CSWP Protocol Specification

DSG

Debug and trace tools

Document number:	ARM-ECM-0667760	Version:	1.2
Date of Issue:	02/11/2018		

© Copyright Arm Limited 2018. All rights reserved.

Abstract

This document describes the CSWP protocol used to communicate between debug tools and on-target debug agents for debug and trace over functional I/O.

Keywords

CSWP debug tools functional I/O USB

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at **http://www.arm.com/company/policies/trademarks**.

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-20349

Contents

1 ABOUT THIS DOCUMENT 6

1.1 Change control 6

1.2 Current status and anticipated changes 6

1.3 Change history 6

1.4 References 6

1.5 Terms and abbreviations 6

2 SCOPE 6

3 INTRODUCTION 7

4 PROTOCOL OVERVIEW 9

4.1 Devices 9

4.2 Device discovery 9

4.3 Device configuration 10

4.4 Connecting and disconnecting 10

4.5 Registers 12

4.6 Batch operations 12

5 PROTOCOL VERSIONS 13

6 PROTOCOL DESCRIPTION 14

6.1 Field encoding 14
6.1.1 Variable length integer (varint) 14
6.1.2 Strings 14

6.2 Message Header 15
6.2.1 Request 15
6.2.2 Response 15

6.3 Error codes 16

6.4 Messages 17
6.4.1 CSWP_INIT 18
6.4.2 CSWP_TERM 19
6.4.3 CSWP_CLIENT_INFO 20
6.4.4 CSWP_ASYNC_MESSAGE 21

6.4.5 CSWP_SET_DEVICES 22
6.4.6 CSWP_GET_DEVICES 23
6.4.7 CSWP_GET_SYSTEM_DESCRIPTION 24
6.4.9 CSWP_DEV_OPEN 25
6.4.10 CSWP_DEV_CLOSE 26
6.4.11 CSWP_SET_CONFIG 27
6.4.12 CSWP_GET_CONFIG 28
6.4.13 CSWP_GET_DEVICE_CAPABILITIES 29

6.5 Register Commands 30
6.5.1 CSWP_REG_LIST 30
6.5.2 CSWP_REG_READ 31
6.5.3 CSWP_REG_WRITE 32

6.6 Memory commands 33
6.6.1 CSWP_MEM_READ 34
6.6.2 CSWP_MEM_WRITE 35
6.6.3 CSWP_MEM_POLL 36

7 SUPPORTED DEVICES 37

7.1 mem-ap.v2 37
7.1.1 Configuration items 37
7.1.2 Registers 37
7.1.3 Memory 38
7.1.4 Child device discovery 39

7.2 mem-ap.v1 39
7.2.1 Configuration items 39
7.2.2 Registers 39
7.2.3 Memory 39
7.2.4 Child device discovery 40

7.3 memory 40
7.3.1 Configuration items 40
7.3.2 Registers 40
7.3.3 Memory 40
7.3.4 Child device discovery 40

7.4 dap.v6 41
7.4.1 Configuration items 41
7.4.2 Registers 41
7.4.3 Memory 42
7.4.4 Child device discovery 42

7.5 dap.v5 42
7.5.1 Configuration items 42
7.5.2 Registers 42
7.5.3 Memory 43
7.5.4 Child device discovery 43

7.6 jtag-ap 43
7.6.1 Configuration items 43
7.6.2 Registers 44

7.6.3 Memory 44
7.6.4 Child device discovery 44

7.7 cscomp 44
7.7.1 Configuration items 44
7.7.2 Registers 44
7.7.3 Memory 44

7.8 linux.cscomp 44
7.8.1 Configuration items 44
7.8.2 Registers 45

8 EXAMPLE SESSIONS 45

8.1 Enabling STM trace via kernel driver 45

9 USB TRANSPORT 47

10 USB TRACE 48

10.1 Setup and configuration 48

10.2 Data channel 50

10.3 Status channel 50

1 ABOUT THIS DOCUMENT

1.1 Change control

1.2 Current status and anticipated changes

Initial version for partner review

1.3 Change history

[Comments]

1.4 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
[ADIv6]	IHI 0074A	ARM	ARM Debug Interface Architecture Specification ADIv6.0
[ADIv5]	IHI 0031D	ARM	ARM Debug Interface Architecture Specification ADIv5.0 to ADIv5.2
[CSARCH]	IHI 0029E	ARM	ARM CoreSight Architecture Specification

1.5 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning

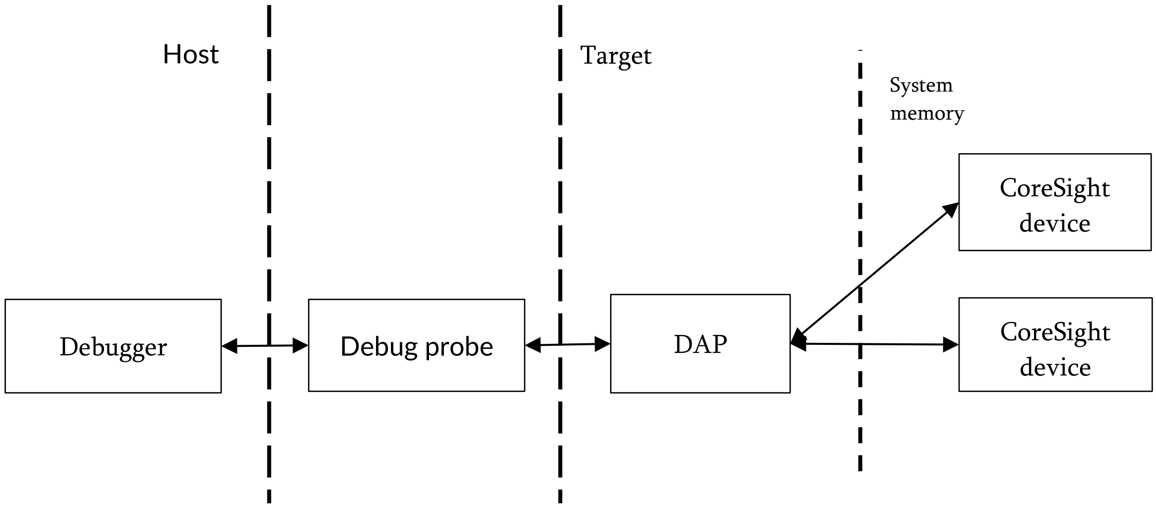
2 SCOPE

This document describes the CSWP protocol used to communicate between debug tools and on-target debug agents.

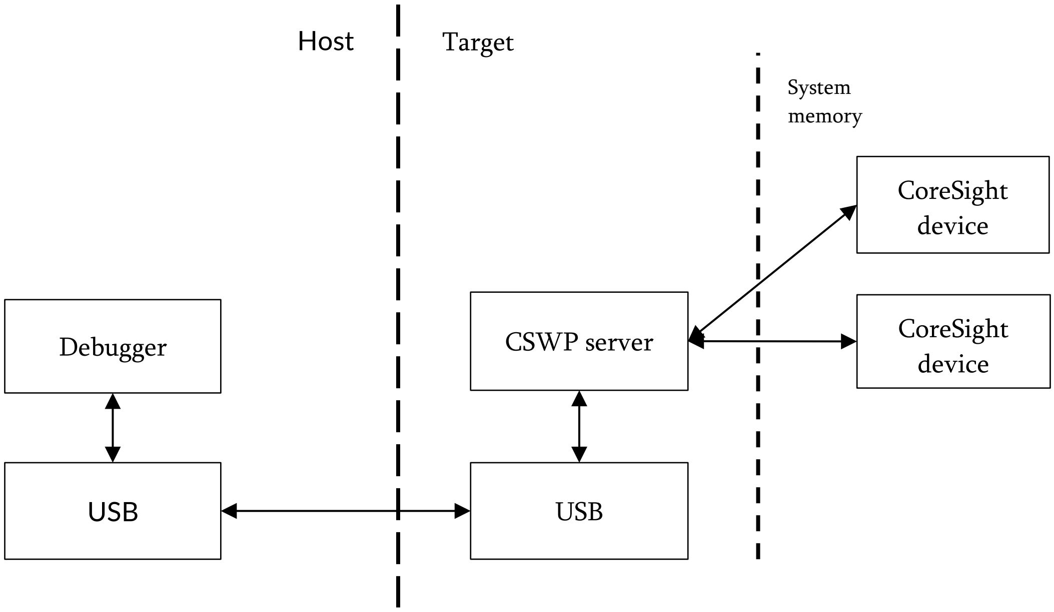
3 INTRODUCTION

CoreSight SoC-600 introduces support for debug and trace over functional I/O, allowing a debug tool to use interfaces such as USB, PCIe or Ethernet instead of JTAG/SWD and dedicated trace connectors.

