
4. 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 0b0111100111100111, MSB first. This can be represented as 16'h79E7 if transmitted MSB first or 16'hE79E if 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:

1. Send more than 50 **SWCLKTCK** cycles with **SWDIOTMS**=1. This ensures that both SWD and JTAG are in their reset states.
2. Send the 16-bit SWD-to-JTAG select sequence on **SWDIOTMS**.
3. Send at least five **SWCLKTCK** cycles with **SWDIOTMS**=1. This ensures that if SWJ-DP was already in JTAG mode before sending the select sequence, it goes into the TLR state.
4. 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 0b0011110011100111, MSB first. This can be represented as 16'h3CE7 if transmitted MSB first or 16'hE73C if 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**.

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, that the system remains in this mode throughout the debug session. Switching between modes must not be attempted while any component of the DAP is active.

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

2.3 JTAG-DP

The JTAG-DP supplied with the DAP-Lite is an implementation of the JTAG-DP specified in the *ARM Debug Interface v5 Architecture Specification*, which also contains a detailed explanation of its programmer's model, capabilities and features.

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:

- *Overview*
- *Implementation specific details*.

2.3.1 Overview

With the JTAG-DP, IEEE 1149.1 scan chains are used to read or write register information. A pair of scan chain registers is used to access the main control and access registers within the Debug Port:

- DPACC used for *Debug Port* (DP) accesses.
- APACC used for *Access Port* (AP) accesses. An APACC access might access a register of a debug component of the system to which the interface is connected.

The scan chain model implemented by a JTAG-DP has the concepts of capturing the current value of APACC or DPACC, and of updating APACC or DPACC with a new value. An update might cause a read or write access to a DAP-Lite register that might then cause a read or write access to a debug register of a connected debug component. The operations available on JTAG-DP are described in the *ARM Debug Interface v5 Architecture Specification*. The implemented registers present within the supplied JTAG-DP are described in *Implementation specific details*.

2.3.2 Implementation specific details

The implementation specific details are described in the following:

- *Physical interface* on page 2-12
- *Programmer's model* on page 2-12.

Physical interface

The physical interface for JTAG-DP and the relationship to the signal references in the *ARM Debug Interface v5 Architecture Specification* is given in Table 2-1. The interface does not include a return clock signal. **RTCK** and the **nTRST** signals are optional because this only relates to resetting the DBGTAP state machine which can be performed by transmitting 5 **TCK** pulses with **TMS** HIGH.

Table 2-1 JTAG-DP physical interface

Implementation signal name (JTAG-DP)	ADIv5 signal name (JTAG-DP)	Direction	JTAG-DP signal description
TDI	**DBGTDI**	Input	Debug Data In
TDO	**DBGTDO**	Output	Debug Data Out
SWCLKTCK	**TCK**	Input	Debug Clock
SWDITMS	**DBGTMS**	Input	Debug Mode Select
nTRST	**DBGTRSTn**	Input	Debug TAP Reset

Programmer's model

Table 2-2 lists all implemented registers accessible by JTAG-DP. All other IR instructions are implemented as BYPASS and an external TAP controller must be implemented in accordance with the *ARM Debug Interface v5 Architecture Specification* if more IR registers are required, for example JTAG TAP boundary scan.

Table 2-2 JTAG-DP registers

IR instruction value	JTAG-DP register	DR scan width	Description
b1000	ABORT	35	JTAG-DP Abort Register (ABORT)
b1010	DPACC	35	JTAG DP/AP Access Registers (DPACC/APACC)
b1011	APACC	35	
b1110	IDCODE	32	JTAG Device ID Code Register (IDCODE)
b1111	BYPASS	1	JTAG Bypass Register (BYPASS)

For more information about these registers, their features, and how to access them, see the *ARM Debug Interface v5 Architecture Specification*. Implementation specific detail is described in *Common debug port features and registers* on page 2-19.

2.4 SW-DP

This section briefly describes the *Serial Wire Debug Port* (SW-DP) interface. This implementation is taken from the *ARM Debug Interface v5 Architecture Specification* and operates with a synchronous serial interface. This uses a single bidirectional data signal, and a clock signal.

2.4.1 Overview

The SW-DP provides a low pin count bi-directional serial connection to the DAP with a reference clock signal for synchronous operation.

Communications with the SW-DP use a three-phase protocol:

- A host-to-target packet request.
- A target-to-host acknowledge response.
- A data transfer phase, if required. This can be target-to-host or host-to-target, depending on the request made in the first phase.

A packet request from a debugger indicates whether the required access is to a DP register (DPACC) or to an AP register (APACC), and includes a two-bit register address. The protocol is described in detail in the *ARM Debug Interface v5 Architecture Specification*.

2.4.2 Implementation specific details

This section contains the following:

- *Clocking*
- *Overview of debug interface* on page 2-14.

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.

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 100kΩ 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 if 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.

2.4.3 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 2-4 on page 2-16 shows the effect of signalling ACK = WAIT on the length of the packet.

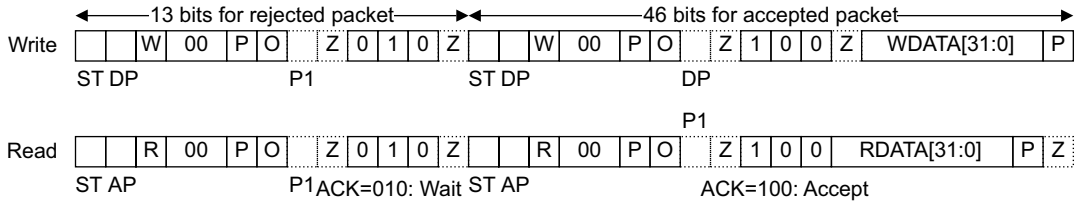


Figure 2-4 SW-DP acknowledgement timing

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 2-3 shows the terms used in Figure 2-5 on page 2-17 to Figure 2-7 on page 2-18.

