ARM® Debug Interface v5
Architecture Specification
ADIv5.1 Supplement

Document number: DSA09-PRDC-008772 1.0
Date of Issue: 17 August, 2009
Author: ARM Limited

Proprietary notice

Contents

1 ABOUT THIS DOCUMENT

1.1 References

This document refers to the following documents.

Doc No	Author	Title
ARM DDI 0211	ARM	ARM1136JF-S™ and ARM1136J-S™ Technical Reference Manual*
ARM DDI 0314	ARM	CoreSight™ Components Technical Reference Manual*
ARM DDI 0406	ARM	ARM® Architecture Reference Manual ARM®v7-A and ARM®v7-R edition*
ARM IHI 0011	ARM	AMBA™ Specification (Rev 2.0) *
ARM IHI 0024	ARM	AMBA™ 3 APB Protocol Specification*
ARM IHI 0031	ARM	ARM® Debug Interface v5 Architecture Specification*
ARM IHI 0033	ARM	AMBA® 3 AHB-Lite Protocol Specification*
IEEE 1149.1-2001	IEEE	IEEE Standard Test Access Port and Boundary Scan Architecture

* These documents can be accessed through the ARM website at http://infocenter.arm.com/. For further information, contact ARM Limited.

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Data Link

The layer of an ADIv5 implementation that provides the functional and procedural means to transfer data between the external debugger and the Debug Port. ADIv5 defines two Data Link layers, one based on the IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture, widely referred to as JTAG, and one based on the ARM Serial Wire protocol interface.

DATA LINK DEFINED

Means that the behavior is not defined by the base architecture, but should be defined and documented by individual Data Link layers of the architecture.

IMPLEMENTATION DEFINED

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

RAZ/WI fields

Read-As-Zero, Writes Ignored.

In any implementation, the bit must read as 0 (or all 0s for a bit field), and writes to the field must be ignored. Software can rely on the field reading as zero, and on the write being ignored.

Read-As-One fields (RAO)

In any implementation, the bit must read as 1, or all 1s for a bit field.

Read-As-Zero fields (RAZ)

In any implementation, the bit must read as 0, or all 0s for a bit field.

Should-Be-One fields (SBO)

Software should write as 1, or all 1s for a bit field. Values other than 1 produce UNPREDICTABLE results.

Should-Be-Zero fields (SBZ)

Software should write as 0, or all 0s for a bit field. Values other than 0 produce UNPREDICTABLE results.

Should-Be-Zero-or-Preserved fields (SBZP)

Must be written as 0, or all 0s for a bit field, by software if the value is being written without having been previously read, or if the register has not been initialized. Where the register was previously read on the same processor, since the processor was last reset, the value in the field should be preserved by writing the value that was previously read.

Hardware must ignore writes to these fields.

If a value is written to the field that is neither 0 (or all 0s for a bit field), nor a value previously read for the same field on the same processor, the result is UNPREDICTABLE.

UNK

Is an abbreviation indicating that software must treat a field as containing an UNKNOWN value.

In any implementation, the bit must read as 0, or all 0s for a bit field. Software must not rely on the field reading as zero.

UNK/SBZP

UNKNOWN on reads, Should-Be-Zero-or-Preserved on writes.

In any implementation, the bit must read as zero, or all 0s for a bit field, and writes to the field must be ignored. Software must not rely on the bit reading as zero, or all 0s for a bit field, and must use an SBZP policy to write to the field.

UNKNOWN

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

UNKNOWN values must not be documented or promoted as having a defined value or effect.

UNPREDICTABLE

Means the behavior cannot be relied upon. UNPREDICTABLE behavior must not represent security holes.

UNPREDICTABLE behavior must not halt or hang the device, or any parts of the system. UNPREDICTABLE behopavior must not be documented or promoted as having a defined effect.

Note

In the *ARM® Debug Interface v5 Architecture Specification*, UNPREDICTABLE was also used to mean an UNKNOWN value.

Write-One-To-Clear bits (W1C)

Writing a 1 to the bit clears it to zero. Writing a 0 to the bit has no effect.

1.3 Introduction

This document is an update to the *ARM® Debug Interface v5 Architecture Specification*. It includes errata and new features for ADIv5.

Those new features represent a minor revision of the architecture specification, and hence the architecture version number is v5.1. The new features are backwards compatible with v5.0; the version documented by the *ARM® Debug Interface v5 Architecture Specification*. The terms *ARM Debug Interface v5* and *ADIv5* refer to the major revision of ADI, that is, to v5.0, v5.1 or any future minor revision of ADIv5.

Section 2, *ADIv5.0 Errata and Clarifications*, starting on page 8, describes each of the errata and clarifications in turn.

Section 3, *ADIv5.1 Updates*, starting on page 13, describes each of the new features in turn.

Sections 4, 5, 6, and 7 give more detailed information for some of the errata, clarifications and updates.

The final sections are new appendices to the *ARM® Debug Interface v5 Architecture Specification*. The information provided is not architectural and hence not part of ADIv5, but is nevertheless useful reference material for implementations and users of ADIv5.

Section 8, *Appendix: Serial Wire Protocol* starting on page 42 provides more background information on the Serial Wire protocol that is required for compatibility with Serial Wire protocol implementations. The Serial Wire protocol was previously described in the *CoreSight™ Components Technical Reference Manual*.

Section 9, *Appendix: Standard Memory Access Port Definitions* starting on page 45 provides reference material for standard implementations of Memory Access Ports.

Section 10, *Appendix: Cross-over with the ARM® Architecture* starting on page 47 provides detail of how the ADIv5 specification should be used when selecting or implementing Debug Access Ports for each of a variety of ARM® architecture variants.

2 ADIV5.0 ERRATA AND CLARIFICATIONS

2.1 Introduction

This section corrects errata in the *ARM® Debug Interface v5 Architecture Specification*, and also provides some clarifications where the *ARM® Debug Interface v5 Architecture Specification* might be ambiguous.

Please use the six-digit reference number provided in each section in any related correspondence with ARM Limited.

2.2 Erratum: ADIv5 gives wrong IDCODE value for SW-DPs (451413)

The *ARM® Debug Interface v5 Architecture Specification* gives the required value for the Part Number field, PARTNO: IDCODE[27:12], for an ARM implementation of a Serial Wire Debug Port (SW-DP) as 0xBA10. The correct value for current ARM implementations of Serial Wire protocol version 1 is 0xBA01.

In this manual, the SW-DP IDCODE register is renamed DPIDR. See *Debug Port architecture versions* on page 13. Also see *Serial Wire Protocol Version 2* on page 40.

2.3 Erratum: Emulation of debug reset request (497727)

Table 6-7 of the *ARM® Debug Interface v5 Architecture Specification* directs the reader to section 3.4.4 for details of how to emulate implementation of CDBGRSTACK and CDBGRSTREQ. Section 3.4.4 does not have that information.

The correct behavior is that if the debug reset control is not supported then:

- CDBGRSTACK is RAZ
- it is IMPLEMENTATION DEFINED whether CDBGRSTREQ is read/write or RAZ.

That is, in a standard implementation, CDBGRSTACK can be tied LOW. Whether, in such a system, CDBGRSTREQ registers values written to it is IMPLEMENTATION DEFINED.

See also *Limitations of CDBGRSTREQ/CDBGRSTACK* on page 18.

2.4 Erratum: References to asynchronous Serial Wire protocol (513764)

Several sections of the *ARM® Debug Interface v5 Architecture Specification*, including sections 2.2.2, 5.1, 5.2, 5.3, 5.4.6 and 6.2.6, refer to support for an asynchronous mode of operation for the CoreSight Serial Wire interface.

The Serial Wire interface does not support an asynchronous mode of operation.

Bits [7:6] of the Data Link Control Register (DLCR, formerly the Wire Control Register or WCR) are reserved, read as 0b01 and ignore writes. Bits [2:0] of the DLCR are reserved, RAZ/WI.

2.5 Erratum: JTAG-DP and SW-DP responses when Sticky Overrun Detect is enabled (604567)

Section 3.1.2 of the *ARM® Debug Interface v5 Architecture Specification* states that when Sticky Overrun Detect is enabled in the Debug Port CTRL/STAT register, the only permitted responses to transactions are OK/FAULT on the JTAG-DP and OK on the SW-DP.

These responses should be described not as “permitted responses”, as this implies is that they are the only responses generated in this state. However, the section goes on to correctly describe that when other responses are generated, the STICKYORUN flag is set. That is, all responses can be generated in overrun detection mode.

That is, in overrun detection mode, the Sticky Overrun flag STICKYORUN in the Debug Port (DP) Control/Status register is set to 1 if the response to any transaction is other than:

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

As the JTAG-DP and SW-DP protocols differ, the exact behavior is DATA LINK DEFINED:

For JTAG-DP:

The response is WAIT so long as the previous Access Port (AP) transaction remains not completed, and OK/FAULT thereafter. This is correctly described in the section *Sticky overrun behavior on DPACC and APACC accesses* in section 4.4.3 of the *ARM® Debug Interface v5 Architecture Specification*.

For SW-DP:

The first response to a transaction when a previous AP transaction has not completed is WAIT. Following responses will be FAULT, since the STICKYORUN bit is set to 1. This is correctly described in the section 5.4.5 of the *ARM® Debug Interface v5 Architecture Specification*.

Furthermore, when Sticky Overrun Detect is enabled, the STICKYORUN bit is also set to 1:

- when the DP issues a FAULT response
- following a protocol error.

2.6 Erratum: Host must drive the Park bit HIGH (618567)

Section 5.3.1 of the *ARM® Debug Interface v5 Architecture Specification* states that the host does not drive the line for the Park bit of Serial Wire protocol, instead relying on the line being pulled HIGH by the Serial Wire interface.

The host must in fact actively drive the line HIGH to park it before tri-stating the line for the turnaround period. This ensures the line is HIGH and is read as HIGH by the target. This is required as the pull-up on the Serial Wire interface is weak.

Section 5.4.6 of the *ARM® Debug Interface v5 Architecture Specification* further implies that a protocol error only occurs on incorrect parity in a command header. The target will in fact signal protocol error if any of the Parity, Stop or Park bits is not driven with the correct value.

2.7 Erratum: Line reset does not have to be detected asynchronously (625117)

Section 5.4.1 of the *ARM® Debug Interface v5 Architecture Specification* states that the serial protocol requires that any run of 50 consecutive 1s on the data input is detected as a line reset, regardless of the state of the protocol.

This is incorrect. The target may not detect a line reset if it is not in the correct state. It must detect a sequence of 50 consecutive 1s as a line reset if it receives the sequence when it is expecting the initial Start bit of a packet header.

However, regardless of what state the target is in, it will at least detect the sequence as a protocol error, as it will interpret at least one bit of the sequence as an invalid Stop bit in a packet header.

Having detected a protocol error, the target may respond to the DP DPIDR register read that follows the line reset sequence. However, if the target does not respond, the debugger must retry the line reset sequence.

For more information, see *Line reset* on page 43.

2.8 Erratum: FAULT response to CTRL/STAT writes in SW-DP (628016)

Table 5-3 of the *ARM® Debug Interface v5 Architecture Specification* correctly shows that SW-DP must issue a FAULT response to a *write* to the CTRL/STAT register if any sticky flag is set.

However, section 5.4.4 of the *ARM® Debug Interface v5 Architecture Specification* states that a SW-DP must not issue a FAULT response to an *access* to the IDCODE, CTRL/STAT or ABORT registers. The implication that this includes writes to CTRL/STAT is incorrect. An SW-DP must only not issue a FAULT response for:

- reads of the IDCODE register (the IDCODE register is read-only)
- writes to the ABORT register (the ABORT register is write-only)
- reads of CTRL/STAT.

2.9 Erratum: FAULT and WAIT responses to DLCR reads in SW-DP (628066)

Sections 5.4.3 and 5.4.4 of the *ARM® Debug Interface v5 Architecture Specification* list the SW-DP register accesses that must never return WAIT or FAULT responses.

However, Table 5-1 implies that reads of DLCR (formerly known as the WCR) will also return a result immediately and never respond with FAULT. This is incorrect. Reads of DLCR return a FAULT response if a sticky error flag is set.

Whether reads of a DLCR return a WAIT response if the AP is not ready is IMPLEMENTATION DEFINED. From Debug Port (DP) architecture version 2, reads of DLCR return a WAIT response if the AP is not ready.

2.10 Erratum: Use of DP RDBUFF by Serial Wire debug ports (DE 643221)

Sections 5.4.2 and 6.2.5 of the *ARM® Debug Interface v5 Architecture Specification* states that, for SW-DP, to obtain the result of an AP register read the next access must be either a second AP register read or a read of the DP Read Buffer (RDBUFF) register.

This is incorrect. To allow the debugger to recover from line errors, the next transaction after an AP register read can be any DP register read. If the next transaction is a DP register read other than a read of RDBUFF then a following AP register read or read of RDBUFF will return the result of the first AP register read.

However, if the next transaction following an AP register read is an AP register write or a DP register write then the result of the first AP register read is effectively lost, as any following AP register read or read of RDBUFF will return an UNKNOWN value.

2.11 Erratum: Incorrect description of JTAG-DP TAP state machine (674167)

Section 4.2.3 of the *ARM® Debug Interface v5 Architecture Specification* states that for the transition from the Capture-IR state to the Shift-IR state, the instruction register scan chain advances one bit.

The IEEE 1149.1 JTAG specification, upon which the JTAG-DP is closely based, requires that the scan chain advances for each rising edge of **TCK** whilst in the Shift state. That is, the scan chain does not advance for the transition from Capture-IR to Shift-IR. The same is true for data register scan chains on transitions from Capture-DR to Shift-DR.

For more information, see the *IEEE Standard Test Access Port and Boundary Scan Architecture*.

2.12 Clarification: UNPREDICTABLE use of transaction counter and transaction mode (414544)

An Access Port (AP) is permitted to define that accesses to certain registers in the AP are UNPREDICTABLE when the Debug Port's (DP) Transaction Counter field, TRNCNT, CTRL/STAT[23:12] field is non-zero.

An AP is also permitted to define that the Sticky Compare flag, STICKYCMP, CTRL/STAT[4] is set to an UNKNOWN value in response to an access made to certain registers in the AP when the Transfer Mode field, TRNMODE, CTRL/STAT[3:2] is any value other than 0b00 (*Normal operation*), regardless of the value in the register.

In general, both UNPREDICTABLE behaviors are expected to be defined for any register that does not change its value other than in response to an APACC or DPACC access.