CoreSight provides access to debug and trace logic in a system as a number of components (e.g. core debug logic, trace source, trace sink) on a memory bus. Each component has a number of registers that are accessed by reading or writing on this bus. A debugger will implement higher level operations such as stop a core, read core registers or configure an ETM as a sequence of accesses to these registers. A traditional JTAG/SWD debugger will perform these accesses by generating the JTAG / SWD scans to access the DAP (debug access port) on the system.



Debug over functional I/O replaces the JTAG / SWD transport with a functional I/O interface, e.g. USB, communicating with an agent within the target. The CSWP protocol defines the messages sent between the debugger and the target agent over this functional I/O interface.



The on target agent provides access to a number of memory busses, with CoreSight components accessed at addresses on these busses. These are accessed using the memory read and write commands.

The on target agent can also provide access to higher level components in the system. For example, the linux kernel contains support for hardware trace using CoreSight components and provides drivers that can be accessed via the filesystem. These devices can be accessed via CSWP, with the properties of these drivers presented as registers.

The CSWP protocol is transport neutral and the same encoding can be used on any transport. This document includes interface definitions for use over USB. Other interfaces such as PCIe, Ethernet are yet to be defined.

ARM has also defined the CMSIS-DAP protocol for accessing CoreSight DAPs over USB. However, CMSIS-DAP is a lower level protocol, providing SWD/JTAG and DAP register access and so has little in common with the CSWP protocol.

Trace data from the target to the host is not sent using the CSWP channel / protocol, but on a different channel for example a different USB interface / endpoint. This document includes information on the trace over USB interface.

4 PROTOCOL OVERVIEW

CSWP consists of a number of request messages sent from client (host) to server (target) and corresponding response messages sent from server to client. The server may also send asynchronous messages to the client.

Each request message consists of a header, followed by one or more sub-requests. Likewise, each response message consists of a header followed by one or more sub-responses. Packing multiple sub-requests / sub-responses into one message improves efficiency by allowing multiple operations to be performed on every round trip to the target – e.g. configuring and enabling an ETM may require many registers to be written. Note that one protocol message may span several transport level packets (e.g. 64 or 512 byte packets with USB).

4.1 Devices

Configuring and controlling trace involves accessing a number of CoreSight devices. These can be higher level devices such as the CoreSight drivers provided by the Linux kernel, low-level access to memory mapped devices via a memory access port (MEM-AP), or register based APs (JTAG-AP). CSWP allows connections to multiple devices, with requests taking a device parameter where necessary. Each device can provide access to:

- Registers: The device provides a number of registers that can be discovered and accessed via the CSWP_REG_xxxx requests
- Memory: The device provides an address space that can be accessed via the CSWP_MEM_xxxx requests

Thus, a higher level device such as a kernel CoreSight driver will be a Register device and the components accessible via a MEM-AP will be accessed via a Memory device. The MEM-AP device may also provide access to the MEM-AP registers.

An example system might contain the following devices:

Device	Type	Description
0	Memory	Direct access to system memory
1	Memory	APB accessed via MEM-AP at address 0x2000000
2	Register	Kernel driver for ETM trace source at address 0x28010000
3	Register	Kernel driver for STM trace source at address 0x28020000
4	Register	Kernel driver for ETR trace sink at address 0x28030000

A target may define a fixed set of devices that it supports. This can be discovered by the client by calling CSWP_GET_DEVICES after initialising the connection, without first calling CSWP_SET_DEVICES.

4.2 Device discovery

CoreSight provides ROM tables and topology detection registers to allow a debugger to discover the CoreSight devices within a target. However, this is an invasive operation that may affect the state of the target and often is not possible when some subsystems within the target are powered down.

To address this, CSWP provides the CSWP_GET_SYSTEM_DESCRIPTION message which allows the CSWP server to return a description of the components within the system and their relationships. A server may store this description as static data (e.g. in flash memory or filesystem), so does not need to interact with and affect the system in order to provide the description. The system description uses the SDF format which provides a list of access points (MEM-APs) in the system, the CoreSight devices in the system with their base addresses on their access points and the relationships between the devices, e.g. the links between devices for trace data.

Once a debug tool has obtained the system description, it must match the devices in the description with the devices reported by the CSWP server (using CSWP_GET_DEVICES) using the names of the devices. For each device in the description, the CSWP server must either:

- Provide a specific device in CSWP_GET_DEVICES. A debug tool should use this to interact with the device.
- Provide the memory device (MEM-AP) that the device can be accessed through. A debug tool should use this memory device with the base address of the device.

If a target cannot provide a system description, CSWP_GET_SYSTEM_DESCRIPTION will return CSWP_UNSUPPORTED and the debug tool can:

- Use system descriptions provided by other methods (e.g. as files or stored within the debug tool)
- Use ROM tables + topology detection to discover the devices in the system
- Allow the user to manually configure the system

To read the ROM tables, the debug tool should:

- Read the ROM table address registers. The names of registers to read depend on the device_type field – see section 7 for details on supported device type and the registers to read.
- Read the component ID and peripheral ID registers of the ROM table to get the table format
- Read the ROM table content. The ROM table may contain references to other devices that have further levels of ROM tables (e.g. DP may have MEM-APs, MEM-APs can be nested) – These may already be reported by CSWP_GET_DEVICES, or if the target server supports CSWP_SET_DEVICES, then the device list can be modified to add the discovered device. The ROM table discovery can then be performed on the added device.

4.3 Device configuration

Devices may have configurable options, for example the base address of a MEM-AP, the path to access a kernel driver, or the default flags to use for memory accesses. The CSWP_GET_CONFIG command allows a client to get the value of a configuration item and the CSWP_SET_CONFIG command allows the client to set the value of a configuration item.

Each device has a pre-defined read-only configuration item, “CONFIG_ITEMS” which is a newline separated list of the configuration items supported by that device.

Configuration item values are encoded as strings. Integer values are represented as decimal digits, or hexadecimal digits with a “0x” prefix. Boolean values are represented as “true” or “false” (case insensitive).

