ASAM

Association for Standardisation of
Automation and Measuring Systems

ASAM MCD-1 (XCP)

Universal Measurement and Calibration Protocol

Software Debugging over XCP

Version 1.0.0

Date: 2017-11-30

Associated Standard

Table of Contents

Foreword 5

1 Introduction 6

1.1 Motivation 6

1.2 Scope 7

1.3 Document Overview 8

2 Relations to Other Standards 9

2.1 References to Other Standards 9

3 Use Cases and Concepts 10

3.1 Standalone debugging 10

3.2 MC and debugging in parallel 10

3.3 Service Levels 11

3.4 Handling of target resources 11

3.5 Comparison of debug solutions 12

4 SW-DBG specific XCP Commands 14

4.1 Description of commands 15
4.1.1 Debugger Attach 17
4.1.2 Get Vendor Information 18
4.1.3 Get Debugging Properties 19
4.1.4 Get Target JTAG ID 21
4.1.5 Halt Target after Reset 22
4.1.6 Get HW-IO Pin Information 23
4.1.7 HW-IO Event Control 25
4.1.8 HW-IO Pin State and Control 26
4.1.9 Request Exclusive Target Interface Access 28
4.1.10 Processing of Multiple JPL Debug Sequences 29
4.1.11 Low Level Telegram 32
4.1.12 Read-Modify-Write 35
4.1.13 Download from Master to Slave 37
4.1.14 Download from Master to Slave (Block mode) 39
4.1.15 Download from Master to Slave (CAN – Part 1) 40
4.1.16 Download from Master to Slave (CAN – Part 2) 41
4.1.17 Download from Master to Slave (CAN – Part 3) 43
4.1.18 Upload from Slave to Master 44
4.1.19 Upload from Slave to Master (CAN – Part 1) 46
4.1.20 Upload from Slave to Master (CAN – Part 2) 47

5 Communication Error Handling 48

5.1 SW-DBG specific Error Codes 48

5.2 Error Code Handling 49

6 SW-DBG specific XCP Events 58
6.1 Change of Service Level 58
6.2 Change of HW-IO 59
7 XCP Sequence Examples 60
7.1 Setting up a session 61
7.2 Debug Attach 62
7.3 Get Debug Mode 62
7.4 Get JTAG ID 63
7.5 HW-IO Pin Handling 63
7.6 JPL Sequence 64
7.7 DAP over LLT 68
7.8 Memory read and write 70
8 Terms and Definitions 73
9 Symbols and Abbreviated Terms 74
10 Bibliography 75
Appendix: A. The JTAG Programming Language 76
A.1. JPL Command definitions 77
A.2. Examples 79
Appendix: B. Mapping between Low Level Telegram and Infineon DAP Telegram 82
Appendix: C. Explanation of Target Resource Identifier 85
C.1. Infineon DAP Devices 85
C.2. MPC5xxx/SPC5xxx PowerPC Devices 85
C.3. Renesas RH850 Devices 85
Figure Directory 87
Table Directory 88

Foreword

Software Debugging over XCP is an associated standard to the XCP Base Standard. It extends the XCP Protocol Layer Specification by software debugging features for electronic control units (ECU). For a variety of debugging use cases, these features allow a debugging tool to access and debug an ECU using an XCP communication channel, instead of connecting a debug probe to the debug interface of the ECU.

1 Introduction

1.1 Motivation

Software debugging as well as measurement data acquisition, calibration and data stimulation are essential techniques used during all stages of electronic control unit (ECU) development.

Sometimes it is impossible to attach a debug probe to the ECU for software debugging purposes, e.g.:

- a plug on device (POD) uses the debug interface exclusively for measurement, calibration or data stimulation
- a cooperative debug interface hardware arbitration mechanism between tools may not be applicable because of mechanical, electrical or interface protocol restrictions
- repeatedly switching between a debug probe and a POD may require a high amount of effort and lead to electrical or mechanical problems
- the debug interface may not be electrically or mechanically accessible

While the integrator prepares ECUs usually for measurement, calibration and data stimulation purposes even when mounted in the car, such preparations are seldom made for debugger access. Therefore, the customer uses internal or external XCP slaves instead. Besides their XCP base capabilities, many XCP slaves have access to debug resources within the ECU. For example, external XCP slaves, also called PODs, usually are physically attached to the debug interface of the ECU as shown in Figure 1.

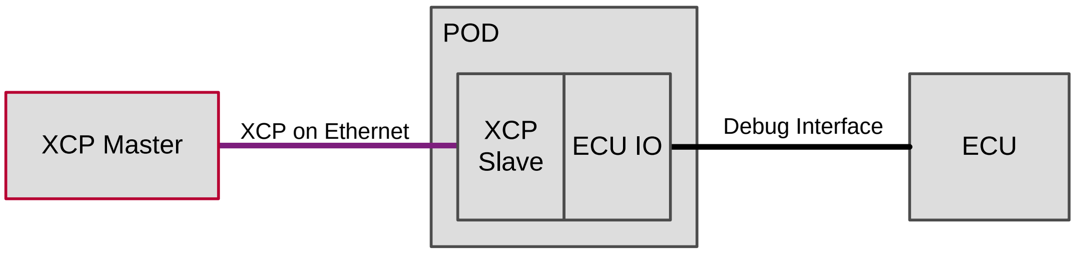

Figure 1 XCP Master accessing the ECU via a POD using XCP

In some microcontroller architectures, an internal XCP slave may also be capable to access ECU-internal debug resources as depicted in Figure 2. In such a scenario, the XCP master connects to the ECU via a vehicle bus system, e.g. CAN.

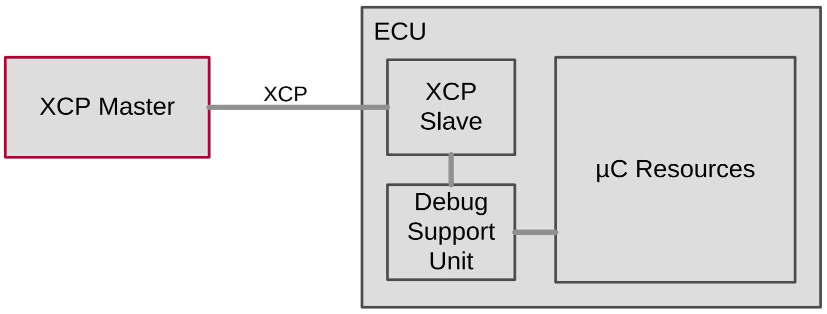

Figure 2 XCP Master directly connected to the ECU using XCP

The XCP Standard extension for software debugging has the following benefits:

- debugger software can use external XCP slaves (PODs) or internal XCP slaves to access the ECU for software debugging
    - this allows debugging of hardware that has not been accessible before
    - this will often eliminate the need to switch between debugger hardware and a POD during development
- besides exclusive access, the customer can debug the software in parallel to measurement, calibration and data stimulation by two separate tools as shown in the figure below

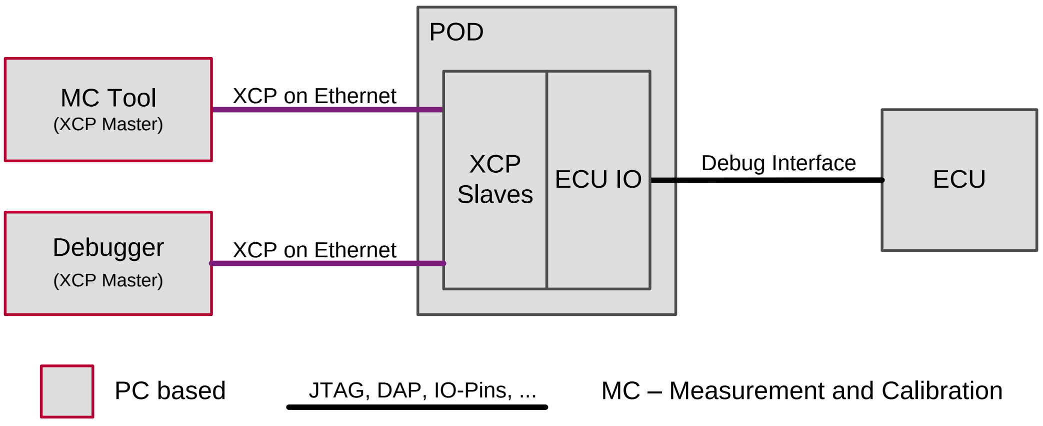

Figure 3 Tools accessing the ECU in parallel via a POD

- compared to the approach to share the debug interface between a POD and a debug probe using hardware arbitration, Software Debugging over XCP may allow a better system behavior, as arbitration is done on a higher level

1.2 Scope

The Software Debugging over XCP standard supports the same transport layers as the XCP Base Standard with some limitations regarding CAN.
Software Debugging over XCP does not require an AML extension. The debug XCP master handles all parameters by specific commands or by tool configuration.
The Software Debugging over XCP standard has some limitations in comparison to the capabilities of a debugger having exclusive access to the debug interface of the ECU. Details can be found in Chapter 3.

1.3 Document Overview

The document consists of the following main chapters:

Chapter 3 shows software debugging use cases covered by this standard and explains the necessary concepts.

Chapter 4 contains the specification of the additional XCP commands that are required for software debugging.

Chapter 5 specifies the error handling for the additional XCP commands.

Chapter 6 specifies Software Debugging over XCP specific XCP events which allow the XCP slave to asynchronously inform the XCP master of debugging-related state changes.

Chapter 7 shows exemplary XCP sequences typically used for ECU debugging.

2 Relations to Other Standards

2.1 References to Other Standards

ASAM e.V.: ASAM MCD-1 XCP V1.5

ASAM e.V.: ASAM MCD-1 POD V1.0

IEEE Std. 1149.1-2013: IEEE Standard for Test Access Port and Boundary-Scan Architecture

Infineon Technologies AG: AURIX OCDS User’s Manual V2.9.1

3 Use Cases and Concepts

3.1 Standalone debugging

One of the main use cases of the Software Debugging over XCP protocol will be the exclusive usage of the microcontroller debug interface for debugging purposes. In this case the debugger may use all of the microcontroller debug resources and the whole bandwidth of the debug interface. When the XCP slave operates in this mode, it notifies the debugger about this exclusive access state using the concept of service levels, see chapter 3.3.

3.2 MC and debugging in parallel

Another common use case is performing measurement, calibration, data stimulation (MC) - using the XCP resources CAL, DAQ, STIM, PGM - and debugging (DBG) in parallel. The Software Debugging over XCP standard allows realizing this use case for embedded XCP slaves as well as for external XCP slaves (PODs). In the latter scenario the debugger and the MC tool commonly use the debug interface as shown in Figure 4.