Table 1 lists the register accesses defined by the ARM Debug Interface v5 to:

- be UNPREDICTABLE when the TRNCNT field is not zero
- set the STICKYCMP flag to an UNKNOWN value when the TRNMODE field is not zero.

Access Port type	Register being accessed
Any	Identification Register (IDR)
	Any reserved register location
Memory Access Port (MEM-AP)	Control/Status Word (CSW)
	Transfer Address Register (TAR)
	ROM Table Base Register (BASE)
	Configuration Register (CFG)
JTAG Access Port (JTAG-AP)	Control/Status Word (CSW)
	Port Select Register (PSEL)

Table 1: UNPREDICTABLE register accesses

If the DP SELECT register selects a non-existent AP, then register accesses:

- are UNPREDICTABLE when the TRNCNT field is not zero
- set the STICKYCMP flag to an UNKNOWN value when the TRNMODE field is not zero.

See also *Minimal Debug Port (MINDP) extensions* on page 15.

2.13 Clarification: Use of PWRUPREQ signals (424966)

CSYSPWRUPREQ and CDBGPWRUPREQ, bits [30,28] of the Debug Port CTRL/STAT register, can independently request power up of debug functions in the debug and system power domains.

The system power domain *includes* the debug power domain. CDBGPWRUPREQ must be set to 1 whenever CSYSPWRUPREQ is set to 1. Setting bits [30,28] to 0b10 gives UNPREDICTABLE system behavior.

Given this, the example circuit shown in Figure 3-5 of the *ARM® Debug Interface v5 Architecture Specification* can be replaced with the simpler circuit shown in Figure 1.

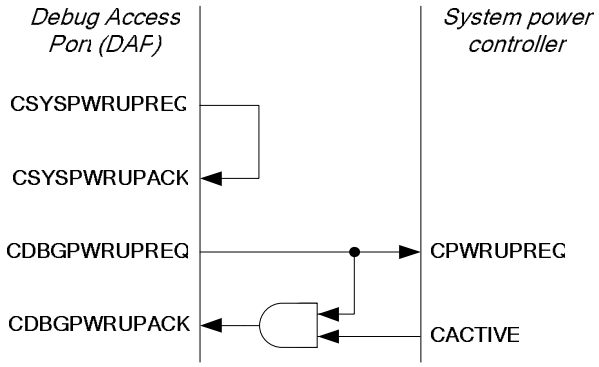

Figure 1: Signal generation for single power domain

This manual also:

- gives clarification of the roles of these power-up request bits
- describes some limitations of these power-up request bits.

For more details, see *Power-up and Reset Controls* on page 16, which clarifies the descriptions of the power-up request/acknowledge signals, and replaces parts of the descriptions in section 3.4 of the *ARM® Debug Interface v5 Architecture Specification*.

2.14 Clarification: Use of the RESEND register (DE 618517)

ARM recommends that debuggers only access the SW-DP RESEND register when a failed read has been indicated by the SW-DP, and not at other times. This is because an implementation is permitted to treat reads of RESEND as a protocol error in the case where it cannot resend the information.

2.15 Clarification: AP accesses following a DAP abort (DE 618519)

Section 6.2.1 of the *ARM® Debug Interface v5 Architecture Specification* describes the behavior following a DAP abort. The section states that, following a DAP abort:

- new transactions can be accepted by the DP
- an AP access to the AP that was aborted might return additional WAIT responses
- other APs can be accessed.

This is ambiguous, since the WAIT response is generated by the DP and is the DP's mechanism to not accept a new transaction. Section 6.2.1 should state that, following a DAP abort:

- An AP access to the AP that was aborted might not be accepted by the DP, with the DP returning a WAIT response.
- A DP access or an AP access to any other AP will be accepted by the DP. This includes AP accesses to non-existent APs, which are defined to behave as RAZ/WI.

2.16 Clarification: ReadResult following IDCODE and ABORT scans (DE 618570)

On entering the Capture-DR state with the JTAG-DP instruction register (IR) set to the APACC or DPACC instructions, the value captured for ReadResult is always UNKNOWN if either:

- the most recent DR scan was not made with IR set to one of the DPACC, APACC or BYPASS instructions
- there has been no DR scan since leaving the Test-Logic-Reset state.

2.17 Clarification: Unaligned access to Banked Data registers (DE 643218)

Section 8.1.3 of the *ARM® Debug Interface v5 Architecture Specification* requires that accesses to a MEM-AP's BD0 to BD3 access addresses starting at the address TAR[31:4].

The description in section 8.1.3 should further state that the value in TAR[3:0] is ignored in constructing the access address for a banked register transfer, and:

- bits [3:2] of the access address depend solely on which of the four banked data registers is being accessed.
- bits [1:0] of the access address will always be zero.

3 ADIV5.1 UPDATES

3.1 Introduction

This section describes the main changes in ADIv5 between version 5.0 and version 5.1.

3.2 Debug Port architecture versions

The *ARM® Debug Interface v5 Architecture Specification* describes two variants of the Debug Port (DP) linked to two implementations, SW-DP and JTAG-DP. This manual formalizes those differences by introducing the concept of a DP architecture version.

The definition of the DP identification registers is updated to allow for this change, and also to accommodate future additions to the DP programmer's model.

The *ARM® Debug Interface v5 Architecture Specification* describes a Debug Port register named IDCODE. For SW-DP, this register is accessed as DP register 0; for JTAG-DP it is accessed via the JTAG IDCODE instruction. The JTAG IDCODE is part of the IEEE 1149.1 Test Access Port specification, and therefore lies outside the scope of the ADIv5 specification. This manual separates the two concepts.

The versions of the DP architecture described in the *ARM® Debug Interface v5 Architecture Specification* are DP architecture version 0 for JTAG-DP and DP architecture version 1 for SW-DP. ADIv5.1 defines an additional DP architecture version 2.

For full details, see *Debug Port Architecture Versions* on page 19.

3.3 The identification model for Access Ports

The *ARM® Debug Interface v5 Architecture Specification* defines a single, common identification register that must be implemented by all Access Ports, IDR. In version 5.0, bit [16] of this register is defined as the Access Port Class field and bits [15:8] as reserved, UNK. Version 5.1 redefines the bits in IDR, as shown in Figure 2.

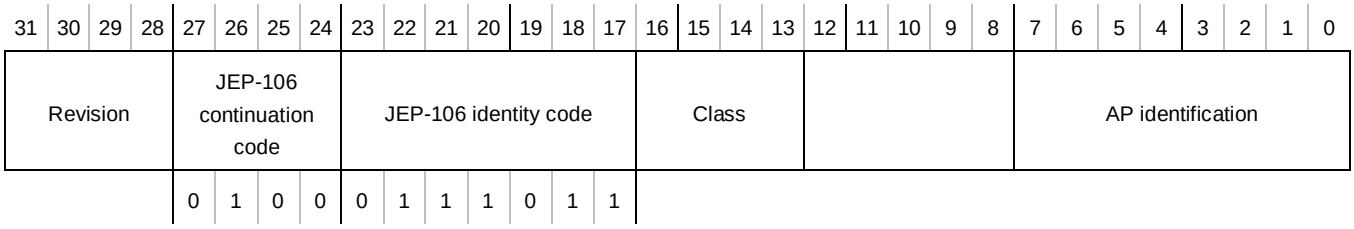

Figure 2: Access Port ID Register bit assignments

Revision, JEP-106 continuation code, and JEP-106 identity code, bits [31:17]

Unchanged. The codes for ARM Limited are shown. See the *ARM® Debug Interface v5 Architecture Specification*.

Class, bits [16:13]

This field defines the class of Access Port. An Access Port belongs to a class if it follows a programmer's model defined as part of the ADIv5 specification or extensions to it.

Table 2 lists the Access Port classes defined by the ADIv5 specification. All other values are reserved.

Class	Description
0b0000	No defined class
0b1000	Memory Access Port. See Chapter 11 of the *ARM® Debug Interface v5 Architecture Specification.*

Table 2: Class field values

Bits [12:8]

Reserved, must-be-zero. This field is reserved for future ID register fields. If a debugger reads a non-zero value in this field, it must treat the AP as unidentifiable.

AP identification, bits [7:0]

Access Port Identification. This field identifies the AP implementation. Each AP designer must maintain their own list of implementations and associated Identification codes.

In an AP implementation by ARM Limited this field is sub-divided as bits [7:4] *Variant*, bits [3:0] *Type*. Table 3 lists the possible values of the Type field for an AP designed by ARM Limited. The Variant field is used to identify different implementations of the same Type.

Class	Type	Bus type
0b0000	0x0	Not used by MEM-APs; indicates JTAG-AP. Variant field must be non-zero.
0b1000	0x1	AMBA AHB bus
0b1000	0x2	AMBA APB bus
-	-	All other values are reserved.

Table 3: Interpretation of IDR[3:0] for ARM designed MEM-APs

Note

The *ARM® Debug Interface v5 Architecture Specification* required this designation of Variant and Type for all ARM designed APs. In ADIv5.1, this interpretation of the AP Identification field is restricted.

ARM Limited will only assign IDR[3:0] values 0x0, 0x1 and 0x2 in a manner backwards compatible with ADIv5. However, for all other IDR[3:0] values, the Type interpretation only applies for MEM-APs.

The AP Identification field must be unique for each Access Port designed by a given designer, regardless of the value of IDR.Class.

3.4 Multiple Protocol Interoperability

The Serial Wire/JTAG Debug Port (SWJ-DP) allows for multiple protocol interoperability. An implementation of SWJ-DP is described in the *CoreSight™ Components Technical Reference Manual*. This manual incorporates those existing implementations within ADIv5.

A further extension, *Dormant state*, is introduced to increase interoperability with other protocol standards.

For full details, see *Multiple Protocol Interoperability* on page 30.

3.5 Multi-drop Serial Wire protocol extensions

Serial Wire protocol version 1 is a point-to-point architecture, supporting connection between a single host and a single device. It allows connection to multiple devices only by providing additional connections from the host.

This has a number of disadvantages:

- It complicates the physical connection standard, by having variants with different numbers of connections.
- It increases the number of pins required for the connector on the device PCB. This is unacceptable where size is a limiting factor.
- It increases the number of pins required on a package with multiple dies inside.
- It makes it difficult to integrate multiple platforms accessed by the Serial Wire protocol into the same chip.

Techniques to solve this require connections that are shared between multiple Serial Wire devices. This is detrimental to the maximum speed of operation, but in many situations this is an acceptable trade-off.

Serial Wire protocol version 2 adds the multi-drop extension, which:

- enables a two wire host connection to communicate simultaneously with multiple devices
- permits an effectively unlimited number of devices to be connected simultaneously, subject to electrical constraints
- is backwards-compatible: provision of multi-drop support in a device does not break point-to-point compatibility with existing host equipment that does not support multi-drop extensions
- permits a device to power down completely while that device is not selected
- prevents multiple devices from driving the wire simultaneously, and continues to support the wire being actively driven both HIGH and LOW, maintaining a high maximum clock speed
- also allows for multi-drop connections incorporating devices that do not implement the Serial Wire protocol.

Serial Wire protocol version 2 is described in *Serial Wire Protocol Version* 2 starting on page 40.

3.6 Minimal Debug Port (MINDP) extensions

The Minimal Debug Port programmer's model is a cut-down version of the Debug Port intended for low-gate-count implementations.

For minimal DP implementations, the following Debug Port features are removed:

- the Transaction Counter
- Pushed Verify operation
- Pushed Find operation.

When MINDP is implemented:

- DPIDR.MIN, bit [16] of Debug Port ID Register, DPIDR, is RAO.
- The following fields, bits [23:8] and [4:3], of the Control/Status Register, CTRL/STAT, are reserved, RAZ:
  - CTRL/STAT.TRNCNT
  - CTRL/STAT.MASKLANE
  - CTRL/STAT.STICKYCMP
  - CTRL/STAT.TRNMODE

  Writing a non-zero value to any of these fields is UNPREDICTABLE.
- ABORT.STKCMPCLR, bit [1] of the Abort Register, ABORT, is reserved, SBZ. Writing 1 to this bit is UNPREDICTABLE.

3.7 Serial Wire protocol programmable turnaround period

Section 6.2.6 of the *ARM® Debug Interface v5 Architecture Specification* describes how the turnaround tri-state period for the Serial Wire protocol can be programmed using the WCR.TURNROUND register.

In ADIv5.1 support for varying the turnaround tri-state period is IMPLEMENTATION DEFINED. An implementation that does not support variable turnaround must treat writing a value other than 0b00 to WCR.TURNROUND as an immediate protocol error.

ARM deprecates use of turnaround tri-state periods other than 1.

3.8 Required support of Memory Access Port (MEM-AP) packed transfers

Section 8.2.7 of the *ARM® Debug Interface v5 Architecture Specification* requires that if a MEM-AP supports access sizes smaller than word it must also support packed transfers.

In ADIv5.1 that requirement is relaxed. Support for packed transfers is IMPLEMENTATION DEFINED.

From ADIv5.1, if a MEM-AP does not support packed transfers then writing 0b10 to CSW.AddrInc selects the *auto-increment off* mode. Subsequently reading back CSW will return 0b00 for CSW.AddrInc.

This was not required by the definition of CSW.AddrInc in the *ARM® Debug Interface v5 Architecture Specification*.

3.9 Scope of ADIv5.1

ADIv5 encompasses a range of technologies and architectures. ADIv5 version 5.0 and version 5.1 are packages of versions of these technologies and architectures. Version 5.1 encompasses all that is defined in version 5.0: an implementation of ADIv5.0 is also an implementation of ADIv5.1.

Figure 3 shows the main components of ADIv5, split between the two main areas of ADIv5: the *Access Port architecture* and the *Debug Port architecture*. For each component, Figure 3 shows whether the component was defined in ADIv5.0 or is introduced with ADIv5.1. ADIv5.1 adds a third major area of ADIv5: *Debug Port interoperability*.

For example, Debug Port architecture version 1 is defined by ADIv5.0 but in that version only supports Serial Wire protocol (version 1), and is called SW-DP. In ADIv5.1 it is formalized as *Debug Port architecture version 1*, and also allows JTAG protocol.