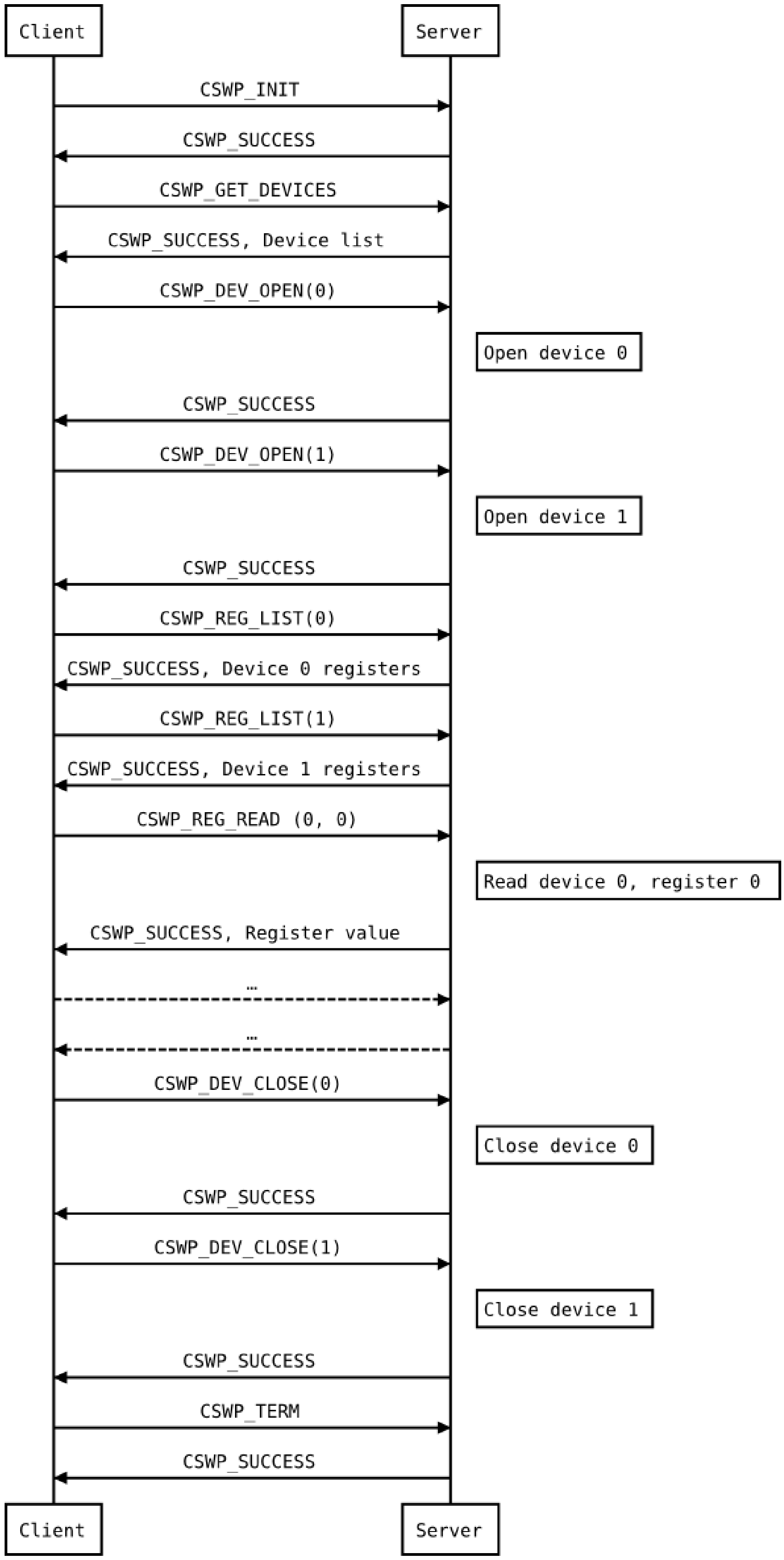
4.4 Connecting and disconnecting

A debug session must start with a CSWP_INIT request from client to server. This allows the on target agent to initialise any resources required and for client and server to agree on the supported protocol and exchange information.

The debug session ends with a CSWP_TERM request. This instructs the server that the client has finished and it can release any resources no longer required. If a server detects the connection has been lost (e.g. USB cable disconnected, client crashed) without a shutdown request, then it should behave as though a CSWP_TERM request was received.

Before accessing the resources of a device, the device must be opened with CSWP_DEV_OPEN. This allows the on target agent to allocate any resources required and ensure the device is available (e.g. powered on). At the end of the debug session the device must be closed with CSWP_DEV_CLOSE to allow the agent to clean up and release resources. Open connections to multiple devices are permitted.

Multiple client connections are permitted where the transport mechanism allows this. For example, a TCP transport would allow multiple connections. However, a USB Device Interface can only be opened by a single process (client), so would not permit multiple clients without a host server acting as a proxy to multiple client processes, or a target could provide multiple client interfaces. Where multiple clients are permitted, the server must be implemented to ensure concurrent access by multiple clients are safe, for example using a mutex to ensure only one message is processed at a time. The device list cannot be changed while there are open connections to devices – all clients must use the same device list.


Client
Server
CSWP_INIT
CSWP_SUCCESS
CSWP_GET_DEVICES
CSWP_SUCCESS, Device list
CSWP_DEV_OPEN(0)
Open device 0
CSWP_SUCCESS
CSWP_DEV_OPEN(1)
Open device 1
CSWP_SUCCESS
CSWP_REG_LIST(0)
CSWP_SUCCESS, Device 0 registers
CSWP_REG_LIST(1)
CSWP_SUCCESS, Device 1 registers
CSWP_REG_READ (0, 0)
Read device 0, register 0
CSWP_SUCCESS, Register value
…
…
CSWP_DEV_CLOSE(0)
Close device 0
CSWP_SUCCESS
CSWP_DEV_CLOSE(1)
Close device 1
CSWP_SUCCESS
CSWP_TERM
CSWP_SUCCESS
Client
Server

4.5 Registers

Registers are made up of 32-bit elements, rounded up to the next 32-bit size, with unused higher order bits RAZ/WI. Registers larger than 32-bit are split across consecutive 32-bit elements, with the low order elements first.

Accesses to multiple element registers must access all elements: partial reads/writes are not supported.

Each register has an ID number that is unique within that device. Registers spanning multiple 32-bit elements have consecutive IDs for each element. Otherwise, IDs do not need to be allocated sequentially, allowing a device to encode access information in the register ID.

For example a device with the following registers:

- R_32: a 32-bit register
- R_8: an 8-bit register
- R_40: a 40-bit register
- R_64: a 64-bit register

Could have the following register IDs:

ID	Register
0	R_32[31:0]
1	R_8[7:0], bits [31:8] RAZ/WI
10	R_40[31:0]
11	R_40[39:32], bits [31:8] RAZ/WI
20	R_64[31:0]
21	R_64[63:32]

Clients can discover the registers supported by a device using the CSWP_REG_LIST command. This returns information about each register, including ID number, size (number of 32-bit elements), name and display name.

4.6 Batch operations

Some operations may require many register / memory accesses. In order to do this efficiently, the number of round trips between client and server should be minimized. CSWP allows several operations (sub-requests) to be performed in a single request message. The request message header indicates the overall message size, number of sub-requests and behaviour on error (whether to continue processing if one sub-request fails or to cancel subsequent sub-requests). The response message header indicates the overall message size and number of sub-responses. The response message contains a sub-response for each sub-request of the original message.

When reading an incoming message, a client / server should use the indicated size to ensure the entire message has been read before processing.

5 PROTOCOL VERSIONS

The current protocol version is 1.0

1.0	Initial protocol version

The first element sent by both client and server in the CSWP_INIT command is the protocol version. A server may reject a connection from a client by replying with CSWP_INCOMPATIBLE if the client's protocol version is not supported by the server. A client may terminate the connection if the server protocol version indicates it does not support features required by the client. Otherwise clients and servers should use the maximum version supported by both client and server.

6 PROTOCOL DESCRIPTION

The CSWP protocol consists of requests sent from client to server and responses from server to client.
Each message consists of a header followed by a number of sub-messages.
The header and each sub-message are a sequence of fields.

6.1 Field encoding

Each field can be one of the following data types

Name	Details	Encoding
uint8	8-bit unsigned integer	1 byte
uint32	32-bit LE unsigned integer	4 bytes, least significant first
uint64	64-bit LE unsigned integer	8 bytes, least significant first
varint	Variable length unsigned integer	1 or more bytes. See below
string	Variable length UTF-8 text string	varint followed by data bytes. See below

Messages fields are packed together with no padding between fields.

6.1.1 Variable length integer (varint)

Variable length integers allow efficient encoding of unsigned integer values. A varint consists of a number of bytes, with the least significant byte first. Each byte contains 7 bits of data (bits 6:0) and a continuation bit (bit 7). If the continuation bit is set, then the subsequent byte contains further higher order bits. For example:

- 1 is encoded as 0x01 (data: 0b0000001, continuation: 0)
- 127 is encoded as 0x7F (data: 0b1111111, continuation: 0)
- 128 is encoded as 0x80 0x01 (data: 0b0000000, continuation: 1; data 0b0000001, continuation 0)

The maximum size of a varint is 64-bits, resulting in a maximum length of 10 bytes.

6.1.2 Strings

Strings are UTF-8 encoded and are represented as a varint giving number of bytes in the string, followed by the string data. For example "Hello world" is encoded as:

0x0B 0x48 0x65 0x6C 0x6C 0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64.

An empty string has 0 length and is encoded as 0x00.

6.2 Message Header

6.2.1 Request

uint32	message_length	Message length (including header)
varint	num_sub_requests	Number of sub-requests
uint8	error_mode	Behaviour on error: 0: Continue processing sub-requests 1: Cancel subsequent sub-requests

The message_length allows the transport layer to ensure the entire message has been read before processing its contents.

Requests consist of a number of sub-requests. Each sub-command has the following header

varint	message_type	Message type

The message_type is followed by request specific data as specified below. After each sub-request, there may be another sub-request consisting of a message_type field followed by request specific data for that message.