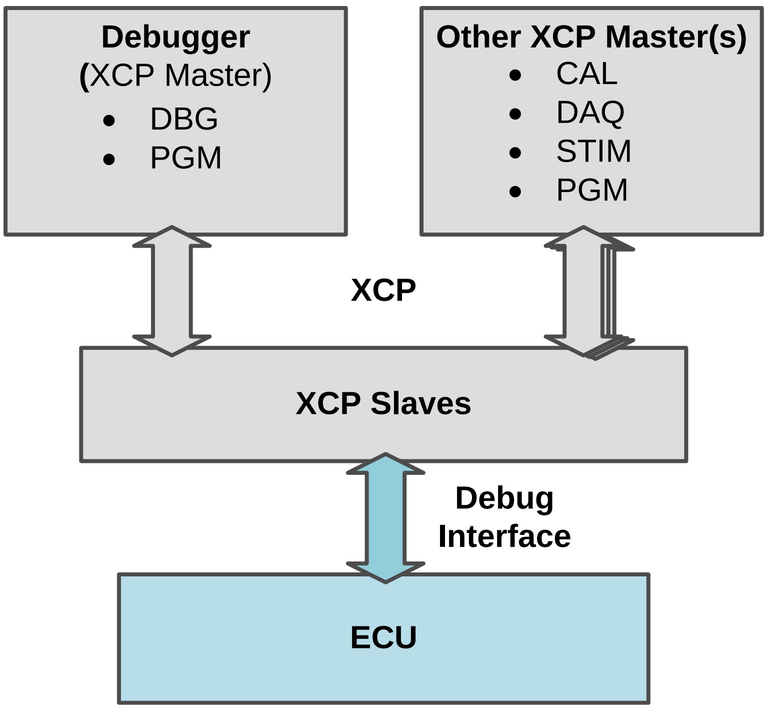

Figure 4 Debugging in parallel with other XCP tools

In this scenario the debugger and the measurement and calibration tool share the bandwidth of the debug interface and all the required microcontroller resources.
The debugger is notified about this state by means of the concept of service levels. Different service levels define the bandwidth distributions between the tools.

3.3 Service Levels

Service level 1 – Debugging not possible
This service level defines that the debug XCP master has no access to the target.

Service level 2 – Exclusive debugger access to target
This service level defines that the debug XCP master has exclusive access to the target.

Service level 3 – High bandwidth assigned to debugger
In this service level the focus lies on debugging rather than MC, e.g. when no fast measurement is active. The debug XCP master may execute long debug sequences and access ECU memory with high bandwidth.

Service level 4 – Low bandwidth assigned to debugger
In this service level the focus lies on MC rather than debugging, e.g. when a fast measurement or stimulation is running. In this state the debugger should not request ECU memory accesses with high bandwidth, e.g. for showing a memory window with high update rate. The debug XCP master should also limit the duration of any debug interface sequences to 100µs whenever possible.

3.4 Handling of target resources

Unless either the MC tool or the debugger has exclusive access to the debug interface, the tools share some of the ECU resources. In order to cooperate successfully in such a scenario, it is necessary for the debug XCP master as well as the XCP slave to know exactly which resources they are allowed to control.

In general, the related ECU resources can be split into three categories:

1. **Resources that are implicitly shared between the tools, e.g.:**
    - Physical debug interface access
    - Basic debug interface registers
    - ECU input pins (reset, break-in, watchdog disable, flash program enable)

2. **Resources that may need to be shared explicitly, e.g.:**
    - Shared overlay/trace memory
    - Core watchpoints (as measurement triggers and as trace triggers)
    - ECU output pins (break-out)

3. **Resources that typically are not shared, e.g.:**
    - Hardware trace interface
    - Dedicated trace memory
    - Dedicated calibration overlay memory
    - Calibration overlay registers
    - Core debug memory
    - Core breakpoints

The integrator is responsible to avoid any resource conflicts for shared resources. Therefore, the tools shall offer the possibility to configure the usage of these resources.

To easily accommodate different use case scenarios, the standard recommends to use mechanisms introduced with the ASAM POD Standard v1.0, see [2]. However, this applies foremost to the Measurement and Calibration (MC) XCP master since the debug XCP master typically does not offer A2L support.

3.5 Comparison of debug solutions

There are different solutions how a debugger, running on the host PC, connects with the target:

1. **Native Debugging:** a debugging software running on a host PC is connected via a debug probe to the target

2. **SW-DBG via POD:** a debug XCP master uses an external XCP slave for accessing the target

3. **SW-DBG via embedded XCP slave:** a debug XCP master connects to an embedded XCP slave for accessing the target. There are two scenarios that need to be considered:
    a. **XCP slave running on separate Core:** an XCP slave is running on a core that controls the debugging of other cores. E.g., an XCP slave is running on core 0 and handles debugging of core 4.
    b. **XCP slave running on same Core:** an XCP slave is running on a core which will be debugged. Typically this is the case in small devices, which may have only one core.

Table 1 Comparison of debug solutions

Feature	Native Debugging	SW-DBG via POD	SW-DBG via embedded XCP slave	
			XCP slave running on separate Core	XCP slave running on same Core
PORST	+	+	-	-
Software-Reset	+	+	+	+
Breakpoint which halts a core	+	+	(+)	-
Breakpoint which jumps to a monitoring program	+	+	(+)	(+)
r/w memory while running	+	+	(+)	(+)
r/w memory while halted	+	+	(+)	-
r/w SFR while running	+	+	(+)	(+)
r/w SFR while halted	+	+	(+)	-

Feature	Native Debugging	SW-DBG via POD	SW-DBG via embedded XCP slave	
			XCP slave running on separate Core	XCP slave running on same Core
r/w CPU Register while halted	+	+	(+) not on Core running XCP	-
Real-time watch window (e.g. refresh rate 1ms)	+	(+) refresh-speed may be slower	(+) refresh-speed may be slower	(+) refresh-speed may be slower
OnChip-Trace to RAM (ED-RAM or PD-RAM) ->Configure trace unit, upload result	+	(+) resource-management required	(+) resource-management required	(+) resource-management required
External-Trace (Aurora) ->Streaming	+	(+)	(+)	(+)
Brain-dead ECU flash programming	+	+	-	-

Legend: + fully supported, (+) supported with restrictions, - not supported

4 SW-DBG specific XCP Commands

All SW-DBG related XCP commands are grouped within a specific command space, which is part of the XCP "Level 1" commands. The following table shows the definition of the SW-DBG specific command space.

Table 2 Definition of the SW-DBG specific XCP command space

Position	Type	Description
0	BYTE	XCP level 1 command space extension = 0xC0
1	BYTE	XCP command space reserved for SW-DBG = 0xFC
2	BYTE	SW-DBG command

An overview of all SW-DBG related commands is given in the table below.

Table 3 SW-DBG command overview

SW-DBG Command	SW-DBG Code	CAN compatible	Support
DBG_ATTACH	0x00	yes	mandatory
DBG_GET_VENDOR_INFO	0x01	yes	mandatory
DBG_GET_MODE_INFO	0x02	yes	mandatory
DBG_GET_JTAG_ID	0x03	yes	mandatory
DBG_HALT_AFTER_RESET	0x04	yes	optional
DBG_GET_HWIO_INFO	0x05	yes	optional
DBG_SET_HWIO_EVENT	0x06	yes	optional
DBG_HWIO_CONTROL	0x07	yes	optional
DBG_EXCLUSIVE_TARGET_ACCESS	0x08	yes	mandatory
DBG_SEQUENCE_MULTIPLE	0x09	irrelevant	mandatory for JTAG targets
DBG_LLT	0x0A	irrelevant	mandatory for DAP targets
Continued on next page			

SW-DBG Command	SW-DBG Code	CAN compatible	Support
DBG_READ_MODIFY_WRITE	0x0B	no	mandatory *
DBG_WRITE	0x0C	no	mandatory *
DBG_WRITE_NEXT	0x0D	no	optional *
DBG_WRITE_CAN1	0x0E	yes	mandatory *
DBG_WRITE_CAN2	0x0F	yes	mandatory *
DBG_WRITE_CAN_NEXT	0x10	yes	mandatory *
DBG_READ	0x11	no	mandatory *
DBG_READ_CAN1	0x12	yes	mandatory *
DBG_READ_CAN2	0x13	yes	mandatory *

* depending on `MAX_CTO_DBG`, for details see command `DBG_ATTACH`.

4.1 Description of commands

The length of CTOs related to SW-DBG shall not be limited by the length of CTOs of the Base Standard. For that reason, `MAX_CTO_DBG` defines the maximum length of a SW-DBG related CTO in bytes. Table 7 shows the resulting structure for the different packet types:

Table 4 SW-DBG command structure

Position	Type	Description
0	BYTE	XCP level 1 command space extension = 0xC0
1	BYTE	XCP command space reserved for SW-DBG = 0xFC
2.. `MAX_CTO_DBG-1`	BYTE	SW-DBG command data

Table 5 SW-DBG command response packet structure

Position	Type	Description
0	BYTE	Packet Identifier = 0xFF
1.. `MAX_CTO_DBG-1`	BYTE	Optional command response data

Table 6 SW-DBG error packet structure

Position	Type	Description
0	BYTE	Packet Identifier = 0xFE
1	BYTE	Error code = ERR_DBG
2	BYTE	SW-DBG specific error code
3.. `MAX_CTO_DBG-1`	BYTE	Optional error information data

Table 7 SW-DBG event packet structure

Position	Type	Description
0	BYTE	Packet Identifier = 0xFD
1	BYTE	SW-DBG event = 0xFC
2	BYTE	SW-DBG specific event code
3.. `MAX_CTO_DBG-1`	BYTE	Optional event subcode specific data

General remarks:

- unused data bytes, marked as „reserved”, must be set to 0. Alike, unused bits, marked as “X” in bit field tables, must be set to 0.
- any of the SW-DBG commands may return `ERR_CMD_BUSY` if the XCP slave cannot process the command temporarily
- any SW-DBG command that requires access to the target may respond `ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE` if the target cannot be accessed temporarily
- The debug XCP master shall send the `DBG_ATTACH` command prior to any other SW-DBG specific command. If the XCP slave receives a debug command prior to a `DBG_ATTACH` command it shall respond with the error `ERR_DBG_ATTACH_MISSING`.

4.1.1 Debugger Attach

Category	mandatory
Target access required	no
Mnemonic	DBG_ATTACH

Table 8 DBG_ATTACH command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x00

This command returns detailed information about the implemented version of the SW-DBG feature of the XCP slave.

The debug XCP master shall send this command prior to any other SW-DBG specific commands and must not send any other debug commands before it received a positive reply. An XCP slave may use this information for assigning the debug flag during ECU authentication as defined in the POD standard [2].

Positive Response:

Table 9 DBG_ATTACH response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFF
1	BYTE	Major version of Software Debugging over XCP, i.e. 1
2	BYTE	Minor version of Software Debugging over XCP, i.e. 0
3	BYTE	Timeout t_1 – resulting value: *n**2 ms
4	BYTE	Timeout t_7 – resulting value: *n**2 ms
5	BYTE	Reserved
6	WORD	`MAX_CTO_DBG`

Debug XCP masters do typically not process A2L files. For this reason, relevant timeout parameters, as defined in the Base Standard [1], are made available to a debug XCP master in the DBG_ATTACH response.

The `MAX_CTO_DBG` parameter allows defining a maximum length in bytes specifically related to CTO packets of SW-DGB. XCP slaves which communicate with the debug XCP master over Ethernet may benefit from increased CTO packet size.

Negative Response

If the debug resource (DBG) is locked the XCP slave shall return `ERR_ACCESS_LOCKED`. Since any other SW-DBG related command requires successful execution of the `DBG_ATTACH` command, only `DBG_ATTACH` shall return `ERR_ACCESS_LOCKED`.

4.1.2 Get Vendor Information

Category	mandatory
Target access required	no
Mnemonic	DBG_GET_VENDOR_INFO

Table 10 DBG_GET_VENDOR_INFO command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x01

This command returns the XCP slave vendor as well as vendor specific information.