Table 2-3 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 this timeframe.
WD[1]	Second write packet data.
RD[0]	First read packet data.
RD[1]	Second read packet data.

Figure 2-5 on page 2-17 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.

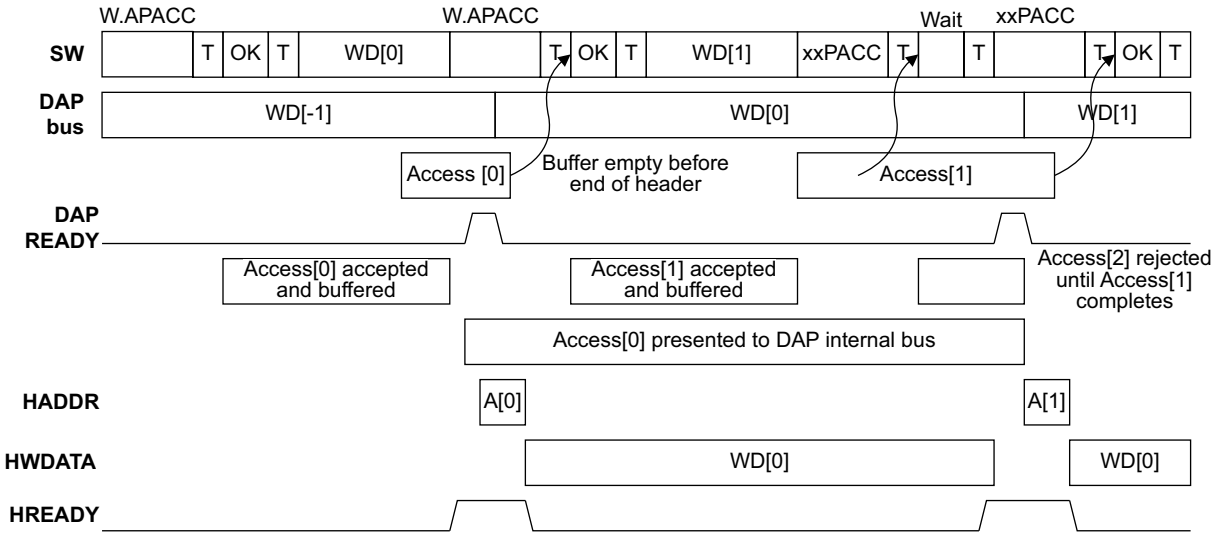


Figure 2-5 SW-DP to DAP bus timing for write

Figure 2-6 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.

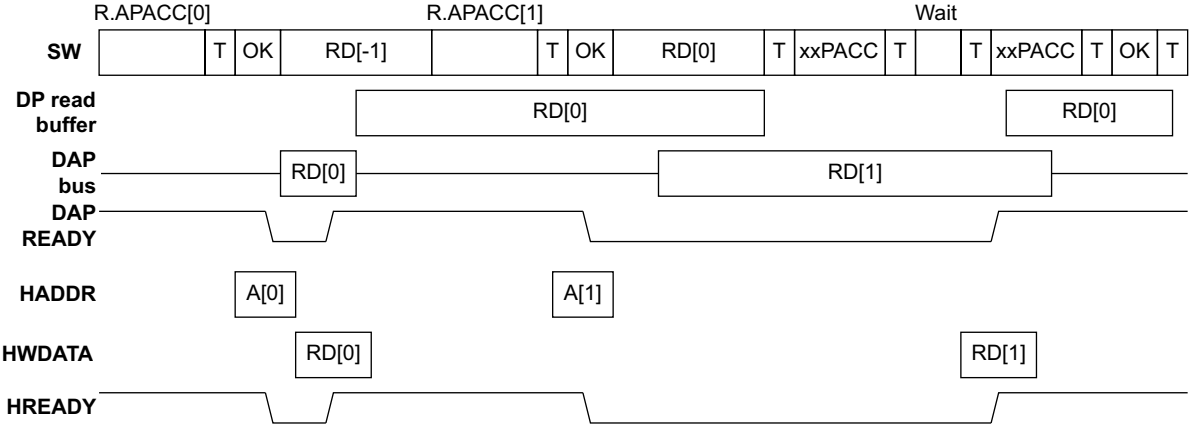


Figure 2-6 SW-DP to DAP bus timing for read

Figure 2-7 on page 2-18 shows a sequence of transfers separated by IDLE periods. It shows that the wire is always handed back to the host after any transfer.

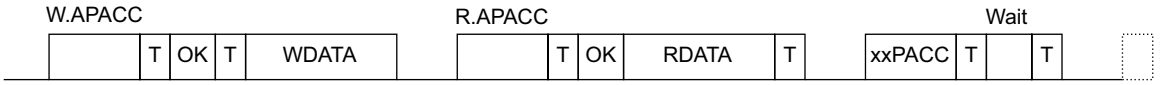


Figure 2-7 SW-DP idle timing

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.