6.2.2 Response

uint32	message_length	Message length (including header)
varint	num_sub_responses	Number of sub-responses

The message_length allows the transport layer to ensure the entire message has been read before processing its contents.

A sub-response is included for each sub-request in the corresponding request. Each sub-response has the following headers

varint	message_type	Message type
varint	error_code	Error code

If error code is zero, then the normal response for the request follows

If error code is non-zero, then error information follows:

string	error_message	Error message

After each sub-response, there may be another sub-response as described above.

6.3 Error codes

The following error codes are defined:

Code	Name	Description
0x0000	CSWP_SUCCESS	The command was executed successfully
0x0001	CSWP_FAILED	Other error occurred
0x0002	CSWP_CANCELLED	Cancelled due to previous error
0x0003	CSWP_NOT_INITIALIZED	Not initialized
0x0010	CSWP_BUFFER_FULL	Insufficient space when encoding buffer
0x0011	CSWP_BUFFER_EMPTY	Insufficient data when decoding buffer
0x0012	CSWP_OUTPUT_BUFFER_OVERFLOW	Insufficient space in output buffer when decoding
0x0020	CSWP_COMMS	A communication error occurred
0x0021	CSWP_INCOMPATIBLE	The server is not compatible with the client
0x0022	CSWP_TIMEOUT	A timeout occurred executing a command
0x0023	CSWP_UNSUPPORTED	Command unsupported
0x0024	CSWP_DEVICE_UNSUPPORTED	Unsupported device
0x0025	CSWP_INVALID_DEVICE	Invalid device ID
0x0026	CSWP_BAD_ARGS	Bad arguments to command
0x0028	CSWP_NOT_PERMITTED	Operation not permitted
0x0200	CSWP_REG_FAILED	Register access failed
0x0201	CSWP_REG_PARTIAL	Attempt to access part of a multiple element register
0x0300	CSWP_MEM_FAILED	Memory access failed
0x0301	CSWP_MEM_INVALID_ADDRESS	Invalid address for memory access
0x0302	CSWP_MEM_BAD_ACCESS_SIZE	Invalid access size for memory access
0x0303	CSWP_MEM_POLL_NO_MATCH	Memory poll target value not read

6.4 Messages

The following message types are defined

Message Type	Name	Description
0x00000001	CSWP_INIT	Initialize CSWP session
0x00000002	CSWP_TERM	Terminate CSWP session
0x00000005	CSWP_CLIENT_INFO	Information message from client
0x00000010	CSWP_SET_DEVICES	Set device list
0x00000011	CSWP_GET_DEVICES	Get device list
0x00000012	CSWP_GET_SYSTEM_DESCRIPTION	Get system description
0x00000100	CSWP_DEVICE_OPEN	Open device
0x00000101	CSWP_DEVICE_CLOSE	Close device
0x00000102	CSWP_SET_CONFIG	Set configuration item
0x00000103	CSWP_GET_CONFIG	Get configuration item
0x00000104	CSWP_GET_DEVICE_CAPABILITIES	Get device capabilities
0x00000200	CSWP_REG_LIST	Get available registers
0x00000201	CSWP_REG_READ	Read registers
0x00000202	CSWP_REG_WRITE	Write registers
0x00000300	CSWP_MEM_READ	Read memory
0x00000301	CSWP_MEM_WRITE	Write memory
0x00000302	CSWP_MEM_POLL	Poll memory
0x00001000	CSWP_ASYNC_MESSAGE	Error/information message

6.4.1 CSWP_INIT

Initialize the CSWP session. This allows the server to perform any initialization required. Note that hardware should not be accessed at this time

6.4.1.1 Command

varint	CSWP_INIT	Command ID
varint	protocol_version	Indicates the maximum protocol version supported by the client. Encoded as: bits[N:8]: major version, currently 1 bits[7:0] minor version, currently 0
string	client_id	A string identifying the client.

6.4.1.2 Response

varint	CSWP_INIT	Command ID
varint	0	Successful command
varint	protocol_version	Indicates the maximum protocol version supported by the server. Encoded as: bits[N:8]: major version, currently 1 bits[7:0] minor version, currently 0
string	server_id	A string identifying the server
varint	server_version	The version number of the server. bits[7:0] indicate the minor version, bits[n:8] indicate the major version

6.4.2 CSWP_TERM

Terminate the CSWP session. This allows the server to cleanup and release any resources

6.4.2.1 Command

varint	CSWP_TERM	Command ID

6.4.2.2 Response

varint	CSWP_TERM	Command ID
varint	0	Successful command

6.4.3 CSWP_CLIENT_INFO

Information message from a client. A server may include this message in any logging that it performs. This allows client operations to be matched to operations performed by the server

6.4.3.1 Command

varint	CSWP_CLIENT_INFO	Command ID
string	message	Message text

6.4.3.2 Response

varint	CSWP_CLIENT_INFO	Command ID
varint	0	Successful command

6.4.4 CSWP_ASYNC_MESSAGE

Asynchronous error / information message from server to client

6.4.4.1 Response

varint	CSWP_ASYNC_MESSAGE	Command ID
varint	error_code	Error code
varint	device_id	Device number (0 indicates system)
varint	level	Message level
string	message	Message text

level can be one of:

CSWP_ASYNC_MESSAGE_ERROR	0	Error
CSWP_ASYNC_MESSAGE_WARNING	1	Warning
CSWP_ASYNC_MESSAGE_INFO	2	Information
CSWP_ASYNC_MESSAGE_DEBUG	3	Debug information

6.4.5 CSWP_SET_DEVICES

Set the device list. This replaces any existing device list in the server and may only be sent when no device connections are open. Device numbers in subsequent commands are indexes into this list. A server may have a fixed device list, in which case CSWP_UNSUPPORTED will be returned.

Changes to the device list are not permitted if there are any open device connections. CSWP_NOT_PERMITTED will be returned if CSWP_SET_DEVICES is sent while there are open devices.

6.4.5.1 Command

varint	CSWP_SET_DEVICES	Command ID
varint	device_count	Number of devices

This is followed by device_count instances of:

string	device_name	Device name
string	device_type	Device type. See section 7 for supported device types

6.4.5.2 Response

varint	CSWP_SET_DEVICES	Command ID
varint	0	Successful command

6.4.6 CSWP_GET_DEVICES

Get the current device list. Device numbers in subsequent commands are indexes into this list. A server may predefine a device list that can be discovered by making a CSWP_GET_DEVICES request without a prior CSWP_SET_DEVICES request.

6.4.6.1 Command

varint	CSWP_GET_DEVICES	Command ID

6.4.6.2 Response

varint	CSWP_GET_DEVICES	Command ID
varint	0	Successful command
varint	device_count	Number of devices

This is followed by device_count instances of:

string	device_name	Device name
string	device_type	Device type. See section 7 for supported device types

6.4.7 CSWP_GET_SYSTEM_DESCRIPTION

Get the system description. This command provides data describing the devices present in the target and the relationship between them.
The format field indicates the format of the system description. Supported formats are:

CSWP_UNSUPPORTED will be retuned if the target does not provide a system description.

0	SDF format
1	SDF format compressed with gzip

System descriptions may be quite large (several kBytes) for complex systems, which could consume a lot of flash memory when the sever is implemented in a microcontroller. Compressing the system description using gzip can greatly reduce the size of the system description

6.4.7.1 Command

varint	CSWP_GET_SYSTEM_DESCRIPTION	Command ID

6.4.7.2 Response

varint	CSWP_GET_SYSTEM_DESCRIPTION	Command ID
varint	0	Successful command
varint	format	Format of system description
varint	size	Length of system description data
uint8 x size	data	System description data

6.4.9 CSWP_DEV_OPEN

Open device. This must be sent before any other request is made to the device. Some devices require further configuration before their registers / memory can be accessed, for example MEM-AP requires a base address.

6.4.9.1 Command

varint	CSWP_DEV_OPEN	Command ID
varint	device_id	Device number

6.4.9.2 Response

varint	CSWP_DEV_OPEN	Command ID
varint	0	Successful command
string	device_info	Description of device

6.4.10 CSWP_DEV_CLOSE

Close device and release any associated resources.