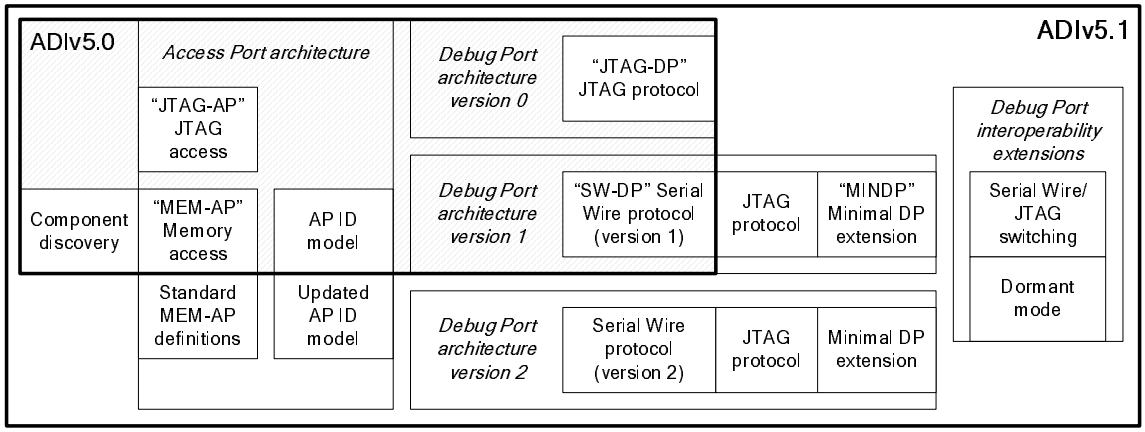

Figure 3: Scope of ADIv5.0 and ADIv5.1 and the architecture components defined by each

4 POWER-UP AND RESET CONTROLS

4.1 Clarification of PWRUPREQ/ACK signals

The *ARM® Debug Interface v5 Architecture Specification* defines two pairs of signals:

- **CDBGPWRUPREQ** and **CDBGPWRUPACK**
- **CSYSPWRUPREQ** and **CSYSPWRUPACK**

Table 4 summarizes the programmer's model for **CDBGPWRUPREQ/ACK** and **CSYSPWRUPREQ/ACK**.

Signal	Programmer's model
CDBGPWRUPREQ	Bit[28] of the CTRL/STAT register
CDBGPWRUPACK	Bit[29] of the CTRL/STAT register
CSYSPWRUPREQ	Bit[30] of the CTRL/STAT register
CSYSPWRUPACK	Bit[31] of the CTRL/STAT register

Table 4: Debug Port programmer's model

These signals are expected to provide hints to the system power/clock controller. The following sections describe these signal pairs in turn.

4.1.1 CDBGPWRUPREQ and CDBGPWRUPACK

The **CDBGPWRUPREQ** signal indicates that the debugger requires the debug resources to be *communicative*. The power/clock controller must power-up and run the clocks of as many domains as necessary to comply with this request.

Communicative means that the debugger can access at least enough registers of the debug resource for it to determine the state of the resource. Whether the resource is *active* is IMPLEMENTATION DEFINED.

The power/clock controller must honor **CDBGPWRUPREQ** for as long as it is asserted. For example, if a component in a debug power domain requests to have its clocks stopped, the request must be emulated. This includes all components with a single shared debug / core power domain: power-down must be emulated for non-debug logic within that power domain.

Components with a split debug / core power domains must have *at least* the minimal debug interface powered up. Even if some debug resources are contained in the core power domain, then power can be removed from the core power domain so long as both:

- there is some means to save and restore the state of these resources over the core power domain being powered down
- the core power domain does not need to be powered for the debugger to be able to communicate with the debug resources.

The means to save and restore these resources might include software means. If the debug resources do lose their value when power is removed from the core power domain, then the debug logic must include means for the debugger to discover that the programmed values have been lost.

CDBGPWRUPACK is the acknowledge signal for the **CDBGPWRUPREQ** request signal. **CDBGPWRUPACK** must be asserted for as long as **CDBGPWRUPREQ** is asserted.

4.1.2 CSYSPWRUPREQ and CSYSPWRUPACK

The **CSYSPWRUPREQ** signal indicates that the debugger requires all debug resources to be *active*. The power/clock controller must power-up and run the clocks of as many domains as necessary so that all debug resources are active.

Active means that the debug resource is capable of performing its debug function. An active resource is also communicative.

The power/clock controller must honor **CSYSPWRUPREQ** for as long as it is asserted.

CSYSPWRUPREQ will have no effect on debug components with a single power-domain, as **CDBGPWRUPREQ** will ensure those components are powered. Similarly, it has no effect on components with a pure debug / core power-domain split, as those components have no debug logic in the core power domain. However, for components where some debug resources are in the core power domain, **CSYSPWRUPREQ** must be seen as a request to emulate power saving in the core power domain.

CSYSPWRUPACK is the acknowledge signal for the **CSYSPWRUPREQ** request signal. **CSYSPWRUPACK** must be asserted for as long as **CSYSPWRUPREQ** is asserted. Whenever **CSYSPWRUPREQ** is asserted by the debugger, **CDBGPWRUPREQ** must also be asserted.

4.2 Limitations of CDBGRSTREQ/CDBGRSTACK

The DP provides two bits, CDBGRSTREQ and CDBGRSTACK, bits [27:26] of the CTRL/STAT register, for reset control of the debug domain.

The *ARM® Debug Interface v5 Architecture Specification* shows how these bits can be used to drive the debug reset signal, **PRESETDBGn**. In a real system there are likely to be other reset signals associated with other debug buses. For example in an ARM CoreSight system, **ATRESETn** resets all register in the AMBA Trace Bus domain.

Because debug logic might be accessible by the system, an implementation might have corner cases if CDBGRSTACK is set whilst the system is using the debug logic. For example, the reset might reset a debug bus whilst a transaction is in progress, causing a system or software malfunction.

ARM recommends that such approaches are not mixed without extreme care; that such system-level usage of debug components is not combined with a reset system that allows those components to be reset without the knowledge of the system.

For more details on this issue, contact ARM.

If the debug reset control is not implemented, CDBGRSTACK is RAZ and CDBGRSTREQ might be RAZ. See *Erratum: Emulation of debug reset request (497727)* on page 8.

5 DEBUG PORT ARCHITECTURE VERSIONS

The *ARM® Debug Interface v5 Architecture Specification* describes two forms of Debug Port (DP): the JTAG Debug Port and the Serial Wire Debug Port. This manual reclassifies these as different versions of a common DP architecture. Table 5 lists the versions of the Debug Port (DP) architecture described by ADIv5.1.

Version number	Description
0	*DP architecture version 0* on page 19
1	*DP architecture version 1* on page 21
2	*DP architecture version 2* on page 26

Table 5: Debug Port architecture versions

JTAG-DP always implements a JTAG TAP ID Register, which is not related to the Debug Port architecture version.

5.1 The JTAG TAP ID Register, IDCODE

The JTAG TAP ID Register, IDCODE, is always present on all JTAG-DP implementations. It provides identification information about the JTAG-DP, such as which scan-chains are implemented.

It is accessed using its own scan chain. See the *ARM® Debug Interface v5 Architecture Specification*.

It is read-only and always accessible. Figure 4 shows the register bit assignment, with their values for an ARM design.

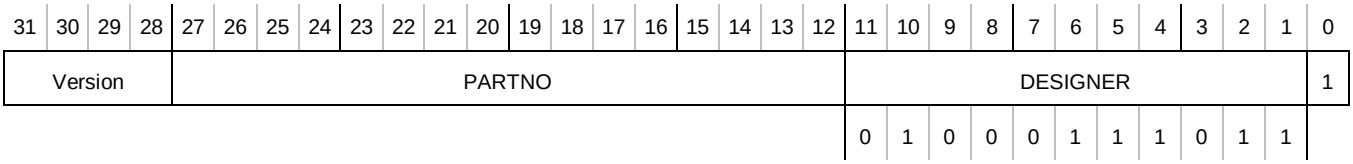

Figure 4: JTAG TAP ID Register bit assignments

Version, bits [31:28]
Version code. The meaning of this field is IMPLEMENTATION DEFINED.

PARTNO, bits [27:12]
Part Number for the DP TAP. This value is provided by the designer of the Debug Port TAP and must not be changed. Current DPs designed by ARM have the JTAG-DP PARTNO value 0xBA00.

DESIGNER, bits [11:1]
Designer ID. An 11-bit code formed from the JEDEC JEP-106 continuation code and identity code. The ID identifies the designer of the JTAG-DP TAP. The ARM value for this field, shown above, is 0x23B. Other designers must insert their own JEDEC assigned code here.

Bit [0] RAO.

5.2 DP architecture version 0

Architecture version 0 only supports JTAG-DPs. Whether a JTAG-DP implements version 0 of the architecture or some other version can be determined by a table lookup based on the JTAG IDCODE value.

In DP architecture version 0:

- the SELECT register is read/write
- bits [5:4] of the CTRL/STAT register are R/W1C

- bits [31:1] of the ABORT register are reserved, SBZ and bit [0] SBO on writes.

Note

Debug Port architecture version 0 is the DP architecture defined for JTAG-DP in the *ARM® Debug Interface v5 Architecture Specification*.

The DP register map in version 0 of the DP architecture is shown in Table 6.

Address	Access	Description
0x0	UNPREDICTABLE	Reserved
0x4	Read/write	*The Control/Status Register, CTRL/STAT (architecture version 0)* on page 20
0x8	Read/write	*The AP Select Register, SELECT (architecture version 0)* on page 21
0xC	Read-only	The Read Buffer, RDBUFF

Table 6: Debug Port register map, architecture version 0

The DP must implement an ABORT register. In DP architecture version 0, how this register is accessed is DATA LINK DEFINED. In JTAG-DP this is implemented through the ABORT instruction.

5.2.1 The Control/Status Register, CTRL/STAT (architecture version 0)

In DP architecture version 0, the Control/Status Register, CTRL/STAT, is:

- a read/write register
- accessed by a read or write of DP register 0x4.

Figure 5 shows the register bit assignments.

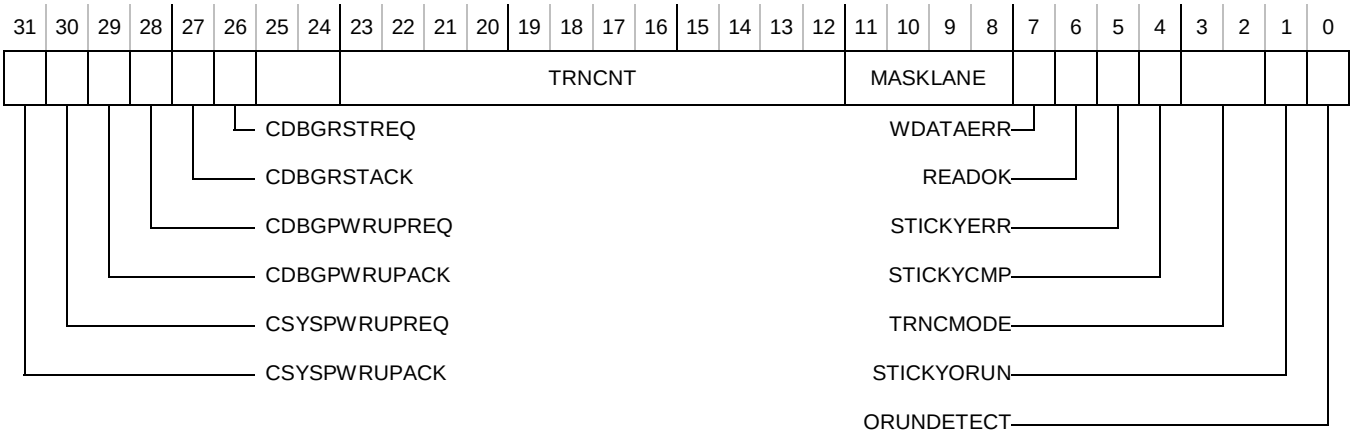

Figure 5: Control/Status Register (architecture version 0) bit assignments

Bits [31:8,3:2,0]

See the *ARM® Debug Interface v5 Architecture Specification*.

WDATAERR, bit [7] and READOK, bit [6]

Support for these bits is DATA LINK DEFINED:

— for JTAG-DP these bits are reserved, RAZ/WI.

DP architecture version 0 only supports JTAG-DP.

STICKYERR, bit [5], STICKYCMP, bit [4] and STICKYORUN, bit [1]

These bits are defined as R/W1C. For a definition of these bits, see the *ARM® Debug Interface v5 Architecture Specification*.

5.2.2 The AP Select Register, SELECT (architecture version 0)

In DP architecture version 0, the AP Select Register, SELECT, is:

- a read/write register
- accessed by a read or write of DP register 0x8.

Figure 6 shows the register bit assignments.

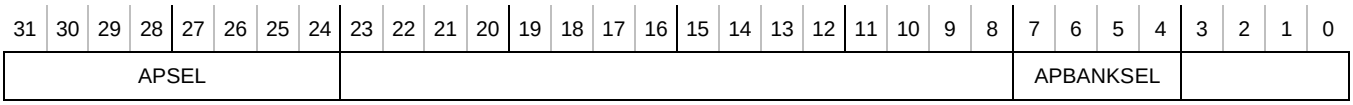

Figure 6: AP Select Register (architecture version 0) bit assignments

APSEL and APBANKSEL, bits [31:24,7:4]
See the *ARM® Debug Interface v5 Architecture Specification*.

Bits [23:8,3:0]
Reserved, UNK/SBZP.

For compatibility with other DP architecture versions, ARM recommends that tools do not read the SELECT register.

5.2.3 The AP Abort Register, ABORT (architecture version 0)

In DP architecture version 0, the AP Abort Register, ABORT, is

- a write-only register
- accessed in a DATA LINK DEFINED manner:
  - for JTAG-DP it is accessed via its own scan-chain.

  DP architecture version 0 only supports JTAG-DP.

Figure 7 shows the register bit assignments.

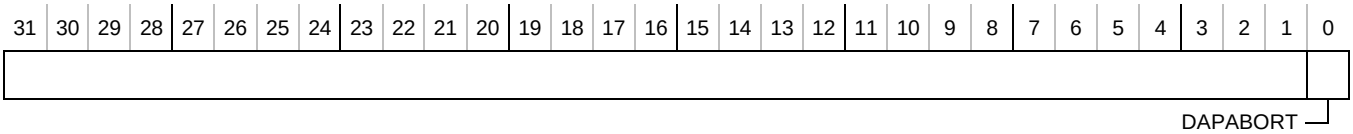

Figure 7: AP Abort Register (architecture version 0) bit assignments

Bits [31:1]
Reserved, SBZ.