2.5 Common debug port features and registers

This section describes specific details of features and registers that are present within this implementation of SW-DP and JTAG-DP as part of the SWJ-DP. For all the features and registers present within SW-DP and JTAG-DP, see the *ARM Debug Interface v5 Architecture Specification*. This section contains the following implementation specific details:

- *Features overview*
- *Example pushed operations*
- *Debug Port registers overview* on page 2-21
- *Implementation specific registers* on page 2-21

2.5.1 Features overview

Both the SW-DP and JTAG-DP views within the SWJ-DP contain the same features described in the *ARM Debug Interface v5 Architecture Specification*. Their features include:

- Sticky flags and debug port error responses as a result of either a read and write error response from the system or because of an overrun detection (STICKYORUN).
- Pushed compare and pushed verify to enable more optimized control from a debugger by performing a set of write transactions and enabling any comparison operation to be done within the debug port. See *Example pushed operations* for specific examples with the DAP-Lite.
- Transaction counter to recover to a point within a repeated operation (typically in combination with a pushed function and auto-incrementing in an access port).
- System and debug power and debug reset control. This is to enable an external debugger to connect to a potentially turned-off system and power up as much as required to get a basic level of debug access with minimal understanding of the system.

These features are described in more detail in the *ARM Debug Interface v5 Architecture Specification*.

2.5.2 Example pushed operations

These are two examples using this specific implementation of the ADIv5. All register and feature references are related to those described in their respective chapters and the *ARM Debug Interface v5 Architecture Specification*.

This section contains two examples:

- *Example use of pushed verify operation on an APB-AP*
- *Example use of pushed find operation on a APB-AP*.

Example use of pushed verify operation on an APB-AP

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

- Make sure that the APB-AP Control/Status Word (CSW) is set up to increment the TAR after each access. See *APB-AP Control/Status Word Register, CSW, 0x00* on page 2-34.
- Write to the TAR to indicate the start address of the Debug Register region that is to be verified, see *APB-AP Transfer Address Register, TAR, 0x04* on page 2-36.
- 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 *APB-AP Control/Status Word Register, CSW, 0x00* on page 2-34.

  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 APB-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 APB-AP *Control/Status Word* (CSW) is set up to increment the TAR after each access. See *APB-AP Control/Status Word Register, CSW, 0x00* on page 2-34.
- Write to the TAR to indicate the start address of the Debug Register region that is to be searched. See *APB-AP Transfer Address Register, TAR, 0x04* on page 2-36.
- 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.

2.5.3 Debug Port registers overview

Table 2-4 summarizes the DP registers, and lists which registers are implemented on a JTAG-DP and which are implemented on a SW-DP.

Table 2-4 Summary of Debug Port registers

Name	Description	JTAG-DP	SW-DP	For description see section
ABORT	AP Abort Register	Yes	Yes	-
IDCODE	ID Code Register	Yes	Yes	*Identification Code Register, IDCODE*
CTRL/STAT	DP Control/Status Register	Yes	Yes	*Control/Status Register, CTRL/STAT* on page 2-23
SELECT	Select Register	Yes	Yes	*AP Select Register, SELECT* on page 2-25
RDBUFF	Read Buffer	Yes	Yes	*Read Buffer, RDBUFF* on page 2-27
WCR	Wire Control Register	No	Yes	*Wire Control Register, WCR (SW-DP only)* on page 2-28
RESEND	Read Resend Register	No	Yes	*Read Resend Register, RESEND (SW-DP only)* on page 2-30
ROUTESEL	Reserved	No	Optional	See footnote[a]

a. The specification of the SW-DP provides for an optional ROUTESEL register. However this is not implemented in this release of the SW-DP.

2.5.4 Implementation specific registers

This section describes the implementation specific registers.

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 is accessed using its own scan chain.

SW-DP 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. The Identification Code Register is:

- a read-only register
- always accessible.

Figure 2-8 shows the Identification Code Register bit assignments.

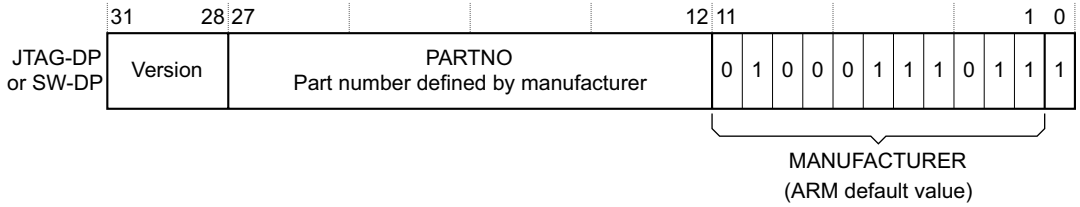


Figure 2-8 Identification Code Register bit assignments

Table 2-5 shows the Identification Code Register bit assignments.

Table 2-5 Identification Code Register bit assignments

Bits	Function	Description
[31:28]	Version	Version code: **JTAG-DP** `0x4` **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** `0xBA01`
[11:1]	MANUFACTURER	JEDEC Manufacturer ID, an 11-bit JEDEC code that identifies the designer of the device. See *JEDEC Manufacturer ID* on page 2-23. The ARM value for this field, shown in Figure 2-8, is `0x23B`. This value must not be changed.
[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 2-6. JDEC codes are assigned by the JEDEC Solid State Technology Association, see JEP106M, Standard Manufacture's Identification Code.

Table 2-6 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.

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 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 2-25.

The Control/Status Register 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. Figure 2-9 shows the Control/Status Register bit assignments.