6.4.10.1 Command

varint	CSWP_DEV_CLOSE	Command ID
varint	device_id	Device number

6.4.10.2 Response

varint	CSWP_DEV_CLOSE	Command ID
varint	0	Successful command

6.4.11 CSWP_SET_CONFIG

Set configuration item value. See section 4.2 for details

6.4.11.1 Command

varint	CSWP_SET_CONFIG	Command ID
varint	device_id	Device number
string	name	Configuration item name
string	value	Configuration item value

6.4.11.2 Response

varint	CSWP_SET_CONFIG	Command ID
varint	0	Successful command

6.4.12 CSWP_GET_CONFIG

Get configuration item value. See section 4.2 for details

Each device has a pre-defined read-only configuration item, “CONFIG_ITEMS” which is a newline separated list of the configuration items supported by that device.

6.4.12.1 Command

varint	CSWP_GET_CONFIG	Command ID
varint	device_id	Device number
string	Name	Configuration item name

6.4.12.2 Response

varint	CSWP_GET_CONFIG	Command ID
varint	0	Successful command
string	value	Configuration item value

6.4.13 CSWP_GET_DEVICE_CAPABILITIES

Get device capabilities. May be called without an open connection to the device.
This command returns a bitfield describing the capabilities of the device. Each capability may have optional data.

6.4.13.1 Command

varint	CSWP_GET_DEVICE_CAPABILITIES	Command ID
varint	device_id	Device number

6.4.13.2 Response

varint	CSWP_GET_DEVICE_CAPABILITIES	Command ID
varint	0	Successful command
varint	capabilities	Device capabilities bitfield
0-N x varint	capability_data	Capability specific data

The device capabilities field is a bitfield containing the following

Bits	Constant	Description	Extra data
0	CSWP_CAP_REG = 0x1	Device supports register access	
1	CSWP_CAP_MEM = 0x2	Device supports memory access	

6.5 Register Commands

6.5.1 CSWP_REG_LIST

List registers supported by device

6.5.1.1 Command

varint	CSWP_REG_LIST	Command ID
varint	device_id	Device number

6.5.1.2 Response

varint	CSWP_REG_LIST	Command ID
varint	0	Successful command
varint	num_regs	Number of registers

This is followed by the information about each register. There are num_regs occurences of:

varint	reg_id	Register base ID
string	reg_name	Register name
varint	reg_size	Number of 32-bit elements in the register
string	reg_display	Display name of register (may be empty)
string	reg_description	Description of register (may be empty)

6.5.2 CSWP_REG_READ

Read registers

Registers larger than 32-bit are accessed as a sequence of 32-bit registers, with the least significant word first. The ID of each sub-register is included in the request. The size of each register is given by CSWP_REG_LIST and the client should use this information when encoding the request and decoding the response. All elements of the register must be read – partial accesses are not permitted. CSWP_REG_PARTIAL is returned if a partial access is attempted.

Registers are read in the order specified in the register_id parameter and register IDs may appear multiple times to allow multiple reads of the register.

CSWP_REG_FAILED is returned if any register cannot be read.

6.5.2.1 Command

varint	CSWP_REG_READ	Command ID
varint	device_id	Device number
varint	count	Number of registers to read

This is followed by count occurrences of

varint	register_id	Register ID

6.5.2.2 Response

varint	CSWP_REG_READ	Command ID
varint	0	Successful command

This is followed by count occurrences of

uint32	register_data	Register values

6.5.3 CSWP_REG_WRITE

Write registers

Registers larger than 32-bit are accessed as a sequence of 32-bit registers, with the least significant word first. The ID and value of each sub-register is included in the request. The size of each register is given by CSWP_REG_LIST and the client should use this information when encoding the request. All elements of the register must be written – partial writes are not permitted. CSWP_REG_PARTIAL is returned if a partial access is attempted.

Registers are read in the order specified in the register_id parameter and register IDs may appear multiple times to allow multiple writes of the register.

CSWP_REG_FAILED is returned if any register cannot be written.

6.5.3.1 Command

varint	CSWP_REG_WRITE	Command ID
varint	device_id	Device number
varint	count	Number of registers to write

This is followed by count occurrences of:

varint	register_id	Register ID
uint32	register_data	Register value

6.5.3.2 Response

varint	CSWP_REG_WRITE	Command ID
varint	0	Successful command

6.6 Memory commands

Access size is specified as the number of bytes for each element. Supported sizes are:

CSWP_ACC_SIZE_DEF	0	Default access
CSWP_ACC_SIZE_8	1	8-bit access
CSWP_ACC_SIZE_16	2	16-bit access
CSWP_ACC_SIZE_32	4	32-bit access
CSWP_ACC_SIZE_64	5	64-bit access

Default access provides the same data as 8-bit (byte), but allows the server to access the data in an optimal way, e.g. using 32-bit accesses

Data is sent in target memory order – for a LE or BE-8 system the order of bytes in the message will be the same order as they are in memory.

The flags field controls the behaviour of the commands and allows device specific parameters to be passed to the memory commands. This could be used to indicate which address space to access, permissions bits etc. See section 7 for flags defined for supported devices.

The CSWP_MEM_NO_ADDR_INC instructs the CSWP_MEM_READ and CSWP_MEM_WRITE commands to repeatedly read or write the same address. The access size must be set to 8/16/32/64 bit and the access is repeated size / accessSizeBytes times.

6.6.1 CSWP_MEM_READ

Read memory

6.6.1.1 Command

varint	CSWP_MEM_READ	Command ID
varint	device_id	Device number
uint64	address	Address
varint	size	Number of bytes to read
varint	access_size	Access size (bytes)
varint	flags	Device specific flags

The flags field can have the following values:

Bits	Name	Description
0	CSWP_MEM_NO_ADDR_INC	Repeatedly read the same address
8-	Device specific	Device specific flags

6.6.1.2 Response

varint	CSWP_MEM_READ	Command ID
varint	0	Successful command
varint	Size	Number of bytes read
size * uint8	mem_data	Memory data

6.6.2 CSWP_MEM_WRITE

Write memory

6.6.2.1 Command

varint	CSWP_MEM_WRITE	Command ID
varint	device_id	Device number
uint64	address	Address
varint	size	Number of bytes to write
varint	access_size	Access size (bytes)
varint	flags	Access flags
size * uint8	mem_data	Memory data

The flags field can have the following values:

Bits	Name	Description
0	CSWP_MEM_NO_ADDR_INC	Repeatedly write the same address
8-	Device specific	Device specific flags

6.6.2.2 Response

varint	CSWP_MEM_WRITE	Command ID
varint	0	Successful command

6.6.3 CSWP_MEM_POLL

Poll memory until a target value is read.

Data is repeatedly read from the specified address until a target value is reached. A mask is applied (bitwise AND) to both the data read and the target value before they are compared. If the CSWP_MEM_POLL_MATCH_NE flag is set, then the poll completes when the masked read value does not match the masked target value. Otherwise, the poll completes when the masked read value matches the masked target value.

If the read value does not match within the specified number of tries, the CSWP_MEM_POLL_NO_MATCH error code is returned.

If the CSWP_MEM_POLL_CHECK_LAST flag is set then the last value read by a previous poll command is masked and compared against the target value. No new data is read from the target. This allows for conditions that cannot be expressed as a single poll operation. For example, a poll until equal command can be used to wait for a bit to be set to indicate an operation is complete and a poll command with CSWP_MEM_POLL_CHECK_LAST can be used with a different mask to check that error status bits are clear.

The last unmasked read value is returned in the response.

6.6.3.1 Command

varint	CSWP_MEM_POLL	Command ID
varint	device_id	Device number
uint64	address	Address
varint	size	Number of bytes to read
varint	access_size	Access size (bytes)
varint	Flags	Access flags
varint	Tries	Number of times to read
varint	Interval	Interval in mircoseconds between each attempt. 0 for no delay.
size * uint8	Mask	Mask to apply to read data and target value when comparing
size * uint8	value	Target value to poll for

The flags field can have the following values:

Bits	Name	Description
1	CSWP_MEM_POLL_MATCH_NE	Poll until read data does not match target
2	CSWP_MEM_POLL_CHECK_LAST	Check last read value against target value
8-	Device specific	Device specific flags

6.6.3.2 Response

varint	CSWP_MEM_POLL	Command ID
varint	0	Successful command
varint	Size	Number of bytes read
size * uint8	mem_data	Last data read

7 SUPPORTED DEVICES

The device type field in the device list indicates the type of the device and informs a debug tool about how that device can be used and how to discover further devices accessed via that device.