DAPABORT, bit [0]
SBO. This bit must be written as 1. See the *ARM® Debug Interface v5 Architecture Specification*.

Note

For JTAG-DP, the ABORT register is written from bits [34:3] of the data register scan-chain selected by the ABORT instruction. Writing a value other than 0x000000008 to the ABORT scan-chain gives UNPREDICTABLE results.

For more details on the ABORT scan-chain see the *ARM® Debug Interface v5 Architecture Specification*

5.3 DP architecture version 1

Architecture version 1 extends version 0 with support for:

- the Debug Port Identification Register, DPIDR

- the AP Abort Register, ABORT
- the Data Link Control Register, DLCR.

The definition of other registers is also changed:

- the behavior on writing to bits [5:4,1] of the CTRL/STAT register is DATA LINK DEFINED
- bit [0] of the SELECT register is defined as DPBANKSEL
- the SELECT register is write-only.

Note

Debug Port architecture version 1 is the DP architecture defined for SW-DP in the *ARM® Debug Interface v5 Architecture Specification*.

The DP register map in version 1 of the DP architecture is shown in Table 7.

Address	DPBANKSEL	Access	Description
0x0	-	Read-only	*The Debug Port ID Register, DPIDR (architecture version 1)* on page 22
		Write-only	*DATA LINK DEFINED registers* on page 26
0x4	0	Read/write	*The Control/Status Register, CTRL/STAT (architecture version 1)* on page 25
	1	Read/write	*The Data Link Control Register, DLCR* on page 26
0x8	-	Read-only	*DATA LINK DEFINED registers* on page 26
		Write-only	*The AP Select Register, SELECT (architecture version 1)* on page 25
0xC	-	Read-only	The Read Buffer, RDBUFF
		Write-only	*DATA LINK DEFINED registers* on page 26

Table 7: Debug Port register map, architecture version 1

5.3.1 The Debug Port ID Register, DPIDR (architecture version 1)

The Debug Port ID Register, DPIDR, provides identification information about the Debug Port, such as what features are implemented.

The DPIDR is:

- a read-only register
- accessed by a read of DP register 0x0
- not affected by the value of SELECT.CTRLSEL.

Figure 8 shows the register bit assignment, with their values for an ARM design.

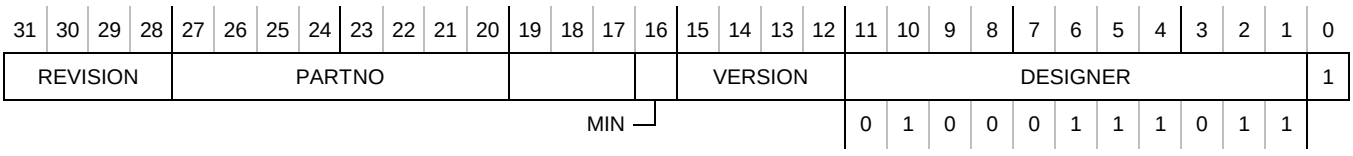

Figure 8: Debug Port ID Register (architecture version 1) bit assignments

REVISION, bits [31:28]

Revision code. The meaning of this field is IMPLEMENTATION DEFINED.

PARTNO, bits [27:20]

Part Number for the Debug Port. This value is provided by the designer of the Debug Port and must not be changed. Current DPs designed by ARM Limited have the PARTNO value 0xBA.

Bits [19:17]

Reserved, UNK.

MIN, bit [16]

Minimal Debug Port functions implemented.

1 Transaction Counter, Pushed Verify and Pushed Find operations are not implemented.

0 Transaction Counter, Pushed Verify and Pushed Find operations are implemented.

VERSION, bits [15:12]

Version of the Debug Port architecture implemented. The valid values for this field are:

0x0 Reserved. Implementations of DP architecture version 0 do not implement DPIDR.

0x1 DP architecture version 1.

All other values are reserved.

DESIGNER, bits [11:1]

Designer ID. An 11-bit code formed from the JEDEC JEP-106 continuation code and identity code. The ID identifies the designer of the Debug Port. The ARM Limited value for this field, shown above, is 0x23B. Other designers must insert their own JEDEC-assigned code here.

Bit [0] RAO.

Note

JTAG-DP implementations of architecture version 1 are required to also implement the IDCODE instruction and IDCODE scan-chain. The architecture does not require that the DPIDR value and JTAG IDCODE value are the same. The JTAG IDCODE value identifies the JTAG-DP TAP and its designer. See *The JTAG TAP ID Register,* IDCODE on page 19.

5.3.2 The AP Abort Register, ABORT (architecture version 1)

In DP architecture version 1, the ABORT register:

- is a write-only register
- accessed in a DATA LINK DEFINED manner:
  - for SW-DP it is accessed by a write to DP register 0x0
  - for JTAG-DP it is accessed via its own scan-chain
- has fields that are not defined for architecture version 0.

Figure 9 shows the register bit assignments.

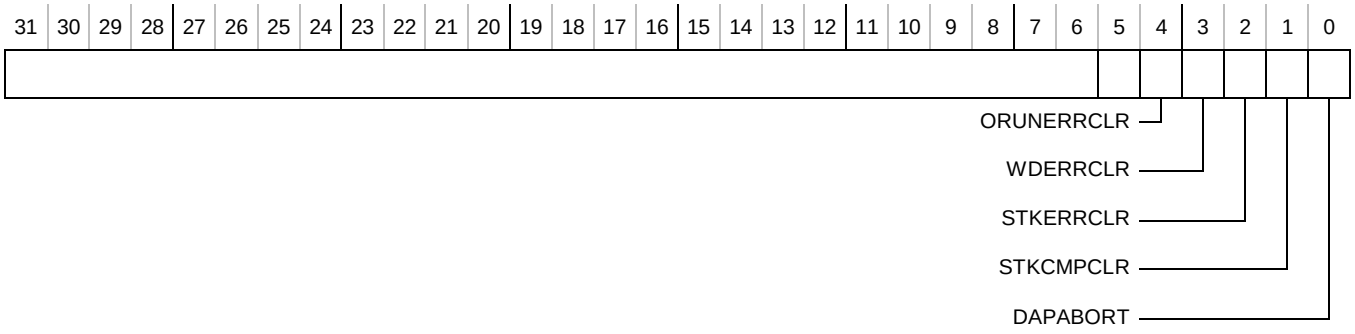

Figure 9: AP Abort Register (architecture version 1) bit assignments

Bits [31:5]

Reserved, SBZ.

ORUNERRCLR, WDERRCLR, STKERRCLR, STKCMPCLR, and DAPABORT, bits [4:0]

See the *ARM® Debug Interface v5 Architecture Specification*.

ABORT scan-chain operation, JTAG-DP (architecture version 1)

As in DP architecture version 0, when the ABORT instruction is the current instruction in the IR, the serial path between **TDI** and **TDO** is connected to a 35-bit scan chain that is used to access the AP Abort Register.

On Capture-DR, an UNKNOWN value is written to this scan chain. The debugger must scan 0 into bits [34:8,2:0] of this scan chain. This value:

- writes the RnW bit as 0
- writes the A[3:2] field as 0
- writes the SBZ fields of the AP Abort Register as 0.

The effect of scanning a 1 into any of bits [34:8,2:0] of this scan chain is UNPREDICTABLE. On Update-DR, bits [34:3] of this scan chain are written to the AP Abort Register.

Figure 10 shows the operation of this scan chain.

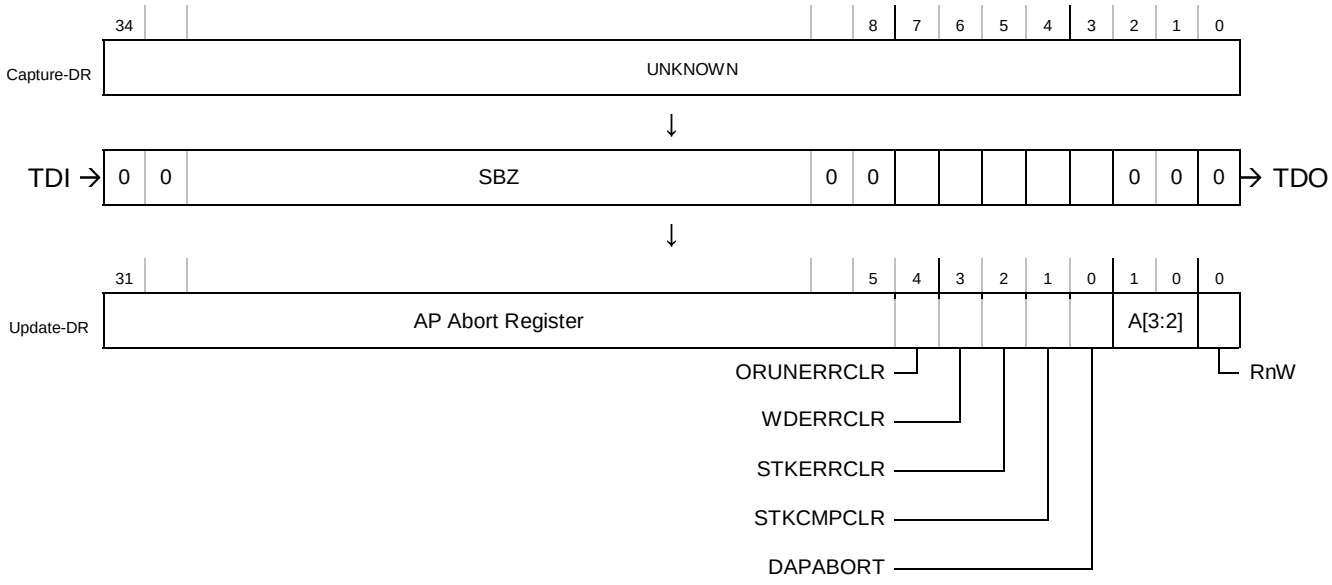

Figure 10: JTAG-DP ABORT scan-chain operation

Note

For JTAG-DP, the ABORT register is written from bits [34:3] of the data register scan-chain selected by the ABORT instruction. As in DP architecture version 0, this scan-chain returns an UNKNOWN value on Capture-DR, and at Update-DR bits [2:0] are SBZ.

In DP architecture version 0, bits [34:4] of the scan-chain are SBZ and bit [3] is SBO, since ABORT[31:1] is SBZ and ABORT[0] is SBO. Bits [2:0] of the scan-chain are SBZ.

In DP architecture version 1, bits [34:8] are SBZ and no bits are SBO, since ABORT[31:5] is SBZ and ABORT[4:0] are as defined above.

For more details on the ABORT scan-chain see the *ARM® Debug Interface v5 Architecture Specification*.

5.3.3 The AP Select Register, SELECT (architecture version 1)

In DP architecture version 1, the SELECT register

- is a write-only register
- is accessed by a write of DP register 0x8
- has fields that are not defined for architecture version 0.

Figure 11 shows the register bit assignments.

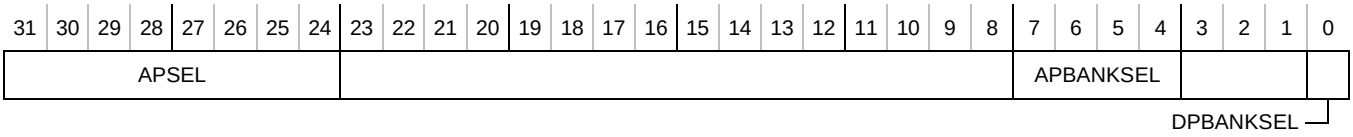

Figure 11: AP Select Register (architecture version 1) bit assignments

APSEL, APBANKSEL, and DPBANKSEL, bits [31:24,7:4,0]

See the *ARM® Debug Interface v5 Architecture Specification*.

> **Note**
> In the *ARM® Debug Interface v5 Architecture Specification* DPBANKSEL is named CTRLSEL.

Bits [23:8,3:1]

Reserved, UNK/SBZP.

The reset value of the SELECT register is UNKNOWN. Tools must initialize SELECT before accessing any AP registers or registers affected by DPBANKSEL.

5.3.4 The Control/Status Register, CTRL/STAT (architecture version 1)

In DP architecture version 1, the CTRL/STAT register:

- is a read/write register
- is accessed by a read or write of DP register 0x4 when SELECT.DPBANKSEL is 0x0
- has behavior for some fields that differs from that for architecture version 0.

Figure 12 shows the register bit assignments.

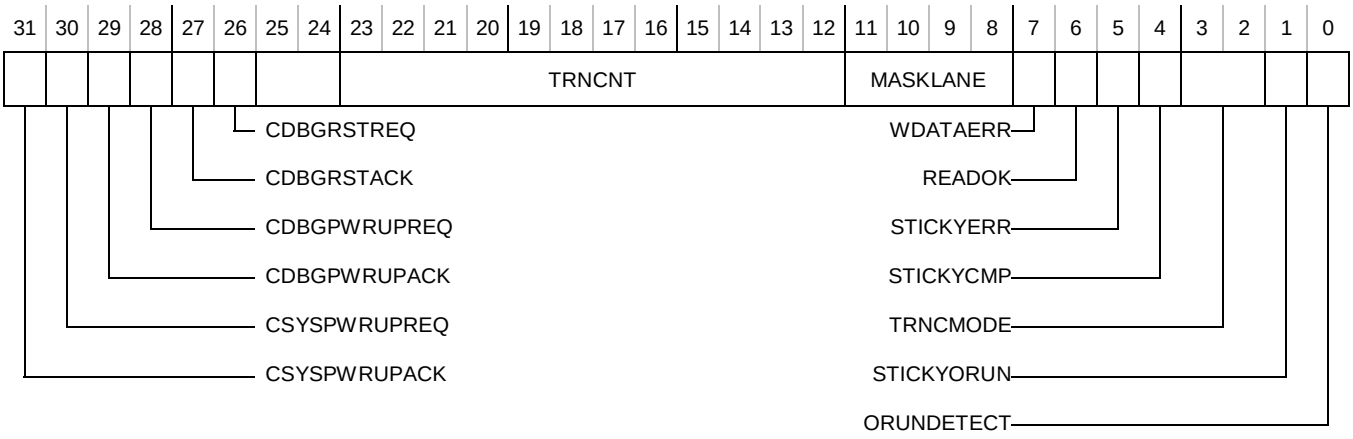

Figure 12: Control/Status Register (architecture version 1) bit assignments

Bits [31:24:2,0]

See the *ARM® Debug Interface v5 Architecture Specification*.