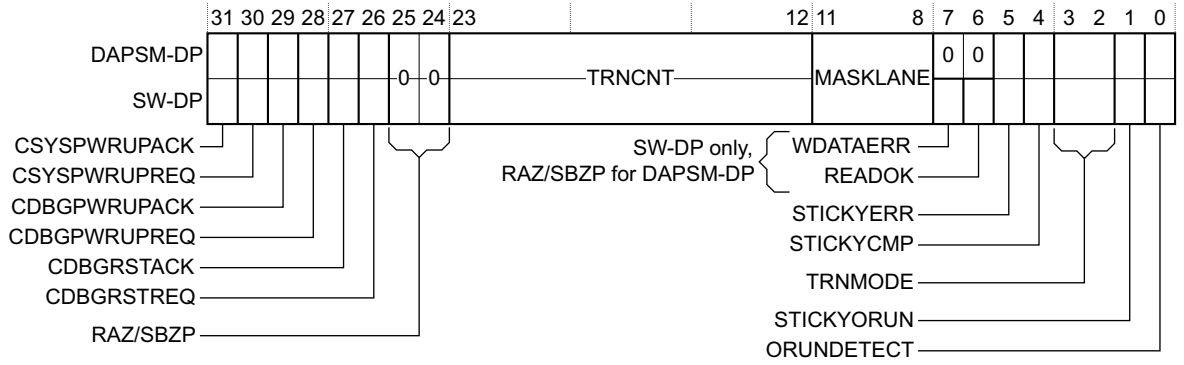


Figure 2-9 Control/Status Register bit assignments

Table 2-7 shows the Control/Status Register bit assignments.

Table 2-7 Control/Status Register bit assignments

Bits	Access	Function	Description
[31]	RO	CSYSPWRUPACK	System power-up acknowledge.
[30]	R/W	CSYSPWRUPREQ	System power-up request. After a reset this bit is LOW (0).
[29]	RO	CDBGPWRUPACK	Debug power-up acknowledge.
[28]	R/W	CDBGPWRUPREQ	Debug power-up request. After a reset this bit is LOW (0).
[27]	RO	CDBGRSTACK	Debug reset acknowledge.
[26]	R/W	CDBGRSTREQ	Debug reset request. After a reset this bit is LOW (0).
[25:24]	-	-	Reserved, RAZ/SBZP
[21:12]	R/W	TRNCNT	Transaction counter. After a reset the value of this field is Unpredictable.
[11:8]	R/W	MASKLANE	Indicates the bytes to be masked in pushed compare and pushed verify operations. After a reset the value of this field is Unpredictable.
[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. After a power-on reset this bit is LOW (0).
[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).

Table 2-7 Control/Status Register bit assignments (continued)

Bits	Access	Function	Description
[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. After a power-on reset this bit is LOW (0).
[4]	RO[a]	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. After a power-on reset this bit is LOW (0).
[3:2]	R/W	TRNMODE	This field sets the transfer mode for access port operations. After a power-on reset the value of this field is Unpredictable.
[1]	RO[a]	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. After a power-on reset this bit is LOW (0).
[0]	R/W	ORUNDETECT	This bit is set to b1 to enable overrun detection. After a reset this bit is LOW (0).

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.

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 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 2-10 shows the AP Select Register bit assignments.

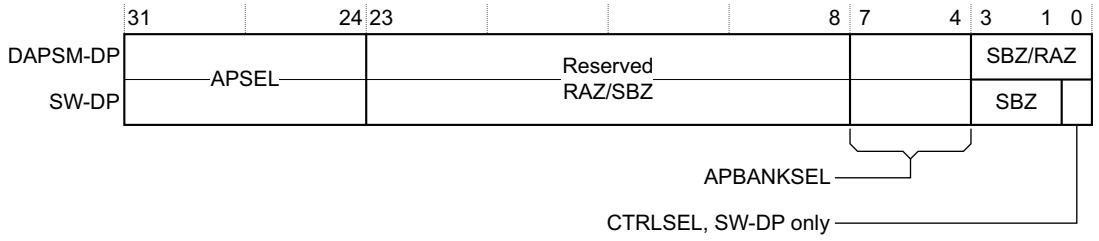


Figure 2-10 AP Select Register bit assignments

Table 2-8 shows the AP Select Register bit assignments.

Table 2-8 AP Select Register bit assignments

Bits	Function	Description
[31:24]	APSEL	Selects the current access port. `0x00` - APB-AP 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. 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, therefore you cannot read the field value.
b. 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.

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 2-25.

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 2-11 shows the Wire Control Register bit assignments.

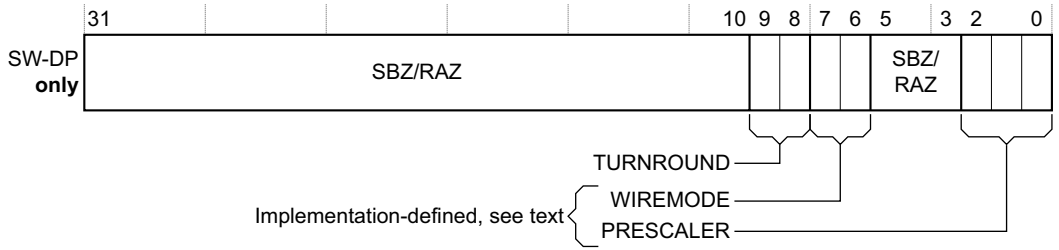


Figure 2-11 Wire Control Register bit assignments

Table 2-9 shows the Wire Control Register bit assignments.

Table 2-9 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]* on page 2-29. After a reset this field is b00.

Table 2-9 Wire Control Register bit assignments (continued)

Bits	Function	Description
[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 2-10 lists the allowed values of this field, and their meanings.

Table 2-10 Turnaround tristate period field bit definitions

TURNROUND[a]	Turnaround tristate 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 2-11 lists the allowed values of the field, and their meanings.

Table 2-11 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. It enables 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.

2.6 Access ports

An access port provides the interface between the debug port interface and one or more debug components present within the system. The DAP-Lite contains a *Memory Access Port* (MEM-AP), which is designed for connection to memory bus system with address and data controls.