The following device types are defined in this section:

Type	Description
mem-ap.v2	MEM-AP APv2 defined in ADIv6 (See [ADIv6] C2)
mem-ap.v1	MEM-AP APv1 defined in ADIv5 (See [ADIv5])
memory	Other memory space
dap.v6	ADIv6 Debug Access Port
dap.v5	ADIv5 Debug Access Port
jtag-ap	JTAG-AP defined in ADIv5 (See [ADIv5]) and ADIv6 (See [ADIv6])
cscomp	CoreSight component
linux.cscomp	Linux driver for a CoreSight component

Other implementation defined device types are permitted.

7.1 mem-ap.v2

This device type represents a Memory Access Port (MEM-AP) implementing the APv2 architecture as defined in ADIv6 (See [ADIv6] C2).

MEM-APs provide access to system memory busses. A CSWP server supporting MEM-APs will provide access to AP registers and the memory space accessed via the AP.

ADIv6 allows MEM-APs to be nested – a MEM-AP and its associated memory space may need to be accessed via another MEM-AP. Thus access to a nested AP may require several operations on the top-level AP. For efficiency, it is recommended that the CSWP server presents nested MEM-APs as separate devices and performs the top level register accesses necessary to access the nested AP.

The MEM-AP may be accessed via another device, defined by the “PARENT” configuration item. This can be:

- Empty: The MEM-AP is accessed directly by the target debug agent at the given BASE_ADDRESS
- A dap.v6 device: The MEM-AP is accessed via a ADIv6 DAP, with DP.SELECT set to BASE_ADDRESS
- A mem-ap.v2 device: The MEM-AP is accessed via another MEM-AP (nested) with BASE_ADDRESS locating the MEM-AP within the parent MEM-AP’s address space

The type of the MEM-AP, e.g. AHB, APB, AXI can be identified from the IDR register.

7.1.1 Configuration items

The MEM-AP has the following configuration items:

Item	Description
BASE_ADDRESS	Base address of the MEM-AP registers
PARENT	Name of parent device
DEFAULT_CSW	Default value for CSW register

7.1.2 Registers

The MEM-AP provides the 32-bit registers defined in (See [ADIv6] C2). The BASE register in ADIv6 is split across 2 non-contiguous registers, so is implemented as 2 registers: BASE0 represents the least significant word, BASE1 represents the most significant word.

The register IDs are their offset from the base address of the MEM-AP. Note that memory accesses via the AP, including via other devices attached to this AP will involve modifying these registers, so debug tools cannot assume their values are preserved following such accesses.

7.1.3 Memory

The addresses in the memory access commands will be written to the AP TAR. The AP CSW will be set according to the size of the access, the default CSW value and the following bits of the flags field:

Bits	Name	Description
8	CSWP_MEMAP_OVERRIDE_INCR	Override CSW[5:4] with value from CSWP_MEM_AP_INCR
9	CSWP_MEMAP_OVERRIDE_PROT	Override CSW[30:24] with value from CSWP_MEM_AP_PROT
10	CSWP_MEMAP_OVERRIDE_MODE	Override CSW[11:8] with value from CSWP_MEM_AP_MODE
11	CSWP_MEMAP_OVERRIDE_TYPE	Override CSW[15:12] with value from CSWP_MEM_AP_TYPE
12	CSWP_MEMAP_OVERRIDE_ERR	Override CSW[17:16] with value from CSWP_MEM_AP_ERR
24:13	CSWP_MEMAP_INCR	Bits written to CSW[5:4] (AddrInc) Controls TAR increment after each element
21:15	CSWP_MEMAP_PROT	Bits written to CSW[30:24] (Prot)
25:22	CSWP_MEMAP_MODE	Bits written to CSW[11:8] (Mode)
29:26	CSWP_MEMAP_TYPE	Bits written to CSW[15:12] (Type)
31:30	CSWP_MEMAP_ERR	Bit written to CSW[17:16] (ERRNSTOP,ERRNPASS)

The writable bits of the CSW register value are set to:

Bits	Value
30:24	DEFAULT_CSW[30:24] if CSWP_MEMAP_OVERRIDE_PROT == 0 CSWP_MEMAP_PROT if CSWP_MEMAP_OVERRIDE_PROT == 1
17:16	DEFAULT_CSW[17:16] if CSWP_MEMAP_OVERRIDE_ERR == 0 CSWP_MEMAP_ERR if CSWP_MEMAP_OVERRIDE_ERR == 1
15:12	DEFAULT_CSW[15:12] if CSWP_MEMAP_OVERRIDE_TYPE == 0 CSWP_MEMAP_TYPE if CSWP_MEMAP_OVERRIDE_TYPE == 1
11:8	DEFAULT_CSW[11:8] if CSWP_MEMAP_OVERRIDE_MODE == 0 CSWP_MEMAP_MODE if CSWP_MEMAP_OVERRIDE_MODE == 1
5:4	DEFAULT_CSW[5:4] if CSWP_MEMAP_OVERRIDE_INCR == 0 CSWP_MEMAP_INCR if CSWP_MEMAP_OVERRIDE_INCR == 1
2:0	b'000: access_size == 1 (8 bit) b'001: access_size == 2 (16 bit) b'010: access_size == 3 (32 bit) b'011: access_size == 4 (64 bit)

Each MEM-AP type (APB-AP, AHB-AP, AXI-AP, etc) defines the valid values for each of these fields.

7.1.4 Child device discovery

Child devices of the MEM-AP can be discovered by reading the ROM table at the address given by the BASE1:BASE0 registers.

7.2 mem-ap.v1

This device type represents a Memory Access Port (MEM-AP) implementing the APv1 architecture as defined in ADIv5 (See [ADIv5] C2).

The MEM-AP is accessed via a ADIv5 DAP with another device, with the AP ID number written to APSEL field of DP.SELECT. The parent DAP may be accessible via CSWP using the device names by the "PARENT" configuration item. This will be empty if the DAP cannot be accessed.

The type of the MEM-AP, e.g. AHB, APB, AXI can be identified from the IDR register

7.2.1 Configuration items

The MEM-AP has the following configuration items:

Item	Description
AP	AP ID number
PARENT	Name of parent device
DEFAULT_CSW	Default value for CSW register

7.2.2 Registers

The MEM-AP provides the 32-bit registers defined in (See [ADIv5] C2). The BASE register in ADIv5 is split across 2 non-contiguous registers, so is implemented as 2 registers: BASE0 represents the least significant word, BASE1 represents the most significant word.

7.2.3 Memory

The addresses in the memory access commands will be written to the AP TAR. The AP CSW will be set according to the size of the access, the default CSW value and the following bits of the flags field:

Bits	Name	Description
8	CSWP_MEMAP_OVERRIDE_INCR	Override CSW[5:4] with value from CSWP_MEM_AP_INCR
9	CSWP_MEMAP_OVERRIDE_PROT	Override CSW[30:24] with value from CSWP_MEM_AP_PROT
10	CSWP_MEMAP_OVERRIDE_MODE	Override CSW[11:8] with value from CSWP_MEM_AP_MODE
11	CSWP_MEMAP_OVERRIDE_TYPE	Override CSW[15:12] with value from CSWP_MEM_AP_TYPE
12	CSWP_MEMAP_OVERRIDE_ERR	Override CSW[17:16] with value from CSWP_MEM_AP_ERR
24:13	CSWP_MEMAP_INCR	Bits written to CSW[5:4] (AddrInc) Controls TAR increment after each element
21:15	CSWP_MEMAP_PROT	Bits written to CSW[30:24] (Prot)
25:22	CSWP_MEMAP_MODE	Bits written to CSW[11:8] (Mode)
29:26	CSWP_MEMAP_TYPE	Bits written to CSW[15:12] (Type)

31:30	CSWP_MEMAP_ERR	Bit written to CSW[17:16] (ERRNSTOP,ERRNPASS)

The writable bits of the CSW register value are set to:

Bits	Value
30:24	DEFAULT_CSW[30:24] if CSWP_MEMAP_OVERRIDE_PROT == 0 CSWP_MEMAP_PROT if CSWP_MEMAP_OVERRIDE_PROT == 1
17:16	DEFAULT_CSW[17:16] if CSWP_MEMAP_OVERRIDE_ERR == 0 CSWP_MEMAP_ERR if CSWP_MEMAP_OVERRIDE_ERR == 1
15:12	DEFAULT_CSW[15:12] if CSWP_MEMAP_OVERRIDE_TYPE == 0 CSWP_MEMAP_TYPE if CSWP_MEMAP_OVERRIDE_TYPE == 1
11:8	DEFAULT_CSW[11:8] if CSWP_MEMAP_OVERRIDE_MODE == 0 CSWP_MEMAP_MODE if CSWP_MEMAP_OVERRIDE_MODE == 1
5:4	DEFAULT_CSW[5:4] if CSWP_MEMAP_OVERRIDE_INCR == 0 CSWP_MEMAP_INCR if CSWP_MEMAP_OVERRIDE_INCR == 1
2:0	b'000: access_size == 1 (8 bit) b'001: access_size == 2 (16 bit) b'010: access_size == 3 (32 bit) b'011: access_size == 4 (64 bit)

Each MEM-AP type (APB-AP, AHB-AP, AXI-AP, etc) defines the valid values for each of these fields.

7.2.4 Child device discovery

Child devices of the MEM-AP can be discovered by reading the ROM table at the address given by the BASE registers.

7.3 memory

A CSWP server can provide access to system memory address spaces. On a Linux system, this will be the physical memory accessed via /dev/mem (this requires a kernel build option to enable).

7.3.1 Configuration items

None

7.3.2 Registers

The memory device may provide an optional pseudo-register “BASE”, pointing to a ROM table (e.g. from a CoreSight subsystem mapped into a processor’s address space) to allow child device discovery.

7.3.3 Memory

No device specific values are defined for the CSWP_MEM_xxxx flags.

7.3.4 Child device discovery

If the device provides a BASE register, then a ROM table can be read at the address given by the BASE register.

7.4 dap.v6

A CSWP server may provide access to a ADIv6 DAP (See [ADIv6] B), for example where the CSWP server is running on another processor connected to the system under test via a JTAG or SWD link.

7.4.1 Configuration items

The DAP has the following optional configuration items:

Item	Description
MODE	“JTAG” or “SWD” to select communications protocol
CLOCKSPEED	Clockspeed for JTAG/SWD link

7.4.2 Registers

The DAP has the following register groups:

ID range	Description
0x000-x0FF	DPACC registers ID bits [3:2]: DPACC index
0x100-0x1FF	APACC registers ID bits [3:2]: APACC index
0x200-0x2FF	DP registers. Sets DPBANKSEL to select appropriate DP register bank. ID bits [7:4]: DPBANKSEL value ID bits [3:2]: DPACC index
0x300-0x3FF	Alternate functions of DP registers, usually datalink defined. Sets DPBANKSEL to select appropriate DP register bank ID bits [7:4]: DPBANKSEL value ID bits [3:2]: DPACC index

The full register list is:

Name	ID	Size	Description
DP0	0x000	32	DPACC Register 0
DP1	0x004	32	DPACC Register 1
DP2	0x008	32	DPACC Register 2
DP3	0x00C	32	DPACC Register 3
AP0	0x100	32	APACC Register 0
AP1	0x104	32	APACC Register 1
AP2	0x108	32	APACC Register 2
AP3	0x10C	32	APACC Register 3
DPIDR	0x200	32	DP DPIDR Register, RO
DPIDR1	0x210	32	DP DPIDR1 Register, RO
BASEPTR0	0x220	32	DP BASEPTR0 Register, RO
BASEPTR1	0x230	32	DP BASEPTR1 Register, RO
CTRLSTAT	0x204	32	DP CTRL/STAT Register, RW
DLCR	0x214	32	DP DLCR Register, RW

TARGETID	0x224	32	DP TARGETID Register, RO
DLPIDR	0x234	32	DP DLPIDR Register, RO
EVENTSTAT	0x244	32	DP EVENTSTAT Register, RO
SELECT1	0x254	32	DP SELECT1 Register, WO
SELECT	0x208	32	DP SELECT Register, WO
RDBUFF	0x20C	32	DP RDBUFF Register, RO
ABORT	0x300	32	DP ABORT Register, WO. Writes ABORT to IR on JTAG-DP
TARGETSEL	0x30C	32	DP TARGETSEL Register, SWD only, WO
RESEND	0x308	32	DP RESEND Register, SWD only, RO

7.4.3 Memory

Not supported.

7.4.4 Child device discovery

The BASEPTR1/BASEPTR0 registers contain the address of the top level component. The ID registers of this component can be read by setting the SELECT1/SELECT registers to the base address + ID register offset, then reading AP0-3 registers. The ID registers will identify the component as one of:

- A ROM table. This can be read using the SELECT/AP registers to discover the top level APs or other components.
- An AP. This is the only AP accessible from this DP and its child devices can be discovered as defined for the AP
- A debug component. This is the only component accessible from the DP

7.5 dap.v5

A CSWP server may provide access to a ADIv5 DAP (See [ADIv5] B), for example where the CSWP server is running on another processor connected to the system under test via a JTAG or SWD link.

7.5.1 Configuration items

The DAP has the following optional configuration items:

Item	Description
MODE	"JTAG" or "SWD" to select communications protocol
CLOCKSPEED	Clockspeed for JTAG/SWD link

7.5.2 Registers

The DAP has the following register groups:

ID range	Description
0x000-x0FF	DPACC registers ID bits [3:2]: DPACC index
0x100-0x1FF	APACC registers ID bits [3:2]: APACC index
0x200-0x2FF	DP registers. Sets DPBANKSEL to select appropriate DP register bank. ID bits [7:4]: DPBANKSEL value

	ID bits [3:2]: DPACC index
0x300-0x3FF	Alternate functions of DP registers, usually datalink defined. Sets DPBANKSEL to select appropriate DP register bank ID bits [7:4]: DPBANKSEL value ID bits [3:2]: DPACC index

The full register list is:

Name	ID	Size	Description
DP0	0x000	32	DPACC Register 0
DP1	0x004	32	DPACC Register 1
DP2	0x008	32	DPACC Register 2
DP3	0x00C	32	DPACC Register 3
AP0	0x100	32	APACC Register 0
AP1	0x104	32	APACC Register 1
AP2	0x108	32	APACC Register 2
AP3	0x10C	32	APACC Register 3
DPIDR	0x200	32	DP DPIDR Register, RO
CTRLSTAT	0x204	32	DP CTRL/STAT Register, RW
DLCR	0x214	32	DP DLCR Register, RW
TARGETID	0x224	32	DP TARGETID Register, RO
DLPIDR	0x234	32	DP DLPIDR Register, RO
EVENTSTAT	0x244	32	DP EVENTSTAT Register, RO
SELECT	0x208	32	DP SELECT Register, WO
RDBUFF	0x20C	32	DP RDBUFF Register, RO
ABORT	0x300	32	DP ABORT Register, WO. Writes ABORT to IR on JTAG-DP
TARGETSEL	0x30C	32	DP TARGETSEL Register, SWD only, WO
RESEND	0x308	32	DP RESEND Register, SWD only, RO

7.5.3 Memory

Not supported.

7.5.4 Child device discovery

The APs attached to a DP can be discovered by setting the SELECT register to examine the IDR register (APBANKSEL = 0xF, read AP3) of each AP.

7.6 jtag-ap

JTAG-AP provides access to JTAG devices from a DAP. Debug tools can use the CSWP server to access registers to perform the JTAG scans as defined in defined in [ADIv5]/[ADIv6] section C3.

7.6.1 Configuration items

None

7.6.2 Registers

The JTAG-AP device has the registers defined in [ADIv5]/[ADIv6] section C3. For an ADIv5 JTAG-AP, the register IDs are 0x00-0xFC. For an ADIv6 JTAG-AP, the JTAG-AP register IDs are 0xD00-0xDFC and CoreSight management register IDs are 0xF00-0xFFC.

7.6.3 Memory

Not supported.

7.6.4 Child device discovery

Debug tools can use scanchain autodetection to discover the devices on each port.