TRNCNT, MASKLANE and TRNMODE, bits [23:8,3:2]

Not supported in MINDP configuration. In MINDP configuration, writing a value other than zero to either TRNCNT or TRNMODE is UNPREDICTABLE.

For a definition of these bits, see the *ARM® Debug Interface v5 Architecture Specification*.

WDATAERR and READOK, bits [7:6]

Support for these bits is DATA LINK DEFINED.

— For SW-DP these bits are RO/WI.

— For JTAG-DP these bits are reserved, RAZ/WI.

For a definition of these bits, see the *ARM® Debug Interface v5 Architecture Specification*.

STICKYERR, STICKYCMP, and STICKYORUN, bit [5:4,1]

The behavior on writing to these bits is DATA LINK DEFINED.

— For SW-DP, these bits are RO/WI.

— For JTAG-DP, these bits are R/W1C. Writing a 0 to these bits is ignored.

For a definition of these bits, see the *ARM® Debug Interface v5 Architecture Specification*.

STICKYCMP is not supported in MINDP configuration. In MINDP configuration, writing a 1 to STICKYCMP is UNPREDICTABLE.

5.3.5 The Data Link Control Register, DLCR

The Data Link Control Register, DLCR, is:

- a DATA LINK DEFINED read/write register
- defined in architecture version 1
- accessed by a read or write of DP register 0x4 when SELECT.DPBANKSEL is 0x1.

The contents of the Data Link Control Register, DLCR, are DATA LINK DEFINED.

In the *ARM® Debug Interface v5 Architecture Specification* the Serial Wire definition of DLCR is named WCR. The definition of DLCR for Serial Wire is unchanged.

For JTAG-DP, DLCR is reserved. Accessing the DLCR is UNPREDICTABLE.

5.3.6 DATA LINK DEFINED registers

For SW-DP the DATA LINK DEFINED registers are listed in Table 8.

Address	Access	Description
0x0	Write-only	*The AP Abort Register, ABORT (architecture version 1)* on page 23
0x8	Read-only	The Read Resend Register, RESEND. See the *ARM® Debug Interface v5 Architecture Specification*.
0xC	Write-only	Reserved

Table 8: SW-DP DATA LINK DEFINED registers

For JTAG-DP both DATA LINK DEFINED registers are reserved.

Accessing a reserved DATA LINK DEFINED register is UNPREDICTABLE.

5.4 DP architecture version 2

Architecture version 2 extends version 1 with support for:

- the Target Identifier Register, TARGETID
- the Data Link Protocol Identification Register, DLPIDR.

The definition of other registers is also changed:

- the DPBANKSEL field in the SELECT register is extended to bits [3:0].

The DP register map in version 2 of the DP architecture is shown in Table 9.

Address	DPBANKSEL	Access	Description
0x0	-	Read-only	*The Debug Port ID Register, DPIDR (architecture version 2)* on page 27
		Write-only	DATA LINK DEFINED registers
0x4	0x0	Read/write	The Control/Status Register, CTRL/STAT
	0x1	Read/write	The Data Link Control Register, DLCR
	0x2	Read-only	*The Target Identification Register, TARGETID* on page 28
	0x3	Read/write	*The Data Link Protocol Identification Register, DLPIDR* on page 29
0x8	-	Read-only	DATA LINK DEFINED registers
		Write-only	*The AP Select Register, SELECT (architecture version 2)* on page 28
0xC	-	Read-only	The Read Buffer, RDBUFF
		Write-only	DATA LINK DEFINED registers

Table 9: Debug Port register map, architecture version 2

5.4.1 The Debug Port ID Register, DPIDR (architecture version 2)

The DPIDR is:

- a read-only register
- accessed by a read of DP register 0
- not affected by the value of SELECT.CTRLSEL.

Figure 13 shows the register bit assignment, with their values for an ARM design.

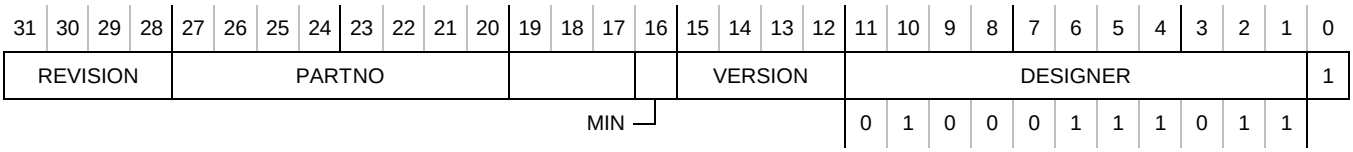

Figure 13: Debug Port ID Register (architecture version 2) bit assignments

REVISION, PARTNO, and DESIGNER, bits [31:16,11:1]

See *The Debug Port ID Register, DPIDR (architecture version 1)* on page 22.

Bits [19:7]

Reserved, UNK.

MIN, bit [16]

See *The Debug Port ID Register, DPIDR (architecture version 1)* on page 22.

VERSION, bits [15:12]

Version of the Debug Port architecture implemented. The valid values for this field are:

0x0 Reserved. Implementations of DP architecture version 0 do not implement DPIDR.

0x1 DP architecture version 1.

0x2 DP architecture version 2.

All other values are reserved.

Bit [0] RAO.

5.4.2 The AP Select Register, SELECT (architecture version 2)

In DP architecture version 2, the SELECT register is:

- a write-only register
- accessed by a write of DP register 0x8.

Figure 14 shows the register bit assignments.

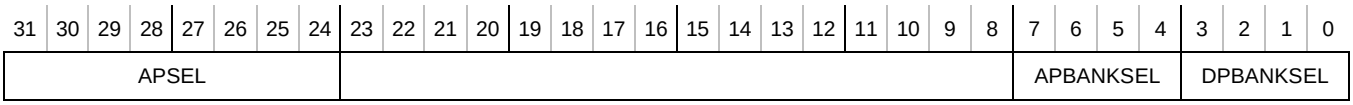

Figure 14: AP Select Register (architecture version 2) bit assignments

APSEL, bits [31:24], and APBANKSEL, bits [7:4]

See the *ARM® Debug Interface v5 Architecture Specification*.

Bits [23:8]

Reserved, UNK/SBZP.

DPBANKSEL, bits [3:0]

Selects the register that appears at DP register 0x4:

0x0 CTRL/STAT, read/write

0x1 DLCR, read/write

0x2 TARGETID, read-only

0x3 DLPIDR, read-only.

All other values are reserved. Writing a reserved value to DPBANKSEL is UNPREDICTABLE.

This field replaces the previously defined DPBANKSEL, bit [0].

5.4.3 The Target Identification Register, TARGETID

The Target Identification Register, TARGETID, is:

- a read-only register
- defined in DP architecture version 2
- accessed by a read of DP register 0x4 when SELECT.DPBANKSEL is set to 0x2.

This register provides information about the target when the host is connected to a single device. Figure 15 shows the register bit assignments.

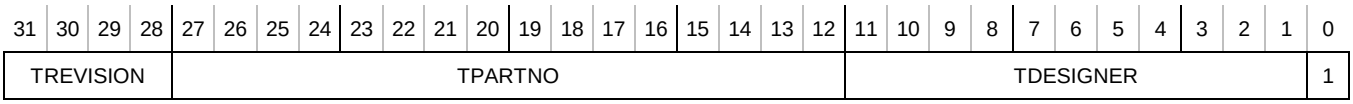

Figure 15: Target Identification Register bit assignments

TREVISION, bits [31:28]

Target revision.

TPARTNO, bits [27:12]

IMPLEMENTATION DEFINED. The value is assigned by the designer of the part and should be unique to that part.

TDESIGNER, bits [11:1]

IMPLEMENTATION DEFINED. An 11-bit code formed from the JEDEC JEP-106 continuation code and identity code. The ID identifies the designer of the part. Designers must insert their own JEDEC-assigned code here.

Note

The ARM Limited JEP-106 value is not shown for this field. Although ARM Limited might design a DP containing the TARGETID register, the designer of *part* is typically another designer who takes that DP and creates a part around it.

If the designer of the part is ARM Limited, then the value of this field is 0x23B.

Bit [0] RAO.

5.4.4 The Data Link Protocol Identification Register, DLPIDR

The Data Link Protocol Identification Register, DLPIDR is:

- a read-only register
- defined in DP architecture version 2
- accessed by a read of DP register 0x4 when SELECT.DPBANKSEL is set to 0x3.

The contents of the Data Link Protocol Identification Register, DLPIDR, are DATA LINK DEFINED.

For Serial Wire, DLPIDR gives Serial Wire protocol version information. Figure 16 shows the register bit assignments for Serial Wire protocol.

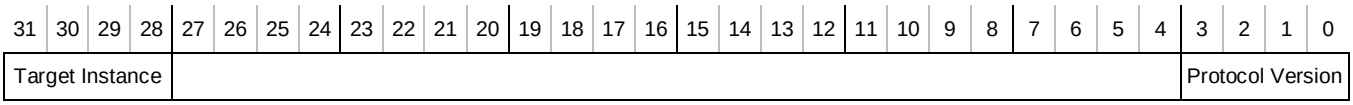

Figure 16: Data Link Protocol Identification Register bit assignments, Serial Wire protocol

Target Instance, bits [31:28]

IMPLEMENTATION DEFINED. Defines an instance number for this device. This value must be unique for all devices with identical TARGETID[28:0] fields that are connected together in a multi-drop system.

Bits [27:4]

Reserved, UNK.

Protocol Version, bits [3:0]

Defines the Serial Wire protocol version. Valid values for this field are:

0x1 Serial Wire protocol version 2. Adds support for multi-drop extensions. See *Serial Wire Protocol Version* 2 on page 40.

All other values are reserved.

Note

A Serial Wire Debug Port which implements DP architecture version 2 must implement at least Serial Wire protocol version 2.

For JTAG-DP, DLPIDR is reserved. Accessing the DLPIDR is UNPREDICTABLE.

6 MULTIPLE PROTOCOL INTEROPERABILITY

6.1 The Serial Wire/JTAG Debug Port (SWJ-DP)

The SWJ-DP provides a mechanism to select between Serial Wire and JTAG Data Link protocols. This enables the JTAG-DP and SW-DP to share pins.

SWJ-DP is a combined JTAG-DP and SW-DP that enables a probe to connect to the target using either the Serial Wire protocol or JTAG. To make efficient use of package pins, the Serial Wire interface shares, or overlays, the JTAG pins, and a mechanism is provided to switch between JTAG-DP and SW-DP, depending on which probe is connected. The SWJ-DP behaves like a JTAG-DP device if normal JTAG sequences are sent to it.

6.1.1 Structure

The SWJ-DP logically consists of a wrapper around the JTAG-DP and SW-DP. Its function is to select JTAG or Serial Wire as the Data Link protocol and enable either JTAG-DP or SW-DP as the interface to the DAP.

Figure 17 below shows such a logical arrangement.

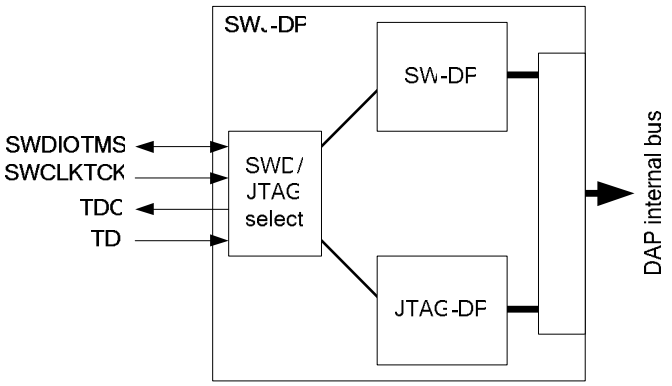

Figure 17: SWJ-DP conceptual model

Notes

- There is no requirement to implement an SWJ-DP as separate JTAG-DP and SW-DP blocks with such a wrapper.
- The JTAG-DP and SW-DP programmers' models do not have to implement the same Debug Port architecture version. See *Debug Port Architecture Versions* on page 19.

6.1.2 Operation

SWJ-DP enables an *Application Specific Integrated Circuit* (ASIC) to be designed which can be used in systems that require either a JTAG interface or a Serial Wire interface. There is a trade-off between the number of pins used and compatibility with existing hardware and test equipment.

There are several scenarios where the use of a JTAG debug interface must be maintained, including:

- to enable inclusion in an existing scan chain, generally on-chip TAPs used for test or other purposes.
- to enable the device to be cascaded with legacy devices which use JTAG for debug; although this can also be supported using a JTAG Access Port (JTAG-AP)
- to enable use of existing debug hardware with the corresponding test TAPs, for example, in *Automatic Test Equipment* (ATE).

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

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

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

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

The connection sequence used for Serial Wire 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** HIGH ensures that all parts of the SWJ-DP are in a known reset state. The pattern used to select Serial Wire has no effect on JTAG devices.

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

6.1.3 Serial Wire and JTAG interface

The external JTAG interface has four mandatory pins, **TCK**, **TMS**, **TDI**, and **TDO**, and an optional reset, **nTRST**. Debug ports also require a separate system reset signal that is asserted, for example, at power-on.

The Serial Wire interface requires two pins:

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

To enable sharing of the connector for either JTAG or Serial Wire, connections must be made external to the SWJ-DP block, as shown in Figure 17. In particular, **TMS** must be a bidirectional pin to support the bidirectional **SWDIO** pin for Serial Wire protocol.

Notes

- When Serial Wire protocol is being used, the JTAG pins **TDI**, **TDO** and **nTRST** are expected to be re-used.
- An SWJ-DP can be implemented in a package where the JTAG pins **TDI**, **TDO** and **nTRST** are not connected. The SWJ-DP is designed to allow selection of Serial Wire protocol without using these JTAG pins.

6.2 Serial Wire and JTAG select mechanism

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

- a first 16-bit sequence is used to switch from JTAG to Serial Wire operation
- a second 16-bit sequence is used to switch from Serial Wire to JTAG.

ARM deprecates use of these sequences on devices where the Dormant state of operation is implemented, and recommends using a transition via Dormant state instead. See *Dormant operation* on page 33.

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

Switching from one protocol to the other can only occur when the selected interface is in its reset state. The JTAG TAP state machine must be in its Test-Logic-Reset state and Serial Wire must be in line-reset. The power-on reset state for a JTAG TAP state machine is the Test-Logic-Reset state.