All access ports follow a base standard for identification, and debuggers must be able to recognize and ignore access ports that they do not support. The connection method does not depend on the type of debug port used and the type of access port being accessed.

For more information on access ports and recommend debugger interaction with access ports, see the *ARM Debug Interface v5 Architecture Specification*.

2.6.1 Overview

The access port supplied within the DAP-Lite is APB-AP to enable direct connection to the dedicated Debug Bus.

2.7 APB-AP

The APB-AP implements the MEM-AP architecture to directly connect to an APB based system. The intention is that this bus is dedicated to CoreSight and other debug components. Figure 2-12 shows the functional blocks of the APB-AP.

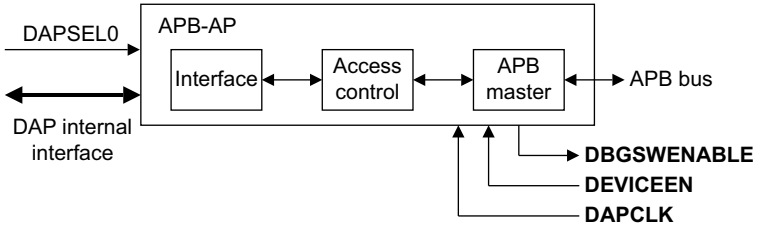


Figure 2-12 APB-AP functional blocks

As part of the MEM-AP description, the APB-AP has a number of implementation specific features; these are covered in:

- *External interfaces*
- *Implementation features* on page 2-33
- *Programmer's model overview* on page 2-33
- *DAP transfers* on page 2-39.

For information on all the registers and features in a MEM-AP, see the *ARM Debug Interface v5 Architecture Specification*.

2.7.1 External interfaces

The primary interface on APB-AP is an APB AMBAv3 compliant interface supporting:

- extended slave transfers
- transfer response errors.

Table 2-12 shows the other APB-AP ports.

Table 2-12 APB-AP other ports

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

2.7.2 Implementation features

The APB-AP provides the following specific MEM-AP features:

- Auto-incrementing of the Transfer Address Register with address wrapping on 1k byte boundaries.
- Slave memory port disabling: a slave interface is provided through the APB-MUX to enable another APB master to connect to the same memory map as the APB-AP

The AHB-AP does not support the following MEM-AP features:

- Big-endian. All accesses performed as expected to be to a little-endian memory structure.
- Sub-word transfers. Only word transfers are supported

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.

2.7.3 Programmer's model overview

Table 2-13 shows the APB-AP registers.

Table 2-13 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
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

Table 2-13 APB-AP registers (continued)

Offset	Type	Width	Reset value	Name
0x20-0xF4	-	-	-	Reserved SBZ
0x80000000	RO	32	Implementation-defined.	Debug ROM Address, ROM
0xFC	RO	32	0x14770002	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 2-36
- *APB-AP Data Read/Write Register, DRW, 0x0C* on page 2-37
- *APB-AP Banked Data Registers, BD0-BD3, 0x10-0x1C* on page 2-37
- *Debug APB ROM Address, ROM, 0xF8* on page 2-38
- *APB-AP Identification Register* on page 2-38.

2.7.4 APB-AP Control/Status Word Register, CSW, 0x00

The APB-AP Control/Status Word Register is used to configure and control transfers through the APB interface. Figure 2-13 shows the bit assignments.

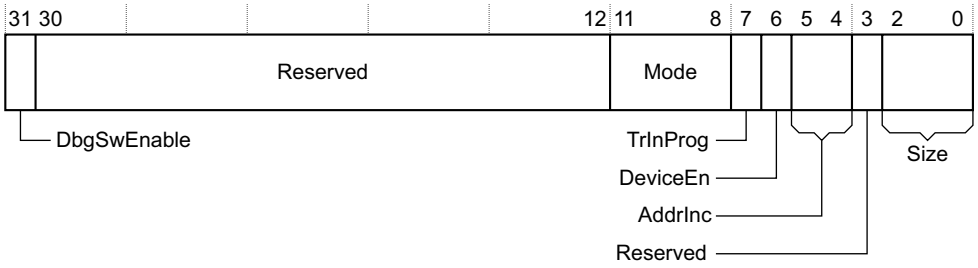


Figure 2-13 APB-AP Control/Status Word Register bit assignments

Table 2-14 shows the bit assignments.

Table 2-14 APB Control/Status Word Register bit assignments

Bits	Type	Name	Function
[31]	R/W	DbgSwEnable	Software access enable. Drives **DBGSWENABLE** 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**.

Table 2-14 APB 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. 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.
[3]	-	-	Reserved SBZ.
[2:0]	RO	Size	Size of the access to perform. Fixed at b010 = 32 bits. Reset value = b010.

2.7.5 APB-AP Transfer Address Register, TAR, 0x04

The Transfer Address Register holds the address of the current transfer. Figure 2-14 shows the bit assignments.

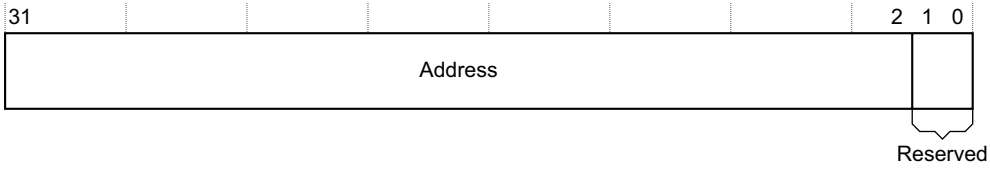


Figure 2-14 APB-AP Transfer Address Register bit assignments

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 2-15 shows the bit assignments.

Table 2-15 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.