7.7 cscomp

A CSWP server can provide direct access to a CoreSight component as a separate device, for example devices attached directly to a DP which are not accessed via an AP.

For consistency with access via a MEM-AP, the CoreSight components registers are accessed as memory using the register offsets from the components programmer's model. This also allows more efficient access using the read/write repeat and batch operations. The component registers are also available as registers on the CSWP device. Thus, a component that is accessible via a MEM-AP and a specific device can be accessed either as:

- Memory accesses on the mem-ap.v6/mem-ap.v5 device, with address = component_base_address + register_offset
- Memory accesses on the cscomp device, with address = register_offset
- Register accesses on the cscomp device, with register ID = register_offset

7.7.1 Configuration items

None

7.7.2 Registers

The CoreSight component registers can be accessed as registers on the CSWP device, with the register IDs equal to the register's offset from the component programmer's model.

Discovery of the registers via CSWP_REG_LIST is not required as the server may not know the names and offsets of the registers from the programmer's model.

7.7.3 Memory

The CoreSight component registers can be accessed as memory on the CSWP device, with the address equal to the register's offset from the component programmer's model.

No device specific values are defined for the CSWP_MEM_xxxx flags.

7.8 linux.cscomp

The CoreSight drivers for trace in the linux kernel can be accessed via CSWP. The CSWP server runs as a userspace application and accesses the device properties in the subdirectories of /sys/bus/coresight/devices/. Each CSWP device represents one CoreSight device and provides registers for each entry in that device.

7.8.1 Configuration items

Each device has the following configuration items:

Item	Description
PATH	Path to directory containing driver properties

7.8.2 Registers

Each file item in the driver directory and its subdirectories is represented with a register. The registers supported by a device can be discovered using the CSWP_REG_LIST command. For example, the STM trace source contains:

```
enable_source
hwevent_enable
hwevent_select
mgmt/devid
mgmt/privmaskr
mgmt/sper
mgmt/spfeat1r
mgmt/spfeat2r
mgmt/spfeat3r
mgmt/spmscr
mgmt/spscr
mgmt/spter
mgmt/syncr
mgmt/tcsr
mgmt/tsfreqr
port_enable
port_select
power/autosuspend_delay_ms
power/control
power/runtime_active_time
power/runtime_status
power/runtime_suspended_time
traceid
uevent
wait4
```

8 EXAMPLE SESSIONS

8.1 Enabling STM trace via kernel driver

- CSWP_INIT
    - Client opens connection, sending client information. Server responds with server information
- CSWP_GET_DEVICES
    - Client discovers the device list with CSWP_GET_DEVICES. Server responds with device list:
        - /dev/mem
        - 20440000.A53_etm0
        - 20540000.A53_etm1
        - 20800000.etr

        - 20830000.etf
        - 20840000.etf
        - 20850000.etb
        - 20860000.stm
        - 208b0000.A53_funnel
        - 208c0000.main_funnel
        - etr_replicator@20890000
        - main_replicator@208a0000
- CSWP_DEV_OPEN, 3:
    - Client opens device 3 (ETR device)
- CSWP_DEV_OPEN, 7:
    - Client opens device 7 (STM device)
- CSWP_REG_LIST, 3
    - Client discovers register list for device 3
- CSWP_REG_LIST, 7
    - Client discovers register list for device 7
- CSWP_REG_WRITE, 3, 1
    - Write ETR enable_sink register
- CSWP_REG_WRITE, 7, 0
    - Write STM enable_source register
- CSWP_REG_READ, 7, 19
    - Read STM mgmt/tcsr register to get ATB ID
- CSWP_DEV_CLOSE, 7
    - Close STM device
- CSWP_DEV_CLOSE, 3
    - Close ETR device
- CSWP_TERM
    - Close connection

9 USB TRANSPORT

A device that implements CSWP using USB shall provide a USB interface descriptor with the following properties:

- Class 0xFF (vendor defined)
- String descriptor 0 starts with “CSWP”
- 1 bulk OUT endpoint
- 1 bulk IN endpoint

Clients using the USB transport for CSWP shall examine the interfaces provided by the USB device and locate the interface with class 0xFF, string descriptor 0 starting with CSWP and providing 1 bulk out endpoint and 1 bulk in endpoint. Commands shall be written to the bulk OUT endpoint and responses read from the bulk IN endpoint. The data written and read shall be the protocol as defined in this document – no transport specific wrapper is required.

10 USB TRACE

The USB trace interface supports multiple channels of trace allowing capture from multiple trace sinks (e.g. ETRs) and a metadata channel which carries status information. The interface has the following endpoints

Type	Direction	Description
INTR	IN	Metadata channel
BULK	IN	Trace data channel 1
BULK	IN	Trace data channel 2
…	…	…
BULK	IN	Trace data channel N

Note there may be fewer data channels than the number of trace sinks in the system. Each channel is attached to a trace sink with a control message.

10.1 Setup and configuration

A client may discover the available trace sinks in the system with the following setup request:

Field	Value
bmRequestType: Direction	1 (Device to Host)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	1 (Interface)
bRequest	0 (REQ_TRACE_SINK_INFO)
wValue	Index of the trace sink, whose name is requested
wIndex	Index of the Interface
wLength	Length of the result buffer supplied

The client should repeat this call, starting with wIndex 0 and incrementing on each call until an empty string is returned to indicate no more sinks are available,

To attach to a sink and collect trace, the client should send the following setup requests:

- Bind the endpoint to a sink

Field	Value
bmRequestType: Direction	0 (Host to Device)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	1 (REQ_STREAM_SET_SINK)
wValue	Index of the trace sink obtained from TRACE_SINK_INFO request
wIndex	EpNumber for the Stream
wLength	Length of the result buffer supplied (should be at least 2bytes)

- Set the timeout before partially filled buffers are sent to the host

Field	Value

bmRequestType: Direction	0 (Host to Device)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	3 (REQ_STREAM_SET_TX_TIMEOUT)
wValue	Timeout in milli seconds
wIndex	EpNumber for the Stream
wLength	Length of the result buffer supplied (should be at least 2bytes)

- Set the sink buffer size

Field	Value
bmRequestType: Direction	0 (Host to Device)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	5 (REQ_STREAM_SET_BUF_SIZE)
wValue	Size in unit of 4k blocks
wIndex	EpNumber for the Stream
wLength	Length of the result buffer supplied (should be at least 2bytes)

- Set the trace sink watermark level. This is controls when the sink will send an interrupt to the driver to trigger data transfer to the host and is specified by the amount of free space left in the buffer

Field	Value
bmRequestType: Direction	0 (Host to Device)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	7 (REQ_STREAM_SET_WATER_MARK)
wValue	Watermark in unit of 4k blocks
wIndex	EpNumber for the Stream
wLength	Length of the result buffer supplied (should be at least 2bytes)

- Attach the sink to the data channel. This will start data transfer to the host as the sink collects data

Field	Value
bmRequestType: Direction	0 (Host to Device)
bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	16 (REQ_STREAM_ATTACH_SINK)
wValue	(ignored)
wIndex	EpNumber for the Stream
wLength	(Ignored)

To detach from the sink after data collection is complete, the host should send the following setup request

Field	Value
bmRequestType: Direction	0 (Host to Device)

bmRequestType: Type	2 (Vendor)
bmRequestType: Recipient	2 (Endpoint)
bRequest	17 (REQ_STREAM_DETACH_SINK)
wValue	(ignored)
wIndex	EpNumber for the Stream
wLength	(Ignored)

10.2 Data channel

Data captured by a trace sink is sent over the data BULK endpoint bound to that sink by a REQ_STREAM_SET_SINK request. The data consists of 16-byte frames formatted as described in [CSARCH].

10.3 Status channel

The device will send the following messages over the metadata endpoint:

Type	Field	Value
uint16 LE	streamIdx	Stream number
uint16 LE	status	Status code
uint32 LE	value	Optional 32-bit value

The status code is one of:

Value	Name	Description
0	CS_STREAM_DETACHED	Detached from sink
2	CS_STREAM_DETACHING	Detaching from sink
3	CS_STREAM_ATTACHED	Attached to sink
4	CS_STREAM_PREPARE_SESSION	Trace capture started. Trace data will be sent over data endpoint
5	CS_STREAM_END_SESSION	Trace capture complete. All data delivered to host