Positive Response:

Table 11 DBG_GET_VENDOR_INFO response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFF
1	BYTE	Length of XCP slave vendor specific information (N)
2	WORD	XCP slave vendor identification (as defined by the POD standard [2])
4… 4+N-1	BYTES	XCP slave vendor specific information

4.1.3 Get Debugging Properties

Category	mandatory
Target access required	no
Mnemonic	DBG_GET_MODE_INFO

Table 12 DBG_GET_MODE_INFO command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x02

This command returns detailed debugging properties from the XCP slave.

<u>Positive Response:</u>

Table 13 DBG_GET_MODE_INFO response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFF
1	BYTE	Reserved
2	BYTE	`MAX_HW_IO_PINS:` number of available HW-IO pins, see `DBG_GET_HWIO_INFO`; shall be 0 if `DBG_GET_HWIO_INFO` is not supported
3	BYTE	Dialect – see Table 14 Dialect used for low level target access, depends on the target's debug interface.
4	BYTE	Feature - see Table 15
5	BYTE	Service level – see Table 17

Table 14 Dialect parameter coding

ID	Description
0x00	Low level target access not available
0x01	JTAG via DBG_SEQUENCE_MULTIPLE (JPL)
0x02	Infineon DAP via DBG_LLT
Others	Reserved

Further dialect specific protocol information is not provided.

Table 15 Feature parameter bitmask structure

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
X	X	X	X	X	X	X	HAR

Table 16 Feature parameter bitmask coding

Flag	Description
HAR	Halt-After-Reset command 0: Halt-after-Reset command is not supported 1: Halt-after-Reset command is supported

Table 17 Service level parameter coding

ID	Description
0x00	Service level 1: No access to target interface for SW-DBG
0x01	Service level 2: Exclusive access to target interface for SW-DBG
0x02	Service level 3: SW-DBG and other XCP slaves share the target interface, high bandwidth share for SW-DBG
0x03	Service level 4: SW-DBG and other XCP slaves share the target interface, limited bandwidth for SW-DBG, duration of native target access (e.g. JPL, LLT) should not exceed 100µs
Others	Reserved

The service level defines the arbitration of the target interface between SW-DBG and other XCP masters. The POD system sets the service level depending on the target capabilities and the MC configuration. Thus, the service level might change during run-time. To indicate such a situation to the debug XCP master the `EV_DBG_SERVICE_LEVEL_CHANGE` event is used. Further information can be found in chapter 3.3.

4.1.4 Get Target JTAG ID

Category	mandatory
Target access required	no
Mnemonic	DBG_GET_JTAG_ID

Table 18 DBG_GET_JTAG_ID command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x03

The XCP slave shall respond the JTAG ID of the target, i.e. the production device. The XCP slave should obtain the JTAG ID of the target as early as possible.

Positive Response:

Table 19 DBG_GET_JTAG_ID response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFF
1	BYTE	Reserved
2	BYTE	Reserved
3	BYTE	Reserved
4	DWORD	Target’s JTAG ID

Negative Response:

If, for whatever reason, the JTAG ID should not be available, the XCP slave shall return `ERR_GENERIC`.

4.1.5 Halt Target after Reset

Category	optional
Target access required	yes
Mnemonic	DBG_HALT_AFTER_RESET

Table 20 DBG_HALT_AFTER_RESET command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x04

`DBG_HALT_AFTER_RESET` performs a power-on-reset of the target and halts the target at the earliest possible point after the reset is de-asserted.

Negative Response:

If the XCP slave does not support the command, the XCP slave shall return the error `ERR_CMD_UNKNOWN`.

If reset was carried out but the execution of the XCP slave internal sequence for halting the target failed, the XCP slave shall responds with the following response:

Table 21 DBG_HALT_AFTER_RESET negative response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_DBG
2	BYTE	ERR_DBG_HALT_AFTER_RESET

4.1.6 Get HW-IO Pin Information

Category	optional
Target access required	no
Mnemonic	DBG_GET_HWIO_INFO

Table 22 DBG_GET_HWIO_INFO command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x05
3	BYTE	Index

`DBG_GET_HWIO_INFO` provides information about the HW-IO pins which are supported by the XCP slave. The XCP slave shall implement the command for `MAX_HW_IO_PINS` larger than 0 (see Table 13).

The positive response will return as many available HW-IO pins as possible, beginning with the given index.

If the response message size is too small to return all HW-IO pins (depending on `MAX_CTO_DBG`), the debug XCP master shall use additional queries with appropriate indexes to get the information about all HW-IO pins.

The response parameter *N* specifies the number of returned HW-IO pins in the response message.

Positive Response:

Table 23 DBG_GET_HWIO_INFO response structure

Position	Type	Description	
0	BYTE	Packet ID = 0xFF	
1	BYTE	Number of HW-IO pins in this response message (N)	
2	BYTE	Index used to identify the pin	PIN 1
3	BYTE	Mode, see Table 24	
4	BYTE	Class of the pin: 0x00: Vref 0x01: PORST 0x02: Secondary reset 0x03: Break (In, out), EVT(I,O) 0x04: Watchdog disable 0x05: Flash programming 0x06: TRSTothers = reserved	
Continued on next page			

Position	Type	Description	
5	BYTE	Active state Defines the active state of the pin. Information only. 0x00 = unknown 0x01 = low 0x02 = high 0x03 = clock others = reserved	
6	BYTE	Index	PIN 2
7	BYTE	Mode	
8	BYTE	Class	
9	BYTE	Active State	
..			
2+(N-1)*4	BYTE	Index	PIN N
3+(N-1)*4	BYTE	Mode	
4+(N-1)*4	BYTE	Class	
5+(N-1)*4	BYTE	Active State	

If there are several pins of the same class, the index parameter shall be in the order of the pin index of the target.

Table 24 Mode parameter bitmask structure

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
X	X	X	Pin triggers an event	X	X	XCP slave output	XCP slave input

After a connection setup (issued by command `CONNECT`, see [1]) the XCP slave disables the pin related event generation. A debug XCP master must enable event generation per pin. Trigger conditions are given in Table 25.

Negative Response:

For an unsupported value of the index the XCP slave shall respond with the error `ERR_OUT_OF_RANGE`.

4.1.7 HW-IO Event Control

Category	optional
Target access required	no
Mnemonic	DBG_SET_HWIO_EVENT

Table 25 DBG_SET_HWIO_EVENT command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x06
3	BYTE	Pin index (see Table 23)
4	BYTE	Trigger condition 0 = transition to low level on the signal attached to the pin 1 = transition to high level on the signal attached to the pin 2 = pin changed 255 = disable event others = reserved

Using the `DBG_SET_HWIO_EVENT` command the debug XCP master can configure for which state change of an IO pin the XCP slave shall generate an XCP Event.

The command shall be implemented by the XCP slave for `MAX_HW_IO_PINS` larger than 0 (see Table 13).

Negative Response:

For an illegal parameter value the XCP slave shall respond with the error `ERR_OUT_OF_RANGE`.

If the debug XCP master tries to enable an event for a pin which does not support event generation the XCP slave shall respond the error `ERR_GENERIC`.

4.1.8 HW-IO Pin State and Control

Category	optional
Target access required	no
Mnemonic	DBG_HWIO_CONTROL

Table 26 DBG_HWIO_CONTROL command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x07
3	BYTE	Number of pins in this command
4	BYTE	Pin index$_n$
5	BYTE	State Set the pin to the desired physical state: 0 = low - the XCP slave shall ensure a low level on the signal attached to this pin 1 = high - the XCP slave shall ensure a high level on the signal attached to this pin 2 = leave pin unchanged 3 = tri-state - the XCP slave shall ensure that the pin is not actively driven any more 4 = clock, frequency in Hz others = reserved
6	WORD	Clock frequency
8	BYTE	Pin index$_m$
9	BYTE	State
10	WORD	Clock frequency
..		

`DBG_HWIO_CONTROL` allows to read and/or modify the state of pins. The slave shall at first perform the requested state changes and afterwards read the current state of the pins.

The command shall be implemented by the XCP slave for `MAX_HW_IO_PINS` larger than 0 (see Table 13).

<u>Positive Response:</u>

The positive response shall return the result for each pin requested by the master. For pins with output direction only, the XCP slave shall return the currently configured value. For all other pins the XCP slave shall return the physical state.

The state of the pins shall only be modified if the command does not lead to an error.

Table 27 DBG_HWIO_CONTROL response structure

Position	Type	Description	
0	BYTE	Packet ID = 0xFF	
1	BYTE	Current state of pin 0 = low 1 = high 3 = tri-state 4 = clock others = reserved	Pin index$_n$
2	BYTE	Current state of pin	Pin index$_m$
..			

<u>Negative Response:</u>

For any error during state change of a pin the XCP shall return the following error:

Table 28 DBG_HWIO_CONTROL negative response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_DBG
2	BYTE	ERR_DBG_HWIO_CONTROL
3	BYTE	Pin index of first failing pin

4.1.9 Request Exclusive Target Interface Access

Category	mandatory
Target access required	yes
Mnemonic	DBG_EXCLUSIVE_TARGET_ACCESS

Table 29 DBG_EXCLUSIVE_TARGET_ACCESS command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x08
3	BYTE	Mode
4	BYTE	Context

Table 30 Mode parameter coding

ID	Description
0x00	Request exclusive target interface access
0x01	Release exclusive target interface access
Others	Reserved

Table 31 Context parameter coding

ID	Description
0x00	Generic
0x01	Non-volatile target memory programming
Others	Reserved

A debug XCP master might request exclusive access to the target interface prior to sending a sequence of commands that directly access the target interface. In this case the XCP slave shall not carry out other activities on the target interface than those caused by the debug XCP master.

Negative Response:

In case that the XCP slave temporarily cannot grant exclusive access to the target, the XCP slave shall respond with the error `ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE`.

4.1.10 Processing of Multiple JPL Debug Sequences

Category	mandatory for JTAG targets
Target access required	yes
Mnemonic	DBG_SEQUENCE_MULTIPLE

Table 32 DBG_SEQUENCE_MULTIPLE command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x09
3	BYTE	Mode
4	WORD	Number of sequences in this frame.
The following structure is repeated “Number of Sequences” times.		
6	WORD	Number of bytes (N) of all JPL commands of this sequence (count starts with byte 8).
8.. (8+(N-1))	BYTES	JPL commands
8+N	BYTE	Optional padding byte for alignment of next sequence (if *N* is odd)

Table 33 Mode parameter bitmask structure

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
X	X	X	X	TDI initialization value	TMS initialization value	JTAG bus release at end of command	JTAG bus initialization at start of command

DBG_SEQUENCE_MULTIPLE executes multiple debug sequences. The XCP slave shall execute the sequences atomically, i.e. the XCP slave shall not interleave other activities on the target interface.

The debug XCP master shall set bit 0 of the mode parameter in the first DBG_SEQUENCE_MULTIPLE command of a sequence of DBG_SEQUENCE_MULTIPLE commands (a sequence consists of one or more DBG_SEQUENCE_MULTIPLE commands). This bit leads to a JTAG bus request. The bus ownership is kept until the bus is explicitly released (see bit 1). The JTAG signals shall be changed without generating TCK clock edges. TCK shall be high before requesting the bus. The initialization levels of the TMS and TDI signals are given in bit 2 and bit 3 respectively.

The XCP slave has to ensure that the JTAG state machine is in the run test idle state before executing the JPL commands.

The debug XCP master shall set bit 1 of the mode parameter in the last `DBG_SEQUENCE_MULTIPLE` command of a sequence of `DBG_SEQUENCE_MULTIPLE` commands.

If the debug XCP master sets bit 1 of the mode parameter, the debug XCP master shall ensure that the JTAG state machine enters the run test idle state at the end of the JPL sequence of this command.