2.7.6 APB-AP Data Read/Write Register, DRW, 0x0C

Table 2-16 shows the bit assignments of the APB-AP Data Read/Write Register.

Table 2-16 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.

2.7.7 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. For example, BD0 reads/write from TAR, and BD1 from TAR+4.

Table 2-17 shows the bit assignments.

Table 2-17 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

2.7.8 Debug APB ROM Address, ROM, 0xF8

A ROM table must be present in all CoreSight systems. See *ROM table programmer's model* on page 2-46 for more information. Figure 2-15 shows the bit assignments.

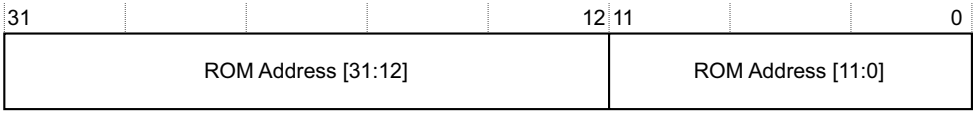


Figure 2-15 Debug APB ROM Address Register bit assignments

Table 2-18 shows the bit assignments.

Table 2-18 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.

2.7.9 APB-AP Identification Register

Figure 2-16 on page 2-39 shows the APB-AP Identification Register bit assignments.

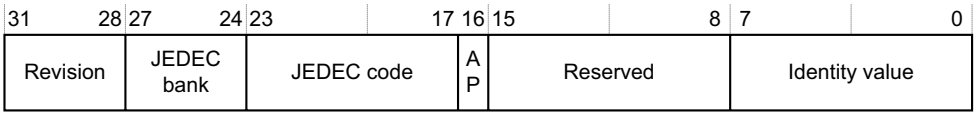


Figure 2-16 APB-AP Identification Register bit assignments

Table 2-19 shows the APB-AP Identification Register bit assignments.

Table 2-19 APB-AP Identification Register bit assignments

Bits	Type	Name
[31:28]	RO	Revision. Reset value is `0x1` for APB-AP.
[27:24]	RO	JEDEC bank. `0x4` indicates ARM Limited.
[23:17]	RO	JEDEC code. `0x3B` indicates ARM Limited.
[16]	RO	Memory AP. `0x1` indicates a standard register map is used.
[15:8]	-	Reserved SBZ.
[7:0]	RO	Identity value. Reset value is `0x02` for APB-AP.

2.7.10 DAP transfers

This section describes DAP transfers.

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.

APB-AP error response generation

APB-AP error response generation is described in:

- *System initiated error response* on page 2-40
- *AP-initiated error response* on page 2-40
- *Differentiation between System-initiated and AP-initiated error responses* on page 2-40

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.

2.8 APB-Mux

The *APB Multiplexor* (APB-Mux) for the DAP-Lite enables external tools and system access to the Debug APB. The APB-Mux encapsulates the multiple interfaces into a single deliverable component, enabling multi-master access to the Debug APB.

Figure 2-17 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.

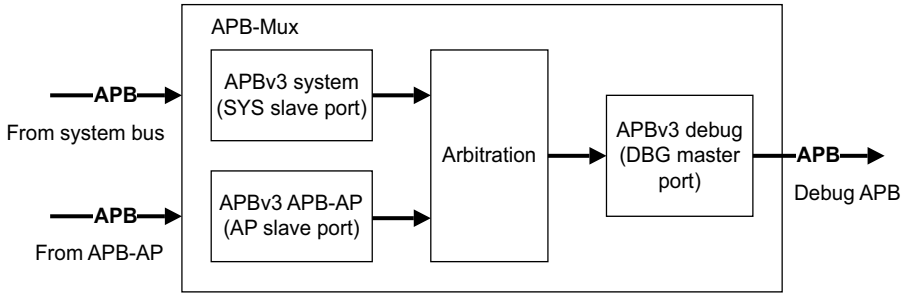


Figure 2-17 APB-Mux block diagram

Figure 2-18 shows the APB-Mux integrated into the DAP-Lite. The 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 2-17 to support distinct debug and system power domains.

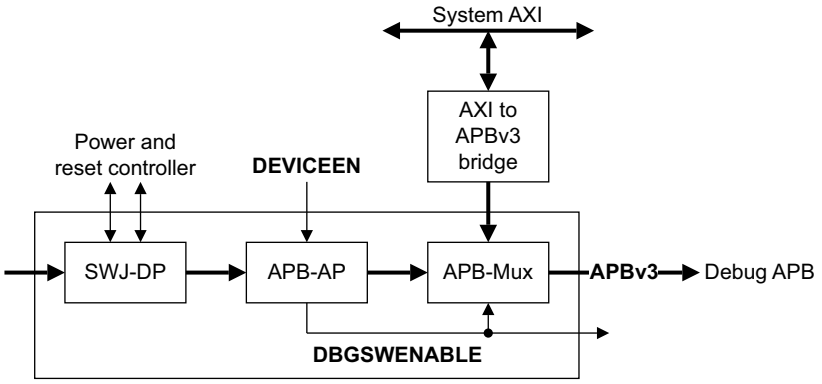


Figure 2-18 APB-Mux integrated into the DAP-Lite

The APB-Mux is described in:

- *APB-Mux port definitions* on page 2-42

- *APB-Mux miscellaneous signals*
- *APB-Mux arbitration and connectivity*
- *APB-Mux software access enable* on page 2-43
- *APB-Mux clocks, power, and resets* on page 2-43.

2.8.1 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*.

2.8.2 APB-Mux miscellaneous signals

Table 2-20 shows the APB-Mux miscellaneous signals.