Having detected a switching sequence, SWJ-DP does not detect further sequences until after a reset condition. If JTAG is selected, the JTAG TAP state machine being in the Test-Logic-Reset state is the reset condition. If Serial Wire is selected, a line reset is the reset condition.

Figure 18 is a simplified state diagram that shows how SWJ-DP transitions between selected, detecting, and selection states.

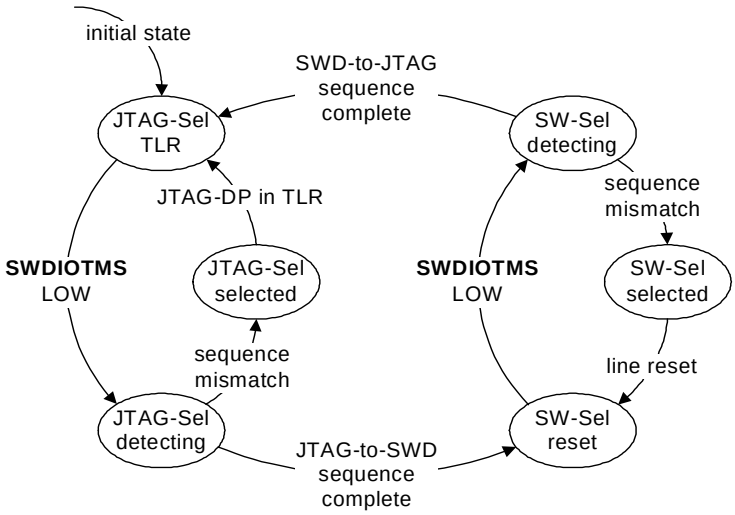

Figure 18: Serial Wire and JTAG select state diagram

Note on Figure 18

The JTAG-to-SWD and SWD-to-JTAG sequences are shown terminating in the *SW-Sel reset* and *JTAG-Sel TLR* states respectively. The recommended sequences end with a reset sequence for the selected state, to ensure the target is in the relevant reset state.

6.2.1 JTAG to Serial Wire switching

To switch SWJ-DP from JTAG to Serial Wire operation:

1. Send at least 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that the current interface is in its reset state. The JTAG interface only detects the 16-bit JTAG-to-SWD sequence starting from the Test-Logic-Reset state.
2. Send the 16-bit JTAG-to-SWD select sequence on **SWDIOTMS**.
3. Send at least 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that if SWJ-DP was already in Serial Wire operation before sending the select sequence, the Serial Wire interface enters line reset state.

The 16-bit JTAG-to-SWD select sequence is 0b0111_1001_1110_0111, most-significant bit (MSB) first. This can be represented as either:

- `0x79E7` transmitted MSB first
- `0xE79E` transmitted least-significant bit (LSB) first.

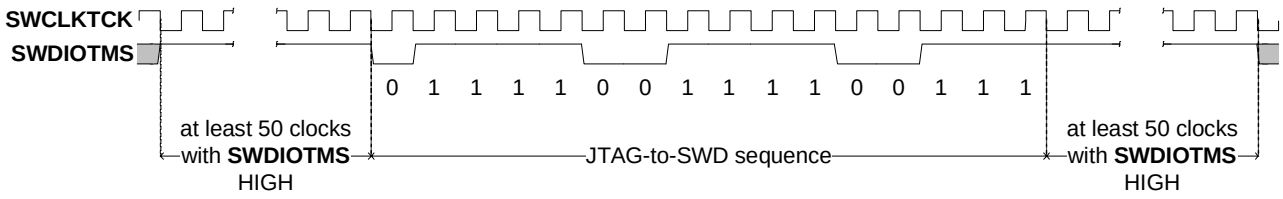

Figure 19: JTAG-to-SWD sequence

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

Note

On selecting Serial Wire operation, the Serial Wire interface is in a reset state. See *Line reset* on page 43. To leave the reset state, the debugger must read the DP DPIDR register.

6.2.2 Serial Wire to JTAG switching

To switch SWJ-DP from Serial Wire to JTAG operation:

1. Send at least 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that the current interface is in its reset state. The Serial Wire interface only detects the 16-bit SWD-to-JTAG sequence when it is in the reset state.
2. Send the 16-bit SWD-to-JTAG select sequence on **SWDIOTMS**.
3. Send at least 5 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures that if SWJ-DP was already in JTAG operation before sending the select sequence, the JTAG TAP enters the Test-Logic-Reset state.

The 16-bit SWD-to-JTAG select sequence is 0b0011_1100_1110_0111, MSB first. This can be represented as either:

- `0x3CE7` transmitted MSB first
- `0xE73C` transmitted LSB first.

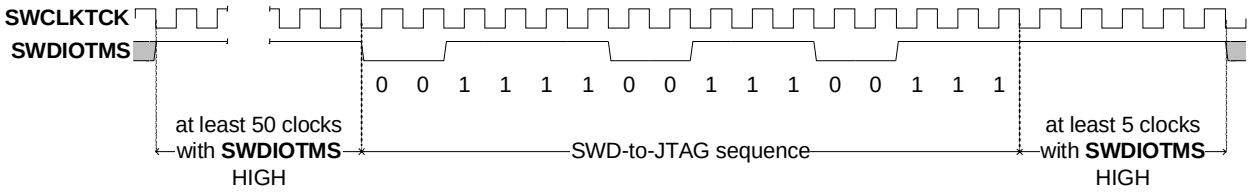

Figure 20: SWD-to-JTAG sequence

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

6.3 Dormant operation

An alternative to the selection mechanism for switching between JTAG and Serial Wire operation described in *Serial Wire and JTAG select mechanism* on page 31 can be implemented using a third state of operation, *Dormant*.

To switch between JTAG and Serial Wire operation, a debugger should first place the target into Dormant state, and then transition to the required operating state.

Using Dormant state allows the target to be placed into a quiescent mode, allowing devices to interoperate with other devices implementing other protocols. Those other protocols must also implement a quiescent state, with a mechanism for entering and leaving that state that is compatible, but not necessarily compliant, with the SWJ-DP and SW-DP protocols.

This third state of operation is required by SWJ-DP and SW-DP implementations that implement Serial Wire protocol version 2. Serial Wire protocol version 2 is described in *Serial Wire Protocol Version* 2 on page 40. Otherwise, support for Dormant state is IMPLEMENTATION DEFINED.

Selection of Dormant state is possible when either JTAG or Serial Wire operation is selected. Figure 21 extends the state diagram of Figure 18 on page 32 to include selection of Dormant state.

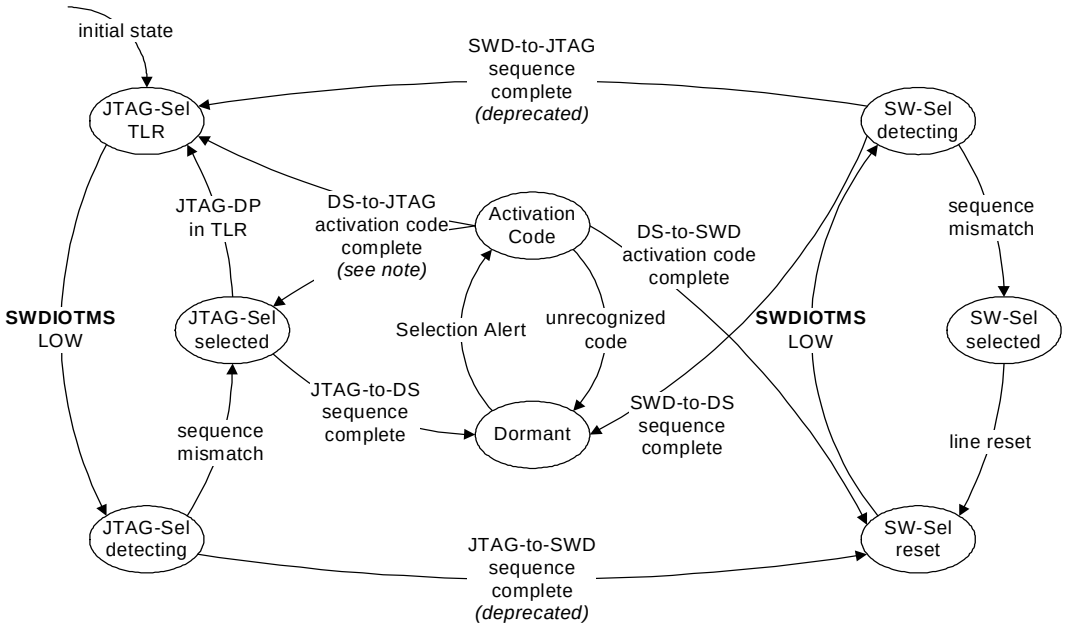

Figure 21: Selection of JTAG, Serial Wire and Dormant states (full SWJ-DP)

Notes on Figure 21

- Following the DS-to-JTAG activation code, the JTAG TAP is in either the Test-Logic-Reset state or Run-Test/Idle state, and hence this state machine is in either the JTAG-Sel TLR state or the JTAG-Sel selected state. Normally, the TAP state returned to is the TAP state left from. However, it is also possible to reset the JTAG TAP state machine when JTAG is not the selected protocol.

  ARM recommends the DS-to-JTAG sequence is followed by a single clock with **SWDIOTMS** LOW to ensure the TAP is in the Run-Test/Idle state.
- The DS-to-SWD sequence is shown terminating in the *SW-Sel reset* state. The recommended sequence ends with a line reset to ensure the target is in the reset state.

6.3.1 Use of Dormant state other than in SWJ-DP

A Serial Wire device that does not implement JTAG can nevertheless implement Dormant state, and interoperate with SWJ-DP and other JTAG devices that also implement Dormant state.

This allows multi-drop SWJ-DP, SW-DP and JTAG TAPs to share a physical connection to a host, as shown in Figure 22. These different devices may be in different physical packages, or on different dies in a single package, or on a single die.

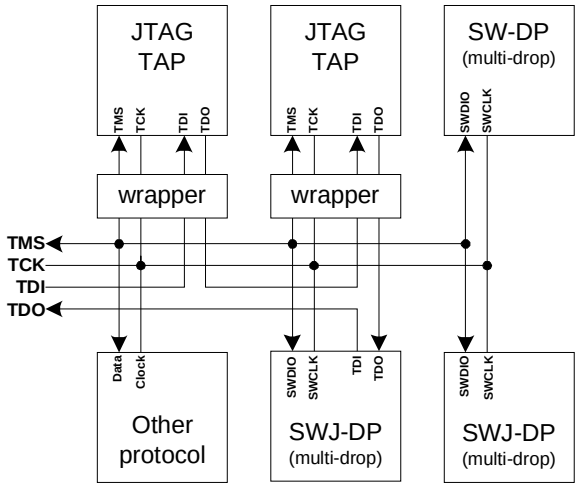

Figure 22: Multiple JTAG, SW, SWJ (multi-drop) and other protocol devices on shared connection

6.3.2 JTAG to Dormant switching

To switch from JTAG to Dormant a debugger should:

1. Send at least 5 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This places the JTAG TAP state machine into the Test-Logic-Reset state, and selects the IDCODE instruction.
2. Send the recommended 31-bit JTAG-to-DS select sequence on **SWDIOTMS**.

The recommended 31-bit JTAG-to-DS select sequence is 0b010_1110_1110_1110_1110_1110_1110_0110, MSB first. This can be represented as either:

- `0x2EEEEEE6` transmitted MSB first (that is, starting from bit 30)
- `0x33BBBBBA` transmitted LSB first.

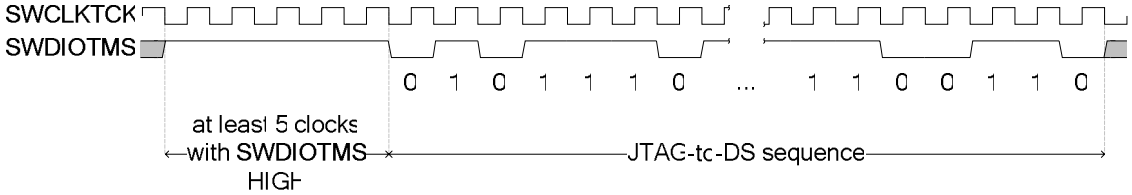

Figure 23: Recommended JTAG-to-DS sequence

Requirements for implementations

The JTAG-to-DS sequence is the shortest sequence that will switch from JTAG-to-DS. For compatibility with other standards, all JTAG devices that implement Dormant state must recognize other sequences as valid JTAG-to-DS select sequences.

The full sequence is defined around the concept of a *zero-bit-DR-scan* (ZBS or ZBS scan) which is in turn defined by transitions of the JTAG TAP state machine. A ZBS is defined as any JTAG TAP state machine sequence that starts at Capture-DR and ends in Update-DR without passing through Shift-DR.

Examples of a ZBS are:

- Capture-DR → Exit1-DR → Update-DR
- Capture-DR → Exit1-DR → Pause-DR → … → Pause-DR → Exit2-DR → Update-DR

The sequence also uses the *ZBS count*, which is defined as follows:

- The ZBS count is unlocked and reset to zero if the TAP state machine enters either the Select-IR-Scan or Test-Logic-Reset state. This includes asynchronously entering Test-Logic-Reset following assertion of **nTRST**. At reset, the ZBS count is unlocked and reset to zero.
- On entering Update-DR at the end of a ZBS scan, if the ZBS count is unlocked and less than seven, it is incremented by one.
- The counter does not increment past seven. On entering Update-DR at the end of a ZBS scan, if the ZBS count is unlocked and equal to seven, it is not incremented. The count does not wrap to zero.
- The ZBS count is locked if the TAP state machine enters the Shift-DR state and the ZBS count is not zero.

The JTAG-to-DS sequence is defined as any sequence of TAP state machine transitions that terminates in the Run-Test/Idle state with a locked ZBS count of six. On entering Run-Test/Idle, the target is placed into Dormant state.

The behavior of the target on entering Run-Test/Idle with other locked ZBS counts is IMPLEMENTATION DEFINED.

Although the recommended JTAG-to-DS sequence starts by placing the JTAG TAP state machine in the Test-Logic-Reset state, this is not a requirement for recognizing the JTAG-to-DS sequence. Tools must, however, ensure the Instruction Register (IR) is loaded with either the BYPASS or IDCODE instruction before placing the target into the Dormant state. If the IR is not loaded with either of these instructions when the target is put into Dormant state, the behavior is UNPREDICTABLE.