The XCP slave shall not interleave any other command that requires target access between a JTAG bus request and a JTAG bus release.

The number of JPL commands and data per XCP message depends on `MAX_CTO_DBG`.

Positive Response:

Table 34 DBG_SEQUENCE_MULTIPLE positive response structure

Position	**Type**	**Description**
0	BYTE	Packet ID = 0xFF
1	BYTE	Reserved
2	WORD	Number of results (NR)
The following structure is repeated NR times		
$4 + 6 \times (NR - 1)$	BYTE	Execution Status 0x00: Sequence executed successfully others = reserved
$5 + 6 \times (NR - 1)$	BYTE	Repeat Count Repeat count after execution of last JTAG command sequence
$6 + 6 \times (NR - 1)$	BYTE	TDO_BIT_24_31 Bit 24…31 of last TDO data after last JPL data command
$7 + 6 \times (NR - 1)$	BYTE	TDO_BIT_16_23 Bit 16…23 of last TDO data after last JPL data command
$8 + 6 \times (NR - 1)$	BYTE	TDO_BIT_8_15 Bit 8…15 of last TDO data after last JPL data command
$9 + 6 \times (NR - 1)$	BYTE	TDO_BIT_0_7 Bit 0…7 of last TDO data after last JPL data command

Negative Response:

For an illegal mode parameter value the XCP slave shall respond with the error `ERR_OUT_OF_RANGE`.

If bit 0 of the mode parameter is no set in the first `DBG_SEQUENCE_MULTIPLE` command of a sequence of `DBG_SEQUENCE_MULTIPLE` commands the XCP slave shall respond with the error `ERR_SYNTAX`.

In case of a JPL related error, the XCP slave shall respond the following error:

Table 35 DBG_SEQUENCE_MULTIPLE negative response structure

Position	**Type**	**Description**
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_DBG
2	BYTE	ERR_DBG_JPL
3	BYTE	Reserved
4	WORD	Number of results in negative response (NNR)
The following structure is repeated NNR times		
$6 + 6 \times (NNR - 1)$	BYTE	Execution Status 0x00: Sequence executed successfully 0x01: Incompatible dialect. The XCP slave shall not execute the sequence. 0x02: Syntax error in JPL sequence. The XCP slave shall abort the execution of the sequence. 0x03: JPL repeat-count exceeded. The XCP slave shall execute the whole sequence even in case of this error. 0x04: Low level target interface error. The XCP slave shall abort the execution of the sequence. others = reserved
$7 + 6 \times (NNR - 1)$	BYTE	Repeat Count Repeat count after execution of last JTAG command sequence
$8 + 6 \times (NNR - 1)$	BYTE	TDO_BIT_24_31 Bit 24…31 of last TDO data after last data transfer command
$9 + 6 \times (NNR - 1)$	BYTE	TDO_BIT_16_23 Bit 16…23 of last TDO data after last data transfer command
$10 + 6 \times (NNR - 1)$	BYTE	TDO_BIT_8_15 Bit 8…15 of last TDO data after last data transfer command
$11 + 6 \times (NNR - 1)$	BYTE	TDO_BIT_0_7 Bit 0…7 of last TDO data after last data transfer command

4.1.11 Low Level Telegram

Category	mandatory for DAP targets
Target access required	yes
Mnemonic	DBG_LLT

Table 36 DBG_LLT command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0A
3	BYTE	Number of Low Level Telegrams (LLT)
4	BYTE	Mode valid for all telegrams of this CTO
5	BYTE	LLT *1* – Expected response length in bits
6	BYTE	LLT *1* – CMD
7	BYTE	LLT *1* – LEN_{LLT1} in bits
8	BYTE	LLT *1* – DATA 0-7, LSBit first
..	BYTE	LLT *1* – DATA
8+$\lceil LEN_{LLT1}/8\rceil - 1$	BYTE	LLT *1* – DATA .. to LEN_{LLT1}-1, optionally zero padded
8+$\lceil LEN_{LLT1}/8\rceil$	BYTE	LLT *2* – Expected response length in bits
8+$\lceil LEN_{LLT1}/8\rceil + 1$	BYTE	LLT *2* – CMD
8+$\lceil LEN_{LLT1}/8\rceil + 2$	BYTE	LLT *2* – LEN_{LLT2} in bits
8+$\lceil LEN_{LLT1}/8\rceil + 3$	BYTE	LLT *2* – DATA 0-7, LSBit first
..	BYTE	LLT *2* – DATA
8+$\lceil LEN_{LLT1}/8\rceil + 3 + \lceil LEN_{LLT2}/8\rceil - 1$	BYTE	LLT *2* – DATA .. to LEN_{LLT2}-1, optionally zero padded
8+$\lceil LEN_{LLT1}/8\rceil + 3 + \lceil LEN_{LLT2}/8\rceil$	BYTE	Begin of LLT *3*
..	..	

The Mode parameter is dialect specific and is described in the related dialect section of Appendix: B.

The Low Level Telegram (LLT) command allows exchanging target debug interface specific telegrams in a generic format between the debug XCP master and an XCP slave. The format is independent from the actually used physical interconnection on the target debug interface, e.g. the number of wires. The XCP slave is responsible to transform the Low Level Telegrams into target debug interface specific telegrams. In difference to the JPL approach, the debug XCP master may not be aware of the physical communication mode on the target debug interface as used by the XCP slave. The mapping between the Low Level Telegram and a target debug interface specific telegram is described in Appendix: B.

An XCP slave shall meet with the following requirements:

- The XCP slave shall execute all Low Level Telegrams of a LLT command in an atomic context. I.e. on the target debug interface these Low Level Telegrams shall not be interleaved with other telegrams (e.g. generated by the XCP slave itself).
- If Low Level Telegram *n* leads to an error, the XCP slave shall not execute all successor Low Level Telegrams.
- For Low Level Telegrams with a response length of 0, the byte describing the Expected Response Length shall be part of the positive response

This command supports the following dialects (see Table 14):

- Infineon DAP

Positive Response:

Table 37 DBG_LLT positive response

Position	**Type**	**Description**
0	BYTE	Packet ID = 0xFF
1	BYTE	Number of Low Level Telegram Responses
2	BYTE	LLT *1* – Expected Response Length (ERL_{LLT1}) in bits
3	BYTE	LLT *1* – Data bits 0-7 of response, LSBit first Data bits are only sent if respective ERL > 0
$3+\lceil ERL_{LLT1}/8\rceil - 1$	BYTE	LLT *1* – Data .. to ERL_{LLT1}-1, optionally zero padded
$3+\lceil ERL_{LLT1}/8\rceil$	BYTE	LLT *2* – Expected Response Length (ERL_{LLT2}) in bits Data bits are only sent if respective ERL > 0
$3+\lceil ERL_{LLT1}/8\rceil + 1$	BYTE	LLT *2* – Data bits 0-7 of response, LSBit first
$3+\lceil ERL_{LLT1}/8\rceil + 1 + \lceil ERL_{LLT2}/8\rceil - 1$	BYTE	LLT *2* – Data .. to ERL_{LLT2}-1, optionally zero padded
..	..	Begin of response of LLT *3*

Negative Response:

In case that a Low Level Telegram cannot be transferred into a valid target debug command `ERR_CMD_SYNTAX` shall be returned.

In case that the execution of the LLT command failed, the error response as given below shall be returned:

Table 38 DBG_LLT negative response

Position	**Type**	**Description**
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_DBG
2	BYTE	ERR_DBG_LLT
3	BYTE	Index of failing command, starting with 1
4	BYTE	Reason

Table 39 Reason parameter coding

ID	Description
0x00	Target access failed during LLT execution
0x01	CRC error
0x02	Timeout error
Others	Reserved

4.1.12 Read-Modify-Write

Category	Mandatory for `MAX_CTO_DBG` >= 32
Target access required	Yes
Mnemonic	DBG_READ_MODIFY_WRITE

Table 40 DBG_READ_MODIFY_WRITE command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0B
3	BYTE	Reserved
4	BYTE	Target Resource Identifier (TRI)
5	BYTE	Element width (EW) – see Table 41
6	WORD	Reserved
8	DLONG	Address
16.. 16+[EW-1]	ELEMENT	Mask
16+[EW].. 16+[2*EW-1]	ELEMENT	Data

Table 41 Element Width parameter coding

Value	Element Description
0x01	BYTE
0x02	WORD
0x04	DWORD
0x08	DLONG
Others	Reserved

There might be situations in which several instances need to operate on one target register. E.g., a debug XCP master might control bits in the registers that are dedicated to target debugging functionality while a POD controls other bits that are dedicated to MC functionality. In such scenarios there must be a method to prevent race conditions that could occur when several instances try to modify the register simultaneously.

The `DBG_READ_MODIFY_WRITE` command addresses such a use case by defining a read-modify-write operation. The XCP slave shall execute the operation in an atomic sequence. The operation shall be (given in C-like syntax):

```
    *Address = (*Address & (~Mask)) | (Data & Mask)
```

Positive Response:

Table 42 DBG_READ_MODIFY_WRITE positive response structure

Position	**Type**	**Description**
0	BYTE	Packet ID = 0xFF
..	BYTEs	Reserved: used for alignment; only if EW > 0x01
[EW]..[2*EW-1]	ELEMENT	Final register value

Negative Response:

In case of address misalignment or if the element width is not supported, the XP slave shall respond with the error `ERR_OUT_OF_RANGE`.

In case that the XCP slave does not support the TRI value, the XCP slave shall respond with the error `ERR_DBG_TRI_UNSUPPORTED`.

In case that the XCP slave does not support the element width, the XCP slave shall respond with the error `ERR_DBG_EW_UNSUPPORTED`.

In case of a bus error the XCP slave shall respond with the error `ERR_DBG_BUS_ERROR`.

4.1.13 Download from Master to Slave

Category	Mandatory for MAX_CTO_DBG >= 32
Target access required	yes
Mnemonic	DBG_WRITE

Table 43 DBG_WRITE command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0C
3	BYTE	Reserved
4	BYTE	Target Resource Identifier (TRI)
5	BYTE	Element width (EW) - see Table 41
6	WORD	Number of data elements (N) Standard mode: [1..(MAX_CTO_DBG-16)/EW] Block mode: [1..(MAX_BS*(MAX_CTO_DBG-8)-8)/EW]
8	DLONG	Byte address
16.. 15+N*EW	ELEMENTs	Data elements to write

The DBG_WRITE command writes a number of elements to the ECU starting at Byte Address. The execution of this command shall always modify the physical memory.

The XCP slave does not guarantee a cache coherent behavior. I.e. if the commands writes a physical memory which is cached by at least one core, the write access to the physical memory does not always lead to an update of the cache.

The Target Resource Identifier (TRI) specifies the interpretation of the byte address. This parameter is target specific. Details are given in Appendix: C.

The DBG_WRITE command is similar to the DOWNLOAD and SHORT_DOWNLOAD commands defined in the XCP base standard. While the XCP slave might perform an address translation for the DOWNLOAD and SHORT_DOWNLOAD commands the address given by the DBG_WRITE command is a physical target address. I.e. the XCP slave shall not modify the address given by this command in any way.

If the XCP slave does not support block transfer mode, all downloaded data are transferred in a single command packet. Therefore, the number of data elements parameter in the request has to be in the range [1..(MAX_CTO_DBG-16)/EW]. The XCP slave shall respond with the error ERR_OUT_OF_RANGE, if the number of data elements is more than (MAX_CTO_DBG-16)/EW.