Table 2-20 APB-Mux miscellaneous signals

Name	Type	Description
nCDBGPWRDN	Input	Indicates that the debug infrastructure is powered down. Any system accesses to the debug APB returns 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.
DBGSWENABLE	Input	Enables software access to the Debug APB.

2.8.3 APB-Mux arbitration and connectivity

This section describes APB-Mux arbitration and connectivity:

- *APB-Mux arbitration*
- *APB-Mux connectivity* on page 2-43.

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 Debug APB at any one time.

The AP Slave port always has priority over the System Slave port. If two transfers are initiated at the same time then the transfer from the 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 AP Slave port is not requesting an access, that is, **PSELAP** is LOW. It is therefore possible for the AP Slave port to maintain back-to-back transfers on the Debug APB without enabling 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 or default response generation. This must be implemented by an external address decoder incorporating a default slave.

APB-Mux connectivity

The connection rules are:

- AP Slave port connects to the APB-AP
- System Slave port connects to the system bus.

2.8.4 APB-Mux software access enable

The APB-Mux receives **DBGSWENABLE** from the APB-AP to enable software access from the System Slave Port to the Debug APB.

DBGSWENABLE 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.

DBGSWENABLE HIGH

System access to the Debug APB is permitted.

2.8.5 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 2-19 on page 2-44. 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.

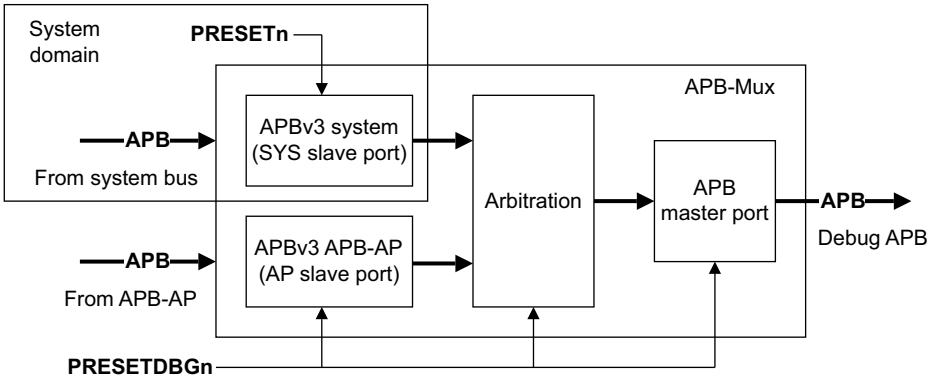


Figure 2-19 APB-Mux domains

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 crosses no 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 then 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 then 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, then this must not invalidate the existing transfer already in progress from the 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 within the Debug domain, and the system slave port in the SoC power domain, then clamping logic must be instantiated on the outputs of signals crossing each power down domain.

Figure 2-20 shows the RTL structure to support power domain separation.

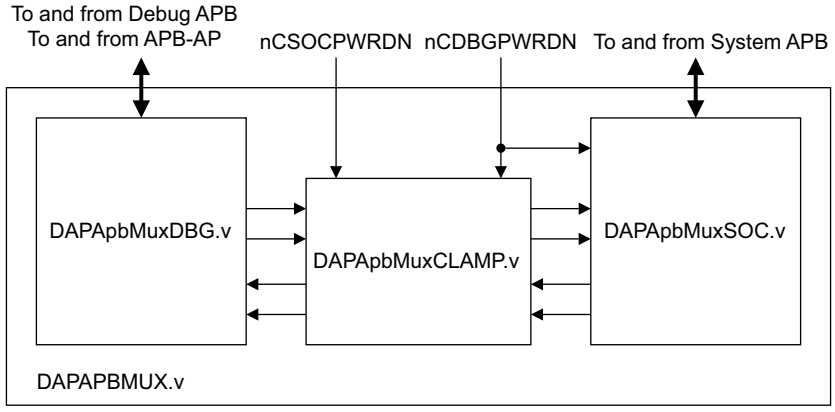


Figure 2-20 APB-Mux power domain separation

2.9 ROM table

The DAP-Lite provides a configurable internal *Read Only Memory* (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. This ROM table will typically be configured to list all of the CoreSight components in a system.

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.

The ROM table is described in:

- *ROM table programmer's model*
- *ROM table entries* on page 2-48.

2.9.1 ROM table programmer's model

Table 2-21 shows the ROM table registers. The values of the table entries depend on the debug subsystem that is implemented.

Table 2-21 ROM table registers

Offset	Type	Bits	Name	Function
`0xFDC`	-	[7:0]	Peripheral ID7	Unused -Reserved SBZ for future use. Read as `0x00`.
`0xFD8`	-	[7:0]	Peripheral ID6	Unused -Reserved SBZ for future use. Read as `0x00`.
`0xFD4`	-	[7:0]	Peripheral ID5	Unused -Reserved SBZ for future use. Read as `0x00`.
`0xFD0`	RO	[7:4]	Peripheral ID4	4KB count, set to `0x0`.
		[3:0]		JEP106 continuation code, implementation configurable.
`0xFEC`	RO	[7:4]	Peripheral ID3	RevAnd, at top level, implementation configurable.
		[3:0]		Customer Modified, implementation configurable.

Table 2-21 ROM table registers (continued)

Offset	Type	Bits	Name	Function
0xFE8	RO	[7:4]	Peripheral ID2	Revision number of Peripheral, implementation configurable.
		[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 configurable.
0xFE4	RO	[7:4]	Peripheral ID1	JEP106 Identity Code [3:0], implementation configurable.
		[3:0]		PartNumber1, implementation configurable.
0xFE0	RO	[7:0]	Peripheral ID0	PartNumber0, implementation configurable.
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 must be read as zero. Locations 0xF00-0xFCC are reserved and must be read as zero.