The pseudocode function `EnterDormantState` describes the function of the JTAG-to-DS sequence detector. It is notionally called on every TAP state machine transition. The function's argument is the state being entered, and the function's result is a Boolean indicating whether Dormant state should be entered.

For details of the pseudocode language, see the *ARM® Architecture Reference Manual ARM®v7-A and ARM®v7-R edition*.

```
enumeration TAPState {
    TestLogicReset, RunTestIdle,
    SelectDRScan, CaptureDR, ShiftDR, Exit1DR, PauseDR, Exit2DR, UpdateDR,
    SelectIRScan, CaptureIR, ShiftIR, Exit1IR, PauseIR, Exit2IR, UpdateIR};

boolean shiftDRflag = FALSE;
integer ZBScount = 0;
boolean ZBSlocked = FALSE;

// EnterDormantState
// =================

boolean EnterDormantState(TAPState state)

    case state of
        when CaptureDR:  shiftDRflag = FALSE;

        when ShiftDR:
            shiftDRflag = TRUE;
            if ZBScount != 0 then ZBSlocked = TRUE;

        when UpdateDR:
            if !ZBSlocked && !shiftDRflag && ZBScount < 7 then
               ZBScount = ZBScount + 1;

        when SelectIRScan, TestLogicReset:
            ZBScount = 0;  ZBSlocked = FALSE;

    return (state == RunTestIdle && ZBSlocked && ZBScount == 6);
```


Note

If the JTAG-to-DS sequence is terminated by a entering the Test-Logic-Reset state, an SWJ-DP can immediately detect a JTAG-to-SWD sequence.

6.3.3 Serial Wire to Dormant switching

To switch from SW to Dormant:

1. Send at least 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This ensures the Serial Wire interface is in the reset state. The target only detects the SWD-to-DS sequence when it is in the reset state.
2. Send the 16-bit SWD-to-DS select sequence on **SWDIOTMS**.

The 16-bit SWD-to-DS select sequence is 0b0011_1101_1100_0111, MSB first. This can be represented as either:

- 0x3DC7 transmitted MSB first
- 0xE3BC transmitted LSB first.

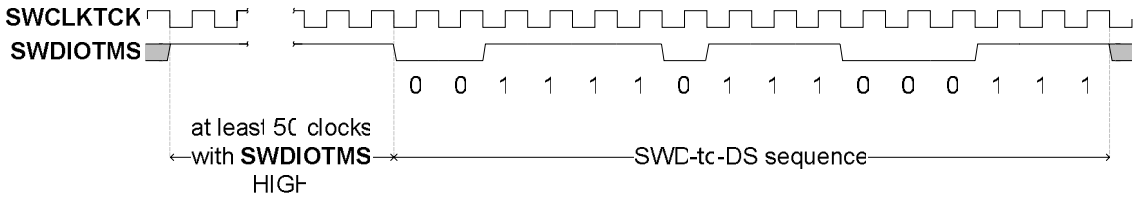

Figure 24: SWD-to-DS sequence

6.3.4 Switching out of Dormant state

The sequence to switch out of Dormant state is considerably longer to avoid it being generated accidentally by whichever alternative protocol is in use. The sequence is long enough that it is statistically highly improbable that it will be generated any other way.

To switch out of Dormant state:

1. Send at least 8 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This is to ensure the target is not in the middle of detecting a Selection Alert sequence. The target is permitted to detect the Selection Alert sequence even if this 8-cycle sequence is not present.
2. Send the 128-bit Selection Alert sequence on **SWDIOTMS**.
3. Send 4 **SWCLKTCK** cycles with **SWDIOTMS** LOW. The target is permitted to ignore the value on **SWDIOTMS** during these cycles.
4. Send the required *activation code* sequence on **SWDIOTMS**.
5. Send a sequence to place the target into a known state:
   - If selecting JTAG, send one **SWCLKTCK** cycle with **SWDIOTMS** LOW. This is to ensure that the TAP state machine is in the Run-Test/Idle state. Alternatively, send at least 5 SWCLKTCK cycles with SWDIOTMS HIGH to ensure the TAP state machine is in the Test-Logic/Reset state.
   - If selecting Serial Wire, send at least 50 **SWCLKTCK** cycles with **SWDIOTMS** HIGH. This is to ensure the Serial Wire interface is in the line reset state.

The Selection Alert sequence is 0b0100_1001_1100_1111_1001_0000_0100_0110_1010_1001_1011_0100_1010_0001_0110_0001_1001_0111_1111_0101_1011_1011_1100_0111_0100_0101_0111_0000_0011_1101_1001_1000, MSB first. This can be represented as either:

- 0x49CF9046 A9B4A161 97F5BBC7 45703D98 transmitted MSB first
- 0x19BC0EA2 E3DDAFE9 86852D95 6209F392 transmitted LSB first.

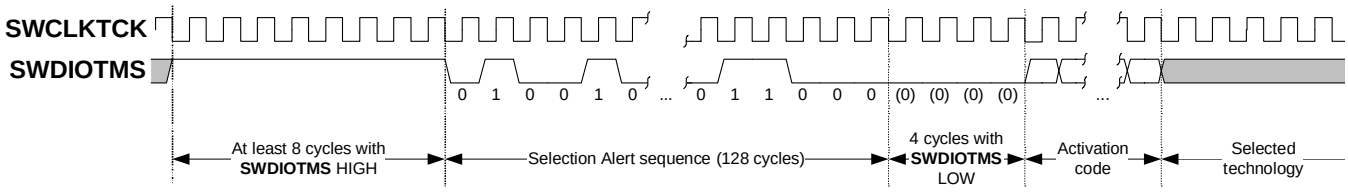

Figure 25: Selection Alert sequence

Note

The Selection Alert sequence can be generated by implementing a Linear Feedback Shift Register (LFSR) implementing feedback on bits 6, 5, 3 and 0, starting in the state 0b1001001 and shifting out one bit from bit 0 each cycle. The sequence starts with a zero start bit and continues with the output of the LFSR.

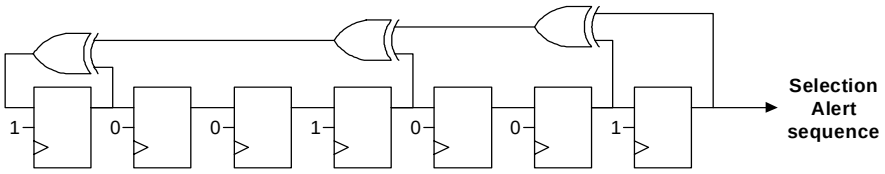

Figure 26: LFSR for generating Selection Alert sequence

The value of the activation code depends on whether Serial Wire and or JTAG operation is to be requested. Table 10 defines the recommended activation codes a debugger should use for JTAG devices, SW-DP devices and SWJ-DP devices. These sequences are sent MSB first.

Activation code	Value (MSB first)	Devices activated				Protocol selected
		Other JTAG	ADIv5 Debug Ports			
			JTAG	SW	SWJ	
JTAG-Serial	0b0000_0000_0000	Yes	Yes	No	Yes	JTAG
ARM CoreSight Serial Wire	0b0101_1000	No	No	Yes	Yes	Serial Wire
ARM CoreSight JTAG-DP	0b0101_0000	No	Yes	No	Yes	JTAG

Table 10: Activation codes

JTAG online activation codes

For compatibility with other standards, all JTAG devices that implement Dormant state using the ADIv5 defined *selection alert sequence*, must recognize other sequences as valid JTAG-Serial activation codes.

Figure 27 shows, as a state diagram, the sequence that a JTAG device must recognize.

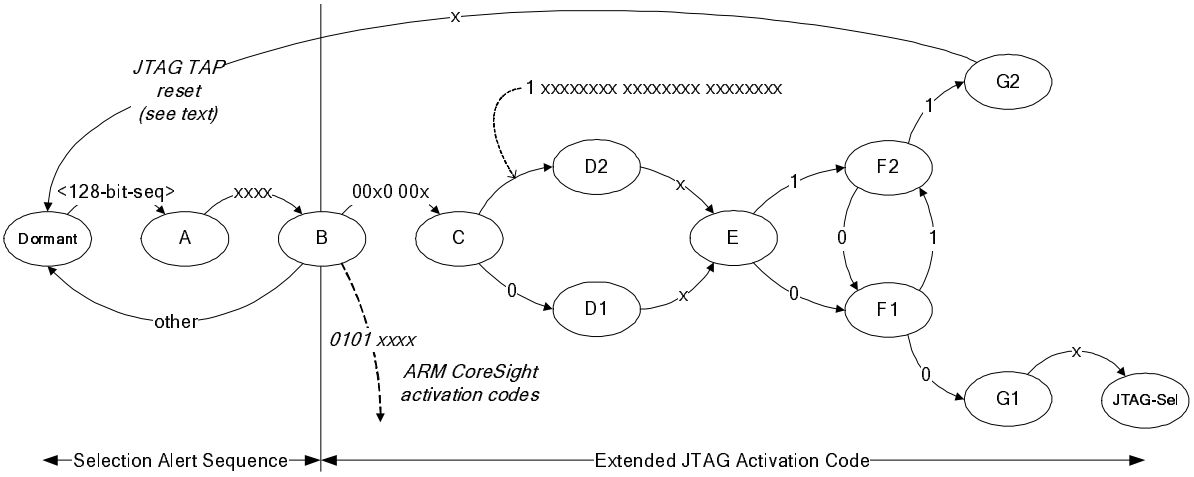

Figure 27: Dormant to JTAG state diagram

Each of the bit-strings shown in Figure 27 are received MSB first. The transition out of state G2 requires a reset of the JTAG TAP, but otherwise returns to Dormant state. For more information on this sequence, contact ARM.

ARM recommends that debuggers use the activation code sequence shown in Table 10.

ARM CoreSight activation codes

All activation codes starting 0b0101 are reserved for use by ARM CoreSight protocols, and must be followed by a 4-bit *protocol selection code*. The two protocol selection codes 0b0000 and 0b1000 are as defined by Table 10. Any other protocol selection code must cause a return to Dormant state.

Note

ADIv5 does not define any other activation codes, but also does not prohibit an implementation from recognizing other activation codes for compatibility with other standards. Implementations may also use alternative selection alert mechanisms. Debuggers can generate multiple selection alert sequences to alert multiple devices, and then use the common activation codes to select which devices to activate.

6.4 Restriction on switching

A debugger must not mix JTAG-DP and SW-DP reads and writes of DAP registers in a single debug session. A single debug session is defined as from when a debugger connection is made with the system in a reset state through to the debugger connection being broken. At the start of a debug session, the state of the target is essentially UNKNOWN.

Attempting to mix JTAG-DP and SW-DP reads and writes of DAP registers while any component of the DAP is active can have UNPREDICTABLE results.

A power-on reset cycle might be required to reset the DAP before a change in active Data Link protocol. However, this is not required when switching between the active protocol and Dormant state.

7 SERIAL WIRE PROTOCOL VERSION 2

7.1 Introduction to multi-drop

Serial Wire protocol version 2 extends the Serial Wire protocol by the addition of multi-drop capability. Multi-drop capability allows more than one Serial Wire interface to share the same connection to a debugger host.

This section describes the multi-drop extensions.

7.2 Limitations of multi-drop

7.2.1 System configuration

Each device must be configured with a unique target ID, which includes a 4-bit instance ID to differentiate between otherwise identical devices. This places a limit of 16 such targets in any system, and means that identical devices must be configured before they are connected together to ensure that their instance IDs do not conflict.

7.2.2 Auto-detection

It is not possible to interrogate a multi-drop Serial Wire system to establish which devices are connected – no communication with a device is possible without prior selection of that target using its target ID. Therefore connection to a multi-drop Serial Wire system requires that either:

- The host is configured with knowledge of the devices in the system before connection.
- The host attempts auto-detection by issuing a target select command for each of the devices it has been configured to support. While this is likely to involve a large number of target select commands, it should be possible to iterate through all the supported devices in a reasonable time from the point of view of a human user of the debug tools.

The practical implications of this restriction are that debug tools will not be able to seamlessly connect to targets in a multi-drop Serial Wire system that they have never seen before. However, if the debug tools can be provided with the target ID of such targets by the user then the contents of the target can be auto-detected as normal.

This limitation allows for significant savings in design complexity.

7.3 Target selection protocol

A host selects a new target by doing the following:

1. Perform a line reset.
2. Write to DP register 0xC, TARGETSEL, where the data indicates the selected target. The target response must be ignored.
3. Read from the DP register 0x0, DPIDR, to verify that the target has been successfully selected. A write to the TARGETSEL register must always be followed by a read of the DPIDR register or a line reset. If the response is incorrect or there is no response, the host must start the sequence again.

In Serial Wire protocol version 1, the host can attempt a read of DPIDR to recover from a protocol error. With Serial Wire protocol version 2, the host must reselect the target, including performing a line reset.

Note

A line reset will cause the STICKYORUN flag in the DP CTRL/STAT register to be set to 1 if overrun detection is enabled, that is, if the ORUNDETECT bit is set to 1. The host must not switch targets when the ORUNDETECT bit is set to 1.

The target is selected on receiving a line reset sequence.

If, following a line reset sequence, the target receives a write to the TARGETSEL which does not select the target, the target is deselected. When deselected, the target ignores all accesses and must not drive the line. Only writes to TARGETSEL immediately following a line reset sequence can select or deselect the target. Writes to TARGETSEL at any other time are UNPREDICTABLE.

If the target encounters a protocol error at any time, it becomes deselected. It particular, it will not respond to a read of the DPIDR register.

The target does not drive the wire during the response phase of the write to the TARGETSEL register. That is, it provides the protocol error response. This prevents the wire from being driven in different directions by different devices, as multiple devices might be selected at this point.

A parity error in the data phase of a write to the TARGETSEL register does not cause the WDATAERR flag to be set to 1. Parity errors in the write phase of a TARGETSEL write are treated as protocol errors.

Accesses to the TARGETSEL register are not affected by the state of the CTRL/STAT WDATAERR, STICKYERR, STICKYCMP or STICKYORUN flags.

Implementations of Serial Wire protocol version 2 must also support Dormant operation. See *Dormant operation* on page 33.

7.4 Programmer's model

7.4.1 Target Selection Register, TARGETSEL

The Target Selection Register, TARGETSEL, is:

- a DATA LINK DEFINED write-only register
- defined by the Serial Wire protocol version 2
- accessed by a write of DP register 0xC
- previously reserved and named ROUTESEL.