After receiving a DBG_WRITE command the XCP slave first has to check whether there are enough resources available in order to cover the complete write request. If the XCP slave does not have enough resources, it shall respond with the error ERR_MEMORY_OVERFLOW and shall not execute any single write request. If the XCP slave rejects a DBG_WRITE request, it shall not change the target memory.

If the XCP slave supports block transfer mode, the downloaded data are transferred in multiple command packets. For the XCP slave however, there might be limitations concerning the maximum number of consecutive command packets (block size MAX_BS). Therefore the number of data elements can be in the range [1..min(65535, (MAX_BS*(MAX_CTO_DBG-8)-8)/EW)].
Without any error, the XCP slave will acknowledge only the last DBG_WRITE_NEXT command packet. The separation time between the command packets (MIN_ST) and the maximum number of packets (MAX_BS) are specified in the response of the GET_COMM_MODE_INFO command (see [1]).
If the XCP slave detects an internal problem during a block mode transfer, it shall send a negative response at once. If block transfer mode is requested and not enough resources are available, the XCP slave shall send the negative response code already after the initial DBG_WRITE command of the debug XCP master.
If the DBG_WRITE does not allow transmitting the complete data size this command shall sent ((MAX_CTO_DBG-16)/EW) data elements.

Furthermore, the following requirements also shall be met:

- the address shall be aligned according to the element width
- write access to the target shall be performed in the granularity given by the element width parameter

<u>Negative Response:</u>

In case of address misalignment, the XCP slave shall respond with the error ERR_OUT_OF_RANGE.

If the number of data elements does not match the expected value, the XCP slave shall respond with the error ERR_OUT_OF_RANGE.

In case that the element width is not supported, the XCP slave shall respond with the error ERR_DBG_EW_UNSUPPORTED.

In case that the TRI is not supported, the XCP slave shall respond with the error ERR_DBG_TRI_UNSUPPORTED.

In case of a bus error, the XCP slave shall respond with the error ERR_DBG_BUS_ERROR.

4.1.14 Download from Master to Slave (Block mode)

Category	Mandatory for MAX_CTO_DBG >= 32
Target access required	Yes
Mnemonic	DBG_WRITE_NEXT

Table 44 DBG_WRITE_NEXT command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0D
3	BYTE	Reserved
4	WORD	Number of remaining data elements (N)
6	WORD	Reserved
8.. 7+N*EW	ELEMENTs	Data elements to write

The debug XCP master shall use DBG_WRITE_NEXT if the MAX_CTO_DBG does not allow transmitting all data elements with DBG_WRITE. That means, depending on the number of data elements and MAX_CTO_DBG, zero, one or more than one DBG_WRITE_NEXT commands can follow a DBG_WRITE.

The debug XCP master shall first send all data elements to the XCP slave using DBG_WRITE_NEXT before sending any other command.

If the data transfer needs more than one DBG_WRITE_NEXT command, all but the last shall send ((MAX_CTO_DBG-8)/EW) elements. The last message sends the remaining elements.

Negative Response:

If the XCP slave does not support master block mode, see command GET_COMM_MODE_INFO as described in [1], the XCP slave shall respond with the error ERR_CMD_UNKNOWN.

If the first DBG_WRITE_NEXT command of an entire data transfer is not preceded by command DBG_WRITE, the XCP slave shall respond with the error ERR_SEQUENCE.

If the number of data elements does not match the expected value, the XCP slave shall respond with the error ERR_SEQUENCE. The negative response will contain the expected number of data elements.

Table 45 DBG_WRITE_NEXT negative response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_SEQUENCE
2	WORD	Number of expected data elements

4.1.15 Download from Master to Slave (CAN – Part 1)

Category	Mandatory for MAX_CTO_DBG < 32
Target access required	Yes
Mnemonic	DBG_WRITE_CAN1

Table 46 DBG_WRITE_CAN1 command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0E
3	BYTE	Target Resource Identifier (TRI)
4	DWORD	Byte Address

The DBG_WRITE_CAN1 command is closely related to the DBG_WRITE command, see chapter 4.1.12. Since this standard and its feature set shall also be usable with CAN, the limit of MAX_CTO_DBG = 8 must be considered. Since the size of the DBG_WRITE command exceeds the MAX_CTO_DBG limit for CAN, the DBG_WRITE command is split up in a sequence of commands: DBG_WRITE_CAN1, DBG_WRITE_CAN2 and DBG_WRITE_CAN_NEXT.

A debug XCP master shall send the commands DBG_WRITE_CAN1, DBG_WRITE_CAN2 and DBG_WRITE_CAN_NEXT directly in succession. The XCP slave shall respond with a negative response, if the debug XCP master sends any other command interleaved with the before mentioned commands.

The behavior of the DBG_WRITE_CAN* sequence towards the target shall be the same as for the DBG_WRITE command. Thus, the description of the DBG_WRITE command largely applies to the DBG_WRITE_CAN* sequence as well. Differences are described in chapter 4.1.16.

<u>Negative Response:</u>

See definition of negative response in chapter 4.1.13.

In case of a negative response, the debug XCP master shall not send any further DBG_WRITE_CAN* command of the sequence.

4.1.16 Download from Master to Slave (CAN – Part 2)

Category	Mandatory for MAX_CTO_DBG < 32
Target access required	Yes
Mnemonic	DBG_WRITE_CAN2

Table 47 DBG_WRITE_CAN2 command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x0F
3	BYTE	Element Width (EW) – see Table 48
4	BYTE	Number of data elements (N) Standard Mode: [1..(MAX_CTO_DBG-4)/EW] Block Mode: [1..MAX_BS*(MAX_CTO_DBG-4)/EW]

Table 48 Element Width parameter coding for DBG_WRITE_CAN2

ID	Description
0x01	BYTE
0x02	WORD
0x04	DWORD
Others	Reserved

For general behavior see the description of the DBG_WRITE command.

Due to the limited command size and the resulting command splitting, the semantic of the element width and number of data elements differ between the commands DBG_WRITE and DBG_WRITE_CAN2.

If the XCP slave does not support block transfer mode, the debug XCP master transfers all elements in a single command packet. Therefore, the number of data elements in the request has to be in the range [1..(MAX_CTO_DBG-4)/EW]. If the number of data elements does not match the expected value, the XCP slave shall respond with the error ERR_SEQUENCE. The negative response will contain the expected number of data elements, see Table 49

Table 49 DBG_WRITE_CAN2 negative response structure

Position	Type	Description
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_SEQUENCE
2	BYTE	Number of expected data elements

After receiving a DBG_WRITE_CAN2 command the XCP slave first has to check whether there are enough resources available in order to cover the complete write request. If the XCP slave does not have enough resources, it shall send ERR_MEMORY_OVERFLOW and does not execute any single write request. If the XCP slave rejects a `DBG_WRITE_CAN*` command, it shall not change the target memory.

If block transfer mode is supported, the debug XCP master transfers the elements in multiple command packets. For the XCP slave however, there might be limitations concerning the maximum number of consecutive command packets (block size `MAX_BS`). Therefore the number of data elements can be in the range [`1..min(255, MAX_BS*(MAX_CTO_DBG-4)/EW)`].

Negative Response:

See definition of negative response in chapter 4.1.13.

If the `DBG_WRITE_CAN2` command is not preceded by command `DBG_WRITE_CAN1`, the XCP slave shall respond with the error `ERR_SEQUENCE`.

In case of a negative response, the debug XCP master shall not send any further `DBG_WRITE_CAN*` commands of the sequence.

4.1.17 Download from Master to Slave (CAN – Part 3)

Category	Mandatory for `MAX_CTO_DBG` < 32
Target access required	yes
Mnemonic	DBG_WRITE_CAN_NEXT

Table 50 DBG_WRITE_CAN_NEXT command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x10
3	BYTE	Number of remaining data elements (N)
4.. 3+N*EW	ELEMENTs	Data elements to write

This command carries the elements that shall be written to the target.

In standard communication mode, the debug XCP master shall send only one `DBG_WRITE_CAN_NEXT`. In case of block transfer mode, the debug XCP master may send multiple `DBG_WRITE_CAN_NEXT` commands. In block transfer mode, the XCP slave acknowledges `DBG_WRITE_CAN1`, `DBG_WRITE_CAN2` and, without any error, the last `DBG_WRITE_CAN_NEXT` commands. The separation time between the command packets (`MIN_ST`) and the maximum number of packets (`MAX_BS`) are specified in the response of the `GET_COMM_MODE_INFO` command (see [1]).

If a single `DBG_WRITE_CAN_NEXT` command does not allow transmitting all elements, the debug XCP master shall send all but the last `DBG_WRITE_CAN_NEXT` commands with (`MAX_CTO_DBG-4/EW)` data elements. The debug XCP master shall send the remaining elements in the last `DBG_WRITE_CAN_NEXT` command.

If the XCP slave detects an internal problem during a block mode transfer, it can send a negative response at once. If block transfer mode is requested and not enough resources are available, the XCP slave can send the negative response code already after the initial `DBG_WRITE_CAN2` command of the debug XCP master.

Negative Response:

See definition of negative response in chapter 4.1.13.

If the first `DBG_WRITE_CAN_NEXT` command is not preceded by command `DBG_WRITE_CAN2`, the XCP slave shall return the error `ERR_SEQUENCE`.

If the slave does not support Master Block Mode, see [1], but the debug XCP master tries to do so, the XCP slave shall return the error `ERR_CMD_UNKNOWN` after the second consecutive `DBG_WRITE_CAN_NEXT` command.

In case of a negative response, the debug XCP master shall not send any further `DBG_WRITE_CAN*` commands of the sequence.

4.1.18 Upload from Slave to Master

Category	Mandatory for MAX_CTO_DBG >= 32
Target access required	yes
Mnemonic	DBG_READ

Table 51 DBG_READ command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x11
3	BYTE	Reserved
4	BYTE	Target Resource Identifier (TRI)
5	BYTE	Element width (EW), see Table 41
6	WORD	Number of data elements (N)
8	DLONG	Byte address

DBG_READ reads a number of data elements from the target. This command always accesses physical memory.

The DBG_READ command is similar to the UPLOAD and SHORT_UPLOAD commands defined in the XCP base standard. While the XCP slave might perform an address translation for the UPLOAD and SHORT_UPLOAD commands the address given by the DBG_READ command is a physical target address. I.e. the XCP slave shall not modify the address given by this command in any way.

The Target Resource Identifier specifies the interpretation of the Byte Address. This parameter is target specific. Details are given in Appendix: C.

Positive Response:

Table 52 DBG_READ positive response structure

Position	Type	Description
0	BYTE	Packet ID: 0xFF
..	BYTEs	Reserved: used for alignment; only if EW > 0x01
EW	ELEMENT 1	1st data element
..	..	..
N*EW	ELEMENT N	nth data element

If the XCP slave does not support block transfer mode, the XCP slave transfers all read elements in a single response packet. Therefore the number of data elements in the request has to be in the range $[1..\left\lfloor\frac{MAX\_CTO\_DBG-\mathrm{EW}}{\mathrm{EW}}\right\rfloor]$. The XCP slave shall respond with the error ERR_OUT_OF_RANGE if the number of data elements is more than $\left\lfloor\frac{MAX\_CTO\_DBG-\mathrm{EW}}{\mathrm{EW}}\right\rfloor$.