Note

The Peripheral ID values must be a product specific identifier for the entire system.

2.9.2 ROM table entries

Table 2-22 shows the ROM table entries bit assignments for each entry in the `0x000-0xEFC` region.

Table 2-22 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, read as zero.
[1]	Format	1 = 32-bit format. In the DAP-Lite 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.

2.10 Authentication requirements

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, then 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.
- 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*.

2.11 Clocks and resets

DAPCLK must be driven by a constant clock. It must not be stopped or altered while the DAP-Lite 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 within the **DAPCLK** domain. **DAPRESETn** enables initialization of the DAP-Lite without affecting the normal operation of the SoC in which the DAP-Lite is integrated, and must 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 SWJ-DP. This resets all the registers in the Debug clock domain, that is, Debug APB and **DAPCLK** domains.

Note

For this release of the DAP-Lite, **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**.

2.12 Connections to debug components and system interfaces

Figure 2-21 shows how debug components and a system bus can be accessed through the DAP-Lite. System access is only possible from JTAG through the processor. Debug components can be accessed by either software, or the DAP-Lite, or both, depending on the configuration of the AXI bus matrix.

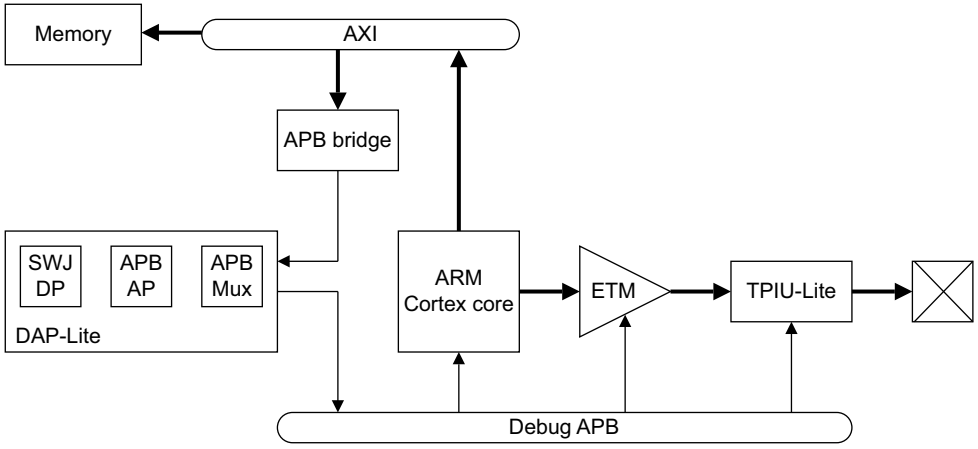


Figure 2-21 Debug trace with a single core

Chapter 3
Programmer's Model

This chapter describes the CoreSight DAP-Lite programmer's model. It contains the following section:

- *About the programmer's model* on page 3-2.

3.1 About the programmer's model

There is no single programmer's model for the DAP-Lite. It consists of a number of blocks, and their programmer's models are described in the following sections:

- *JTAG-DP* on page 2-11
- *APB-AP* on page 2-32
- *ROM table* on page 2-46.

The following apply to all DAP-Lite registers:

- All registers 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.

Appendix A
DAP-Lite Ports

This appendix describes the port and interface signals in the CoreSight DAP-Lite. It contains the following section:

- *CoreSight DAP signals* on page A-2.

A.1 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
CSYSPWRUPACK	Input	System Power Domain power-up acknowledge SWJ-DP	None
CSYSPWRUPREQ	Output	System Power Domain power-up request SWJ-DP	None
DBGSWENABLE	Output	Enable software access to debug registers when HIGH. Not required where access to registers is through the APB-Mux. APB-AP	PCLKDBG
DEVICEEN	Input	Enable access to Debug APB from the DAP APB-AP	None
JTAGNSW	Output	HIGH if JTAG selected, LOW if SWD selected SWJ-DP	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 • SWJ-DP.	SWCLKTCK
nCDBGPWRDN	Input	Debug infrastructure power-down control SWJ-DP	None

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
nCSOCPWRDN	Input	External system, SOC domain, power-down control SWJ-DP	None
nPOTRST	Input	Power-on reset SWJ-DP	SWCLKTCK
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
PRDATADBG[31:0]	Input	Debug APB read data bus	PCLKDBG
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

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
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
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

Table A-1 CoreSight DAP signals (continued)

Name	Type	Description	Clock domain
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

Appendix B
Revisions

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

Table B-1 Differences between issue C and issue D

Change	Location
Update block revision information.	Table 1-1 on page 1-6
Update Identification Code Register bit assignments.	Table 2-5 on page 2-22

Glossary

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

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.

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 describes 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, APB, and AXI conform 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.

AMBA *See* Advanced Microcontroller Bus Architecture.

APB *See* Advanced Peripheral Bus.

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.

Byte An 8-bit data item.

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.

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.

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.

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.

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

ETM *See* Embedded Trace Macrocell.

IEEE 754 standard *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985*. The standard that defines data types, correct operation, exception types and handling, and error bounds for floating-point systems. Most processors are built in compliance with the standard in either hardware or a combination of hardware and software.

Implementation-defined
Means that the behavior is not architecturally defined, but must be defined and documented by individual implementations.

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 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.

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

RealView ICE A system for debugging embedded processor cores using a JTAG interface.

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.

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.

TAP *See* Test access port.

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

TPA *See* Trace Port Analyzer.

TPIU *See* Trace Port Interface Unit.

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.

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.