On a write to TARGETSEL following a line reset sequence, the target is selected if bits [27:0] match TARGETID[27:0] and bits [31:28] match DLPIDR[31:28]. Figure 28 shows the register bit assignments.

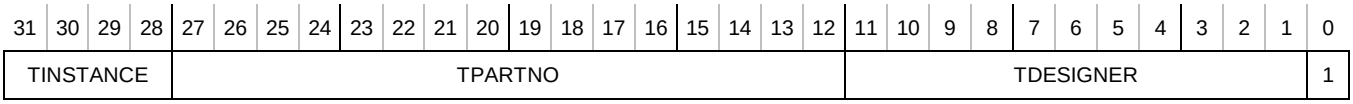

Figure 28: Target Selection Register bit assignments

For a description of these fields, see *The Target Identification Register, TARGETID* on page 28 and *The Data Link Protocol Identification Register, DLPIDR* on page 29.

Writing the value with all bits set deselects all targets. Writing any other value with bits [11:8] set to 0b1111 or 0b0000, or with bit [0] set to 0b0 is UNPREDICTABLE.

Accesses to TARGETSEL are treated differently to accesses to other DP registers. For more information see *Target selection protocol* on page 40.

8 APPENDIX: SERIAL WIRE PROTOCOL

8.1 Introduction

This section describes the synchronous Serial Wire protocol.

Serial Wire operates with a synchronous serial interface. This uses a single bidirectional data signal, and a clock signal.

8.2 Clocking

The Serial Wire interface clock can be asynchronous to any system clock, including the debug logic clock. The Serial Wire interface clock 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 transactions. This ensures that the transaction can be clocked through the Serial Wire interface. This means that after the data phase of transactions the host must do one of the following:

- immediately start a new transaction
- continue to clock the Serial Wire interface until the host starts a new transaction, inserting idle cycles
- after clocking out the data parity bit, continue to clock the Serial Wire interface inserting idle cycles until it has clocked out at least 8 more clock rising edges, before stopping the clock.

See also *Idle cycles* on page 43.

8.3 Overview of Serial Wire interface

This section gives an overview of the physical Serial Wire interface.

8.3.1 Line interface

The Serial Wire interface uses a single bidirectional data pin, **SWDIO**. That is, the same signal is used for both host and target sourced signals. The host emulator drives the protocol timing: only the host emulator generates packet headers.

The Serial Wire interface is synchronous, and requires a clock pin, **SWCLK**.

Synchronous operation uses a clock reference signal, which can be sourced from the target and exported, or provided by the host. 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 over-sampling.

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

8.3.2 Line pull-up

So that the line can be assumed to be in a known state when no interface is driving the line, a 100kΩ pull-up 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 pull-up resistor eventually brings the line to the HIGH state, but this takes many bit periods.

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

Note

A small current drains from the target whenever the line is driven LOW. 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 is activated.

8.3.3 Line turn-round

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

8.3.4 Idle cycles

Following transactions, the host must either insert idle cycles or continue immediately with the start bit of a new transaction. The host clocks the Serial Wire interface with the line LOW to insert idle cycles.

8.3.5 Protocol errors

If the Serial Wire interface detects a protocol error in a packet header it enters the protocol error state. A protocol error is one of:

- the Parity bit does not match the parity of the packet header
- the Stop bit is not 0
- the Park bit is not 1.

In the protocol error state, the interface leaves the line not driven and waits for the host to re-try with a new header, normally after a single idle cycle.

If overrun detection is enabled, the interface must wait until the data phase of the transaction has completed before entering the protocol error state.

It is IMPLEMENTATION DEFINED whether the interface can leave the protocol error state on a read of the DP DPIDR register. The interface can always leave the protocol error state on a line reset.

When in protocol error state, if the interface detects a valid packet header other than the read of the DP DPIDR register, or the interface detects an IMPLEMENTATION DEFINED number of further protocol errors, it enters the lockout state. ARM recommends that the interface locks out after one further protocol error in the protocol error state.

The Serial Wire interface only leaves the lockout state on a line reset. (If the interface cannot leave the protocol error state on a read of the DP DPIDR register then the protocol error and line reset states are equivalent.)

If the Serial Wire interface implements Serial Wire protocol version 2, it must lock out after a single bad data sequence immediately following line reset, and the host should attempt the line reset immediately after the unexpected response. However, if the first sequence detected by the Serial Wire interface following line reset is valid it may then revert to only locking out after multiple bad data sequences.

If the host does not see an expected response from the Serial Wire interface, it should leave the line not driven for at least the length of any potential data phase and then attempt a line reset. The host may attempt reads of the DP DPIDR register before attempting line reset, as the target may respond and leave the protocol error state as described above, but this approach is not recommended.

8.3.6 Line reset

The Serial Wire interface does not include a reset signal. A line reset is achieved by holding the data signal HIGH for at least 50 clock cycles, followed by at least one idle cycle.

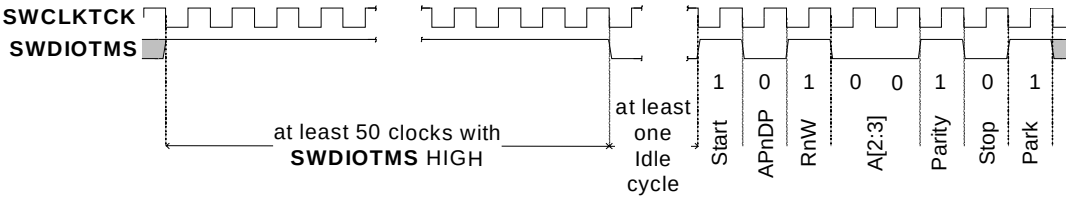

Figure 29: Line reset sequence, followed by read of DP DPIDR register

A line reset is required when first connecting to the target. A line reset is also required following an error. See *Protocol errors* on page 43.

When waiting for a packet header, if the Serial Wire interface detects a sequence of 50 clock cycles with the data signal held HIGH followed by at least one idle cycle, it must enter the reset state. It is IMPLEMENTATION DEFINED whether a sequence of 50 clock cycles with the data signal held HIGH detected at any other time causes the interface to enter the reset state.

The interface also enters reset state following a Serial Wire select sequence, as defined by *Multiple Protocol Interoperability* on page 30.

The only valid transactions in reset state are:

- A read of the DP DPIDR register. This takes the connection out of reset state.
- One of the switching sequences defined by *Multiple Protocol Interoperability* on page 30.
- A write to the DP TARGETSEL register. If this selects the target, the interface remains in reset state.

Note

Only writes to TARGETSEL immediately after entry to the reset state can select or deselect the target. See *Target selection protocol* on page 40 .

Any of these sequences can be aborted by a second line reset. The behavior of the target is UNPREDICTABLE if any other transaction is made in reset state.

If the host does not see an expected response from reading the DPIDR register, it must retry the reset sequence. This is because the interface may have been in a state where, for example, it treated the initial line reset as a data phase of a transaction and so did not detect it as a valid line reset. If this is the case, the interface will have detected the line reset as a protocol error and require a second line reset to correctly respond.

9 APPENDIX: STANDARD MEMORY ACCESS PORT DEFINITIONS

9.1 Introduction

The Memory Access Port (MEM-AP) programmer's model contains some IMPLEMENTATION DEFINED features. This appendix provides reference implementation options for implementers and users of MEM-APs when connecting to standard memory interfaces. In particular, it provides the recommended interpretations of the Prot, AddrInc, SPIDEN and Size fields of the CSW register.

9.2 AMBA AHB

For more information, see *AMBA™ Specification (Rev 2.0)*, and *AMBA® 3 AHB-Lite Protocol Specification*. For AMBA AHB implementations, the CSW register is implemented as follows:

Prot, bits [30:24]

The CSW.Prot field drives the AHB **HPROT** signals. The reset value of CSW.Prot is 0b0000011. The settings for the CSW.Prot field are:

Bit [30]

Reserved, SBO. If this bit is written as 0 the behavior of AHB-AP transactions is UNPREDICTABLE.

MasterType, bit [29]

Master Type bit. MasterType allows the AHB-AP to mimic a second AHB master by driving a different value on **HMASTER[3:0]**. Support for this function is IMPLEMENTATION DEFINED.

1 Drive **HMASTER[3:0]** with the bus master ID for the AHB-AP.

0 Drive **HMASTER[3:0]** with the bus master ID for the second bus master.

If this function is not implemented the bit is RAZ/WI.

HPROT[4], Allocate, bit [28]

Drives **HPROT[4]**, Allocate. **HPROT[4]** is an ARMv6 extension to AHB. For further information, see *ARM1136JF-S™ and ARM1136J-S™ Technical Reference Manual*.

If the AHB master interface does not support the ARMv6 extension to AHB, this bit is RAZ/WI.

HPROT[3:0], bits [27:24]

Drives **HPROT[3:0]**. See Table 11. Support for each **HPROT** signal in the AHB master interface is IMPLEMENTATION DEFINED.

Bit	HPROT signal	Description	Description if not implemented at the AHB master interface
27	**HPROT[3]**	Cacheable	RAZ/WI
26	**HPROT[2]**	Bufferable	RAZ/WI
25	**HPROT[1]**	Privileged	RAO/WI
24	**HPROT[0]**	Data	RAO/WI

Table 11: CSW.Prot to HPROT mapping

SPIDEN, bit [23]

It is IMPLEMENTATION DEFINED whether the CSW.SPIDEN bit reflects the state of the CoreSight authentication signal, **SPIDEN**. Otherwise, the CSW.SPIDEN bit is RAZ.

Note

AMBA AHB does not support Security Extensions.

AddrInc, bits [5:4]

Support for the *Increment Packed* mode of transfer is IMPLEMENTATION DEFINED. See *Required support of Memory Access Port (MEM-AP) packed* transfers on page 16.

Size, bits[3:0]

CSW.Size must support word, half-word and byte sized accesses.

9.3 AMBA APB

For more information, see *AMBA™ Specification (Rev 2.0)*, and *AMBA™ 3 APB Protocol Specification*. For AMBA APB implementations, the CSW register is implemented as follows:

Prot, bits [30:24]

Reserved, UNK/SBZP.

SPIDEN, bit [23]

Reserved, UNK.

AddrInc, bits [5:4]

CSW.AddrInc does not support the *Increment Packed* mode of transfer. See *Required support of Memory Access Port (MEM-AP) packed* transfers on page 16.

Size, bits [3:0]

CSW.Size only supports word accesses, and reads as 0b010. Writes to CSW.Size are ignored.

10 APPENDIX: CROSS-OVER WITH THE ARM® ARCHITECTURE

10.1 Introduction

The ARM Debug Interface v5 is the recommended external debug interface for ARMv6-M and all ARMv7 architecture profiles. This section describes the recommended or required options for each variant.

When designing with ARM Cortex™ processor cores and ARM CoreSight™ Design Kits, the choice of Debug Access Port (DAP) features might be at the discretion of the system designer. ARM recommends that system designers choose a DAP that implements all the recommended features for all ARM architecture processors contained in the design.

ADIv5 might also be used with other architecture variants. For example, an ADIv5 JTAG Access Port (JTAG-AP) might be used to access a Debug Test Access Port (DBGTAP), as defined by ARM Debug Interface v4 (ADIv4) for ARMv6 architecture processors.

10.2 ARMv6-M

ARMv6-M requires an ADIv5-compliant DAP.

ARM recommends that the Debug Port (DP) implements the Serial Wire interface, either through a SW-DP or SWJ-DP. A JTAG-DP is permitted. ARM recommends that the DP implements the MINDP model.

There must be one MEM-AP per processor. That MEM-AP must be capable of addressing the complete memory space visible to the processor, including all debug peripherals and the NVIC. The MEM-AP must support byte, half-word and word accesses to memory. The MEM-AP is not required to support the packed increment transfer mode.

Other Access Ports (APs) may be connected to the DAP.

10.3 ARMv7-M

ARMv7-M requires an ADIv5-compliant DAP.

ARM recommends that the DP implements the Serial Wire interface, either through a SW-DP or SWJ-DP. A JTAG-DP is permitted. ARM recommends that the DP does not implement the MINDP model.

There must be one MEM-AP per processor. That MEM-AP must be capable of addressing the complete memory space visible to the processor, including all debug peripherals and the NVIC. The MEM-AP must support byte, half-word and word accesses to memory. ARM recommends that the MEM-AP does support the packed increment transfer mode.

Other APs may be connected to the DAP.

10.4 ARMv7-A and ARMv7-R

ARMv7-A and ARMv7-R do not require an ADIv5-compliant DAP. Although the ADIv5 interface is not required for compliance with ARMv7, the ARM RealView® tools require this interface to be implemented.

Where an ADIv5-compliant DAP is implemented, ARM recommends that the DP implements the JTAG and Serial Wire interfaces through an SWJ-DP. ARM recommends that the DP does not implement the MINDP model.

Many processors can be connected to a single MEM-AP. That MEM-AP need only be capable of addressing the debug peripherals of those processors. If the MEM-AP can address only debug peripherals, it is only required to support word accesses to memory, and therefore does not need to support the packed increment transfer mode.

ARM recommends that debug implementations include a MEM-AP that can address the complete memory space visible to the processor or processors. This is typically a second MEM-AP in the DAP. ARM recommends that a

MEM-AP that can access the complete memory space supports byte, half-word and word accesses to memory. ARM recommends that this MEM-AP does support the packed increment transfer mode.

Other APs may also be connected to the DAP.

10.5 Summary

Table 12 summarizes the recommended and required elements of an ADIv5 implementation for each of the ARM architecture variants for which ADIv5 is the required or recommended DAP.

	ARMv6-M	**ARMv7-M**	**ARMv7-R and ARMv7-A**
ADIv5-compliant DAP	Required	Required	Recommended
Debug Port elements:			
JTAG-DP	Permitted	Permitted	Permitted
SW-DP	-	-	Permitted
SWJ-DP	-	-	Recommended
SWJ-DP or SW-DP	Recommended	Recommended	-
Not MINDP	Permitted	Recommended	Recommended
Memory Access Port elements:			
One MEM-AP per processor	Required	Required	Permitted
MEM-AP access to system memory	Required	Required	Permitted
MEM-AP support for 8-bit and 16-bit accesses	Required	Required	Required only if system access is supported
MEM-AP support for 32-bit accesses	Required	Required	Required
MEM-AP support for packed increment transfers	Permitted	Recommended	Recommended

Table 12: Recommended ADIv5 implementations for ARM® Architecture variants