If the XCP slave supports the slave block transfer mode it transfers read elements in multiple responses to the same request packet. For the debug XCP master there are no limitations allowed concerning the maximum block size. Therefore the number of data elements can be in the range [1..65535]. The XCP slave will transmit $\left\lceil \frac{N}{\left\lfloor \frac{MAX\_CTO\_DBG-\mathrm{EW}}{\mathrm{EW}} \right\rfloor} \right\rceil$ response packets. The separation time between the response packets is depending on the XCP slave implementation. It’s the responsibility of the debug XCP master to keep track of all packets and to check for lost packets.

Furthermore, the following requirements shall be fulfilled:

- the address shall be aligned according to the element width
- the XCP slave shall perform a memory access via the debug interface

Negative Response:

In case of address misalignment, the XCP slave shall respond with the error `ERR_OUT_OF_RANGE`.

If the number of data elements does not match the expected value, the XCP slave shall respond with the error `ERR_OUT_OF_RANGE`.

In case that the element width is not supported the XCP slave shall respond with the error `ERR_DBG_EW_UNSUPPORTED`.

In case that the TRI is not supported, the XCP slave shall respond with the error `ERR_DBG_TRI_UNSUPPORTED`.

In case of a bus error, the XCP slave shall respond with the error `ERR_DBG_BUS_ERROR`.

4.1.19 Upload from Slave to Master (CAN – Part 1)

Category	Mandatory for `MAX_CTO_DBG` < 32
Target access required	yes
Mnemonic	DBG_READ_CAN1

Table 53 DBG_READ_CAN1 command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x12
3	BYTE	Target Resource Identifier (TRI)
4	DWORD	Byte address

The `DBG_READ_CAN1` command is closely related to the `DBG_READ` command, see chapter 4.1.18. Since this standard and its feature set shall also be usable with CAN, this standard considers the limit of `MAX_CTO_DBG` = 8. Since the size of the `DBG_READ` command exceeds the `MAX_CTO_DBG` limit for CAN, the debug XCP master splits the `DBG_READ` command in a sequence of commands `DBG_READ_CAN1` and `DBG_READ_CAN2`.

A debug XCP master shall send the commands `DBG_READ_CAN1` and `DBG_READ_CAN2` directly in succession. If the debug XCP master sends any other command interleaved with the before mentioned commands, the XCP slave shall respond with a negative response to `DBG_READ_CAN2`.

The behavior of the `DBG_READ_CAN*` sequence towards the target shall be the same as for the `DBG_READ` command. Thus, the description of the `DBG_READ` command largely applies to the `DBG_READ_CAN*` sequence as well. Differences are described in the chapter 4.1.20.

Negative Response:

See definition of negative response in chapter 4.1.18

In case of a negative response, the debug XCP master shall not send any further `DBG_READ_CAN*` command of the sequence.

4.1.20 Upload from Slave to Master (CAN – Part 2)

Category	Mandatory for MAX_CTO_DBG < 32
Target access required	yes
Mnemonic	DBG_READ_CAN2

Table 54 DBG_READ_CAN2 command structure

Position	Type	Description
2	BYTE	SW-DBG command code = 0x13
3	BYTE	Element width (EW), see Table 48
4	BYTE	Number of data elements (N)

The description of the DBG_READ command largely applies to the DBG_READ_CAN* sequence. There are only two differences. Firstly, the element width shall not be of type DLONG. Secondly, the number of data elements that can be read by one DBG_READ_CAN* sequence in case of block transfer mode is limited to the range [1…255].

Positive Response:

See Table 55.

Negative Response:

See definition of negative response in chapter 4.1.18

If the DBG_READ_CAN2 command is not preceded by command DBG_READ_CAN1, the XCP slave shall respond with error ERR_SEQUENCE.

5 Communication Error Handling

This chapter describes the error handling for SW-DBG specific XCP commands. It extends the error handling concepts specified in the chapter “Communication Error Handling” of the XCP base standard [1]. Please refer to the XCP base standard for obtaining fundamentals on XCP error handling.

5.1 SW-DBG specific Error Codes

This standard defines a set of SW-DBG specific error codes given in Table 56. To inform the debug XCP master about a SW-DBG specific error, a negative response shall contain two error codes directly in succession. An example is given in Table 55.

Table 55 Generic structure of a negative response containing a SW-DBG specific error code

Position	Type	Description
0	BYTE	Packet ID = 0xFE
1	BYTE	ERR_DBG
2	BYTE	SW-DBG specific error code
..		Optionally further error specific data

Position 1 of the negative response contains the error code ERR_DBG, as defined in [1]. Upon this code the master can derive that a second, SW-DBG specific error code will directly follow. Apart from this, the error code has no further significance.

At position 2, the SW-DBG specific error code is given which optionally might be followed by further data providing more information about the error. The following table shows the SW-DBG specific error codes:

Table 56 SW-DBG specific error codes

Error	Code	Description
ERR_DBG_BUS_ERROR	0x00	Occurrence of a bus error on the target interface
ERR_DBG_HWIO_CONTROL	0x01	DBG_HWIO_CONTROL specific error
ERR_DBG_HALT_AFTER_RESET	0x02	Target could not be halted after release of reset
ERR_DBG_JPL	0x03	Error in JPL sequence occurred
ERR_DBG_LLT	0x04	Error in LLT sequence occurred
ERR_DBG_EW_UNSUPPORTED	0x05	Unsupported element width
ERR_DBG_TRI_UNSUPPORTED	0x06	Unsupported Target Resource Identifier
ERR_DBG_ATTACH_MISSING	0x07	The XCP slave received a debug command prior to a `DBG_ATTACH` command

5.2 Error Code Handling

Table 57 SW-DBG specific error handling

Command	Error	Pre-Action	Action
DBG_ATTACH	timeout t_1	SYNCH	repeat 2 times
	ERR_ACCESS_LOCKED	unlock slave	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
DBG_GET_VENDOR_INFO	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_GET_MODE_INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_GET_JTAG_ID	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_GENERIC	display error	tool specific error handling
	ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE	-	repeat
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_HALT_AFTER_RESET	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times

Command	Error	Pre-Action	Action
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	-	repeat
	ERR_DBG_HALT_AFTER_ RESET	-	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_GET_HWIO_ INFO	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_SET_HWIO_ EVENT	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_GENERIC	-	retry other parameter
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_HWIO_ CONTROL	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_UNKNOWN	-	command may no longer be used

Command	Error	Pre-Action	Action
	ERR_DBG_HWIO_CONTROL	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_EXCLUSIVE_ TARGET_ACCESS	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	-	repeat
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_SEQUENCE_ MULTIPLE	timeout t1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	-	repeat
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_DBG_JPL	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_LLT	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times

Command	Error	Pre-Action	Action
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	read vref pin, read reset pin	repeat when target is accessible
	ERR_DBG_LLT	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_READ_ MODIFY_WRITE	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	read vref pin, read reset pin	repeat when target is accessible
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_DBG_BUS_ERROR	display error	tool specific error handling
	ERR_DBG_TRI_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_EW_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_WRITE	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	read vref pin, read reset pin	repeat when target is accessible

Command	Error	Pre-Action	Action
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_MEMORY_OVERFLOW	-	display error
	ERR_DBG_BUS_ERROR	display error	tool specific error handling
	ERR_DBG_TRI_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_EW_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_WRITE_NEXT	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE	read vref pin, read reset pin	restart with DBG_WRITE when target is accessible
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_CMD_SYNTAX	-	retry other syntax
	ERR_SEQUENCE	restart with DBG_WRITE	repeat 2 times
	ERR_DBG_BUS_ERROR	display error	tool specific error

Command	Error	Pre-Action	Action
			handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_WRITE_CAN1	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_DBG_TRI_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_WRITE_CAN2	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_MEMORY_OVERFLOW	-	display error
	ERR_SEQUENCE	restart with DBG_WRITE_CAN1	repeat 2 times
	ERR_DBG_EW_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_WRITE_CAN_NEXT	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE	read vref pin, read reset pin	restart with DBG_

Command	Error	Pre-Action	Action
			WRITE_ CAN1 when target is accessible
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_SEQUENCE	restart with DBG_WRITE_ CAN1	repeat 2 times
	ERR_DBG_BUS_ERROR	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_READ	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_ NOT_ACCESSIBLE	read vref pin, read reset pin	repeat when target is accessible
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_DBG_BUS_ERROR	display error	tool specific error handling
	ERR_DBG_TRI_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_EW_UNSUPPORTED	display error	tool specific error

Command	Error	Pre-Action	Action
			handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_READ_CAN1	timeout t_1	SYNCH	repeat 2 times
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_DBG_TRI_UNSUPPORTED	display error	tool specific error handling
	ERR_DBG_ATTACH_MISSING	execute DBG_ATTACH command	repeat 2 times
DBG_READ_CAN2	timeout t_1	SYNCH	repeat 2 times
	ERR_RESOURCE_TEMPORARY_NOT_ACCESSIBLE	read vref pin, read reset pin	restart with DBG_READ_CAN1 when target is accessible
	ERR_CMD_UNKNOWN	-	command may no longer be used
	ERR_CMD_BUSY	wait t_7	repeat ∞ times
	ERR_OUT_OF_RANGE	-	retry other parameter
	ERR_SEQUENCE	restart with DBG_READ_CAN1	repeat 2 times
	ERR_DBG_BUS_ERROR	display error	tool specific error handling

Command	Error	Pre-Action	Action
	`ERR_DBG_EW_UNSUPPORTED`	display error	tool specific error handling
	`ERR_DBG_ATTACH_MISSING`	execute `DBG_ATTACH` command	repeat 2 times

6 SW-DBG specific XCP Events

Events are used when the slave autonomously has to send data to the tool. All SW-DBG related XCP events are grouped within a specific event space. The following table shows the definition of the SW-DBG specific event space:

Table 58 Definition of the SW-DBG specific XCP event space

Position	Type	Description
0	BYTE	XCP event = 0xFD
1	BYTE	SW-DBG event = 0xFC
2	BYTE	SW-DBG event subcode
3.. MAX_CTO_DBG-1	BYTE	Subcode specific data

An overview of all SW-DBG related events is given in the table below:

Table 59 SW-DBG events overview

SW-DBG Event	SW-DBG Subcode
EV_DBG_SERVICE_LEVEL_CHANGE	0x00
EV_DBG_HWIO_CHANGE	0x01

6.1 Change of Service Level

Mnemonic	EV_DBG_SERVICE_LEVEL_CHANGE

Table 60 EV_DBG_SERVICE_LEVEL_CHANGE event packet

Position	Type	Description
2	BYTE	SW-DBG event subcode = 0x00
3	BYTE	Current service level

With `EV_DBG_SERVICE_LEVEL_CHANGE` the slave indicates a service level change to the debug XCP master.

6.2 Change of HW-IO

Mnemonic EV_DBG_HWIO_CHANGE

Table 61 EV_DBG_HWIO_CHANGE event packet

Position	Type	Description
2	BYTE	SW-DBG event subcode = 0x01
3	BYTE	Index of the HW-IO pin, see chapter 4.1.6
4	BYTE	Trigger condition 0 = pin changed to low 1 = pin changed to high

With `EV_DBG_HWIO_CHANGE` the slave indicates to the debug XCP master that the trigger condition of the specified HW-IO pin occurred.

7 XCP Sequence Examples

The sequences below are supplied to aid the understanding of the relationship between individual commands.

Table 62 Notation for indicating the packet direction

Symbol	Direction	Packet direction
→	CMD	Master to slave
←	RES	Slave to master

7.1 Setting up a session

Table 63 Getting basic information

Direction	XCP Packet	Parameters
	CONNECT	
→	FF 00	mode = 0x00
		=> NORMAL
←	FF 35 C0 08 08 00 10 10	RESOURCE = 0x35
		=> CAL/PAG, DAQ, PGM, DBG available
		COMM_MODE_BASIC = 0xC0
		=> Byte Order = Intel
		ADDRESS_GRANULARITY = Byte
		Slave Block Mode available
		GET_COMM_MOD_INFO provides additional information
		MAX_CTO = 0x08
		MAX_DTO = 0x0008
		XCP Protocol Layer Version = 0x10
		XCP Transport Layer Version = 0x10
	GET_COMM_MODE_INFO	
→	FB	
←	FF 00 01 00 02 00 00 64	COMM_MODE_OPTIONAL = 0x01
		=> Master Block Mode available
		MAX_BS = 0x02
		MIN_ST = 0x00
		QUEUE_SIZE = 0x00
		XCP Driver Version = 0x64
	GET_STATUS	
→	FD	
←	FF 00 35 01 00 00	Current Session Status = 0x00
		=> no request active,
		Resume not active,
		no DAQ running
		Resource Protection Status = 0x35
		=> CAL/PAG, DAQ, PGM, DBG are protected
		STATE_NUMBER = 1
		Session Configuration ID= 0x0000
		=> no RESUME session configured

7.2 Debug Attach

Table 64 Attaching to XCP slave with SW-DBG support (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
		DBG_ATTACH
→	C0 FC 00	
←	FF 01 00 FF FF 05 B0	Major version = 0x01
		Minor version = 0x00
		Timeout in n*2 ms = 0xFF
		=> Timeout t_1 = 510 ms
		Timeout in n*2 ms = 0xFF
		=> Timeout t_7 = 510 ms
		`MAX_CTO_DBG =` 0x5B0 = 1456

7.3 Get Debug Mode

Table 65 Getting information about debugging properties of the XCP slave

Direction	XCP Packet	Parameters
		DBG_GET_MODE_INFO
→	C0 FC 02	
←	FF 00 02 01 00 01	Reserved = 0x00
		`MAX_HW_IO_PINS` = 0x02
		=> The XCP slave supports two HW-IO pins
		Dialect = 0x01
		=> JTAG is used for low level target access
		Feature = 0x00
		=> Halt after reset command is not supported
		Service level = 0x01
		=> Exclusive access to target interface for SW-DBG

7.4 Get JTAG ID

Table 66 Getting the JTAG ID of the target (BYTE_ORDER = Motorola)

Direction	**XCP Packet**	**Parameters**
	DBG_GET_JTAG_ID	
→	C0 FC 03	
←	FF 00 00 00 00 11 20 41	Reserved = 0x00
		Reserved = 0x00
		Reserved = 0x00
		Target JTAG ID = 0x00112041

7.5 HW-IO Pin Handling

Table 67 Getting information about available HW-IO pins

Direction	**XCP Packet**	**Parameters**
	DBG_GET_HWIO_INFO	
→	C0 FC 05 00	Index = 0x00
		=> The response begins with index 0
←	FF 03 00 03 01 01 01 03 02 01 02 02 04 03	Number of HW-IO pins in this response = 0x03
		Index = 0x00
		Mode = 0x03
		=> XCP slave in/output, event disabled
		Class = 0x01
		=> PORST
		Active state = 0x01
		=> low active
		Index = 0x01
		Mode = 0x03
		=> XCP slave in/output, event disabled
		Class = 0x02
		=> Secondary reset
		Active state = 0x01
		=> low active
		Index = 0x02
		Mode = 0x02
		=> XCP slave output, event disabled
		Class = 0x04
		=> Watchdog disable
		Active state = 0x03
		=> a clock frequency is generated in active state

Table 68 Asserting PORST and disabling watchdog (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
DBG_HWIO_CONTROL		
→	C0 FC 07 02 00 00 00 00 02 04 00 01	Number of pins in this command = 0x02
		Pin index = 0x00
		Desired state of pin = 0x00
		=> Signal attached to pin is set to low level
		Clock frequency = 0x0000
		Pin index = 0x02
		Desired state of pin = 0x04
		Clock frequency = 0x0001
		=> XCP slave generates signal with clock frequency 1 Hz
←	FF 00 04	Current state of pin at index 0 = 0x00
		=> State is low level
		Current state of pin at index 2 = 0x04
		=> XCP slave generates signal with clock frequency 1 Hz

7.6 JPL Sequence

Table 69 Requesting the JTAG bus and going through RST to IDLE state (BYTE_ORDER = Intel)

Direction	XCP Packet	Parameters
DBG_SEQUENCE_MULTIPLE		
→	C0 FC 09 03 01 00	Request JTAG Bus with TMS=TDI=0 at begin Release JTAG Bus at end Number of sequences = 0x0001
	06 00	Number of bytes of all JPL commands of sequence 0 = 0x0006
	03 00 01 08 7F	JPL command0 = 0x03 0x00 0x01 0x08 0x7F => Go through RST to IDLE state
←	FF 00 01 00 00 00 00 00 00 00	Positive response Number of results = 0x0001 Response 0 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x00000000

A more complex example is shown in Table 70. It first moves the JTAG via the PAUSE-DR state to a defined state. After that a new JTAG Tap is selected. That Tap is used to read out some 192 bit long register via consecutive IR and DR scans.

Table 70 Write a huge JTAG register with preceding IR Tap setup (BYTE_ORDER = Intel)

Direction	XCP Packet	Parameters
	DBG_SEQUENCE_MULTIPLE	
➔	C0 FC 09 03 06 00	Request JTAG Bus with TMS=TDI=0 at begin Release JTAG Bus at end Number of sequences = 0x0006
	6D 00	Number of bytes of all JPL commands of sequence 0 = 0x006D (109)
	03 00 01 04 05 03 00 01 03 03 03 00 01 04 03 04 01 06 20 3E 00 00 00 00 01 00 00 03 00 01 02 01 03 00 01 04 03 04 01 06 20 2A 00 00 00 00 01 00 00 03 00 01 02 01 03 00 01 04 03 04 02 0A 02 00 02 10 00 00 00 00 00 00 01 00 00 03 00 01 02 01 03 00 01 03 01 04 04 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00	JPL command0 = 0x03 0x00 0x01 0x04 0x05 => Go from RTI to PAUSEDR state JPL command1 = 0x03 0x00 0x01 0x03 0x03 => Go from PAUSEDR to RTI state JPL command2 = 0x03 0x00 0x01 0x04 0x03 => Go from RTI to SHIFTIR state JPL command3 = 0x04 0x01 0x06 0x20 0x3E 0x00 0x00 0x00 0x00 0x01 0x00 0x00 => Shift 0x3E, Go to EXIT1IR state JPL command4 = 0x03 0x00 0x01 0x02 0x01 => Go from EXIT1IR to RTI state JPL command5 = 0x03 0x00 0x01 0x04 0x03 => Go from RTI to SHIFTIR state JPL command6 = 0x04 0x01 0x06 0x20 0x2A 0x00 0x00 0x00 0x00 0x01 0x00 0x00 => Shift 0x2A, Go to EXIT1IR state JPL command7 = 0x03 0x00 0x01 0x02 0x01 => Go from EXIT1IR to RTI state JPL command8 = 0x03 0x00 0x01 0x04 0x03 => Go from RTI to SHIFTIR state JPL command9 = 0x04 0x02 0x0A 0x02 0x00 0x02 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 => Shift 0x210, Go to EXIT1IR state JPL command10 = 0x03 0x00 0x01 0x02 0x01 => Go from EXIT1IR to RTI state JPL command11 = 0x03 0x00 0x01 0x03 0x01 => Go from RTI to SHIFTDR state JPL command12 = 0x04 0x04 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 => Shift 0x00000000
	00	Padding byte
	18 00	Number of bytes of all JPL commands of sequence 1 = 0x0018 (24)

Direction	XCP Packet	Parameters

	04 04 20 00 00 00 00 00
	00 00 00 00 00 00 00 00
	00 00 00 00 00 01 00 00
		JPL command0 = 0x04 0x04 0x20 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 0x00000000

	18 00	Number of bytes of all JPL commands of sequence 2 = 0x0018 (24)

	04 04 20 00 00 00 00 00
	00 00 00 00 00 00 00 00
	00 00 00 00 00 01 00 00
		JPL command0 = 0x04 0x04 0x20 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 0x00000000

	18 00	Number of bytes of all JPL commands of sequence 3 = 0x0018 (24)

	04 04 20 00 00 00 00 00
	00 00 00 00 00 00 00 00
	00 00 00 00 00 01 00 00
		JPL command0 = 0x04 0x04 0x20 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 0x00000000

	18 00	Number of bytes of all JPL commands of sequence 4 = 0x0018 (24)

	04 04 20 00 00 00 00 00
	00 00 00 00 00 00 00 00
	00 00 00 00 00 01 00 00
		JPL command0 = 0x04 0x04 0x20 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 0x00000000

	1D 00	Number of bytes of all JPL commands of sequence 5 = 0x001D (29)

	04 04 20 80 00 00 00 00
	00 00 00 00 00 00 00 00
	00 00 00 00 00 01 00 00
	
	03 00 01 02 01
		JPL command0 = 0x04 0x04 0x20 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
		0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 0x00000000, Go to EXIT1DR state
		
		JPL command1 = 0x03 0x00 0x01 0x02 0x01
		=> Go from EXIT1DR to RTI state

	00	Padding byte
←	FF 00 06 00 00 00 09 0B C6 DC 00 00 00 00 00 00 00 00 00 00 92 00 00 00 09 0B C7 08	Positive response Number of results = 0x0006 Response 0 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x090BC6DC Response 1 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x00000000 Response 2 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x00009200 Response 3 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x090BC708

Direction	XCP Packet	Parameters

	00 00
	44 00 44 00
	
	00 00
	00 00 00 02
		Response 4
		=> Execution status = 0x00 (successful)
		=> Repeat count 0
		=> TDO 0x44004400
		
		Response 5
		=> Execution status = 0x00 (successful)
		=> Repeat count 0
		=> TDO 0x00000002

An example for a negative response is given in the next table. It shows an execution error of the second sequence because of a low level target interface error.

Table 71 DBG_SEQUENCE_MULTIPLE negative response structure (BYTE_ORDER = Intel)

Direction	XCP Packet	Parameters
DBG_SEQUENCE_MULTIPLE		
→	C0 FC 09 03 02 00	Request JTAG Bus with TMS=TDI=0 at begin Release JTAG Bus at end Number of sequences = 0x02
	16 00	Number of bytes of all JPL commands of sequence 0 = 0x0016 (22)

	03 00 01 04 03
	
	04 01 08 80 55 00 00 00
	00 01 00 00
	
	03 00 01 02 01
		JPL command0 = 0x03 0x00 0x01 0x04 0x03
		=> Go from RTI to SHIFTIR state
		
		JPL command1 = 0x04 0x01 0x08 0x80 0x55 0x00 0x00 0x00 0x00 0x01 0x00 0x00
		=> Shift 8Bit 0x55, Go to EXIT1IR state
		
		JPL command2 = 0x03 0x00 0x01 0x02 0x01
		=> Go from EXIT1IR to RTI state

	22 00	Number of bytes of all JPL commands of sequence 1 = 0x0022 (34)
	03 00 01 03 01 04 04 20 00 00 00 00 CA FE BE EF 00 00 00 00 00 00 00 00 00 00 01 00 00 03 00 01 02 01	JPL command0 = 0x03 0x00 0x01 0x04 0x03 => Go from RTI to SHIFTDR state JPL command1 = 0x04 0x04 0x20 0x00 0x00 0x00 0x00 0xCA 0xFE 0xBE 0xEF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x00 0x00 => Shift 32Bit 0xCAFEBEEF, Go to EXIT1DR state JPL command2 = 0x03 0x00 0x01 0x02 0x01 => Go from EXIT1DR to RTI state
←	FE FC 03 00 02 00 00 00 00 00 00 00 04 00	Negative response => ERR_DBG_JPL Number of results = 0x0002 Response 0 => Execution status = 0x00 (successful) => Repeat count 0 => TDO 0x00000000 Response 1

Direction	XCP Packet	Parameters
	00 00 00 00	=> Execution status = 0x04 (Low level target interface error) => Repeat count 0 => TDO 0x00000000

7.7 DAP over LLT

The first example shows successful transmission of a DAP telegram with a DATA and response length of 0:

Table 72 LLT Example Sequence 1

Nr.	CMD	LEN	DATA	RLEN	RDATA
1	0x0B	0	N/A	0	N/A

Table 73 XCP Packets for LLT Example Sequence 1

Direction	XCP Packet	Parameters
	DBG_LLT	
→	C0 FC 0A 01 00	1 telegram, mode = 0x00 => Target may reply on both DAP lines
	00 0B 00	Expected response length: 0 bits CMD: 0x0B LEN: 0 bits
←	FF 01	Positive response
	00	Expected response length: 0 bits

The second example shows successful transmission of a sequence of two telegrams. The first writes 3 bits to the target and does not expect response data. The second writes 7 bits and gets 16 bits back.

Table 74 LLT Example Sequence 2

Nr.	CMD	LEN	DATA	RLEN	RDATA
1	0x1C	3	0x1	0	N/A
2	0x1A	7	0x4B	16	0x0020

Table 75 XCP Packets for LLT Example Sequence 2

Direction	XCP Packet	Parameters
DBG_LLT		
→	C0 FC 0A 02 00	2 telegrams, mode = 0x00 => Target may reply on both DAP lines
	00 1C 03 01	Telegram 1: Expected response length: 0 bits CMD: 0x1C LEN: 3 bits DATA: 0x01
	10 1A 07 4B	Telegram 2: Expected response length: 16 bits CMD: 0x1A LEN: 7 bits DATA: 0x4B
←	FF 02	Positive response, 2 telegrams
	00	Telegram 1 Expected response length: 0 bits
	10 20 00	Telegram 2 Expected response length: 16 bits Response: 0x0020

The third example shows unsuccessful transmission of a sequence of two telegrams. The first writes 3 bits to the target and does not expect response data. The second writes 36 bits and results in a timeout error.

Table 76 LLT Example Sequence 3

Nr.	CMD	LEN	DATA	RLEN	RDATA
1	0x1C	3	0x1	0	N/A
2	0x08	36	0x123456781	0	timeout

Table 77 XCP Packets for LLT Example Sequence 3

Direction	XCP Packet	Parameters
DBG_LLT		
→	C0 FC 0A 02 00	2 telegrams, mode = 0x00 => Target may reply on both DAP lines
	00 1C 03 01	Telegram 1: Expected response length: 0 bits CMD: 0x1C LEN: 3 bits DATA: 0x01
	00 08 24 81 67 45 32 01	Telegram 2: Expected response length: 0 bits CMD: 0x08 LEN: 36 bits DATA: 0x123456781

Direction	XCP Packet	Parameters
←	FE FC 04	Negative response (ERR_DBG_LLT)
	02 02	Telegram 2 failed with Timeout error

The fourth example shows successful transmission of a DAP telegram that returns data on DAP1 only:

Table 78 LLT Example Sequence 4

Nr.	CMD	LEN	DATA	RLEN	RDATA
1	0x02	8	0x00	8	0x00

Table 79 XCP Packets for Example Telegram Sequence 4

Direction	XCP Packet	Parameters
	DBG_LLT	
→	C0 FC 0A 01 01	1 telegram, mode = 0x01 => Target replies on DAP1 only
	08 02 08	Expected response length: 8 bits CMD: 0x02 LEN: 8 bits
←	FF 01	Positive response, 1 telegram
	08 00	Expected response length: 8 bits Response: 00

7.8 Memory read and write

Simple memory read example for `MAX_CTO_DBG` >= 32:

Table 80 Reading four bytes at address 0x70000000 (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
	DBG_READ	
→	C0 FC 11 00 01 04 00 01 00 00 00 00 70 00 00 00	Reserved = 0x00
		Target Resource Identifier (TRI) = 0x01
		Element width = 0x04
		=> DWORD
		Number of data elements = 0x0001 (in granularity given by element width)
		Byte address = 0x70000000
←	FF 00 00 00 01 02 03 04	3 reserved bytes for alignment
		Read data element = 0x01020304

Simple memory read example for `MAX_CTO_DBG` < 32:

Table 81 Reading four bytes at address 0x70000000 (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
		DBG_READ_CAN1
→	C0 FC 12 01 70 00 00 00	Target Resource Identifier (TRI) = 0x01
		Byte address = 0x70000000
←	FF	Positive response
		DBG_READ_CAN2
→	C0 FC 13 01 04	Element width = 0x01
		=> BYTE
		Number of data elements = 0x04 (in granularity given by element width)
←	FF 01 02 03 04	Read data elements = 0x01 0x02 0x03 0x04

Simple memory write example for `MAX_CTO_DBG` >= 32:

Table 82 Writing four bytes to address 0x70000000 (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
		DBG_WRITE
→	C0 FC 0C 00 01 04 00 01 00 00 00 00 70 00 00 00 01 02 03 04	Reserved = 0x00
		Target Resource Identifier (TRI) = 0x01
		Element width = 0x04
		=> DWORD
		Number of data elements = 0x0001 (in granularity given by element width)
		Byte address = 0x70000000
		Data element to write = 0x01020304
←	FF	Positive response

Simple memory write example for `MAX_CTO_DBG` < 32:

Table 83 Writing four bytes to address 0x70000000 (BYTE_ORDER = Motorola)

Direction	XCP Packet	Parameters
DBG_WRITE_CAN1		
→	C0 FC 0E 01 70 00 00 00	Target Resource Identifier (TRI) = 0x01
		Byte address = 0x70000000
←	FF	Positive response
DBG_WRITE_CAN2		
→	C0 FC 0F 01 04	Element width = 0x01
		=> BYTE
		Number of data elements = 0x04 (in granularity given by element width)
←	FF	Positive response
DBG_WRITE_CAN_NEXT		
→	C0 FC 10 04 01 02 03 04	Number of remaining data elements = 0x04 (in granularity given by element width)
		Data elements to write = 0x01 0x02 0x03 0x04
←	FF	Positive response

8 Terms and Definitions

For the purposes of this ASAM standard the following terms and definitions apply (in alphabetical order).

Brain-dead ECU	ECU that cannot boot from flash memory
Debug XCP master	A debugger tool acting as XCP master
JPL command	One of the four different commands defined within the JTAG Programming Language
JPL player	A component within the XCP slave that processes the JPL commands transmitted via the `DBG_SEQUENCE_MULTIPLE` command as input and controls the JTAG interface pins accordingly
SW-DBG	Software Debugging over XCP

9 Symbols and Abbreviated Terms

A2L	File Extension for an ASAM 2MC Language File
AML	ASAM 2 Meta Language
CAL	Calibration (see [1])
CPU	Central Processing Unit
CTO	Command Transfer Object
DAP	Device Access Port
DAQ	Data acquisition (see [1])
DBG	Debugging (see [1])
ECU	Electronic Control Unit
EW	Element width
JPL	JTAG Programming Language
LLT	Low Level Telegram
LSBit	Least significant bit
MC	Measurement, calibration and data stimulation
MSBit	Most significant bit
PGM	ECU flash programming (see [1])
POD	Plug-On Device
PORST	Power-On Reset
SFR	Special Function Register
STIM	Data stimulation (see [1])
TAP	Test Access Port
TRI	Target Resource Identifier
VREF	ECU reference voltage
XCP	Universal Measurement and Calibration Protocol

10 Bibliography

[1] ASAM e.V.: ASAM MCD-1 XCP V1.5

[2] ASAM e.V.: ASAM MCD-1 POD V1.0

[3] IEEE Std 1149.1-2013: IEEE Standard for Test Access Port and Boundary-Scan Architecture

[4] Infineon Technologies AG: AURIX OCDS User’s Manual V2.9.1

Appendix: A. The JTAG Programming Language

The JTAG Programming Language (JPL) is a language to control the JTAG interface of the microcontroller. It consists of four different JPL commands. Several JPL commands can be combined to a JPL sequence. An XCP slave can provide a JPL player, which interprets the commands and controls the JTAG pins accordingly. The debug XCP master creates the JPL sequences and sends them to the XCP slave for execution via the `DBG_SEQUENCE_MULTIPLE` command.

JPL offers the following functionalities:

- step through the JTAG state machine
- write data
- read data
- repeat commands until expected data is read

JPL commands:

- request JTAG bus (not to be used in the scope in this standard)
- release JTAG bus (not to be used in the scope in this standard)
- step
    - step through the JTAG state machine exactly in the way as described by the argument
- data
    - read, write and verify data

A JPL player shall meet the following requirements:

- return data: The JPL player returns the last 32 TDO bits and the repeat count after the last JPL data command.
- error handling: The JPL player aborts the execution of a JPL sequence and reinitializes the JTAG interface to a defined state in case of an error. The JTAG state machine is set to idle state by walking through the reset state and the JTAG bus is released.
- The JPL player shall shift out TMS and TDI with LSBit first.
- The JPL player shall shift in bit 0 to bit 31 into last TDO value. If the JPL data command processes more than 32 bits, the JPL player shall shift out LSBit and insert latest TDO bit into MSBit.
- JTAG timing must match the target timing requirements.

A.1. JPL Command definitions

The JPL defines the JTAG bus request and release commands. The debug XCP master shall not issue these commands as part of this standard.

JPL step command

Directs the JPL player to generate exactly the number of TCK clock edges specified within the command with the TMS specified. For TDI a default value is given that remains unchanged during the execution of the step command.

Parameter fields with m > 1 are interpreted in Motorola format, i.e. the LSB is coded in the last byte of the parameter and the MSB in the first byte.

Table 84 Step command structure

Position	**Type**	**Description**
0	BYTE	Command identifier = 0x03
1	BYTE	TDI value
2	BYTE	Field width m in bytes
3	BYTE	Number of steps (bits) n
4..(4+m-1)	BYTEs	n TMS bits (m BYTEs)

JPL data command

Directs the JPL player to generate exactly the number of TCK clock edges specified within the command with the TMS and TDI data specified. For TDO an expected value and a mask are given. After the number of TCK clock cycles has been generated, the JPL player must compare TDO against the expected TDO value with respect to the mask. If the comparison fails, the appended repeat sequence is executed and the command is processed again. This is repeated up to *x* (repeat count) times or until TDO matches. The repeat sequence is not executed and the command is not repeated if *x* equals zero. If the TDO mask is all zeroes (no comparison) but the repeat count is not zero, this is executed as an unconditional repeat (loop), i.e. the command is repeated *x* times regardless of TDO.

Table 85 Data command structure

Position	Type	Description
0	BYTE	Command identifier = 0x04
1	BYTE	Field width *m* in bytes
2	BYTE	Number of data bits *n*
3	BYTEs	*n* TMS bits (*m* BYTEs)
..	..	..
3 + m	BYTEs	*n* TDI bits (*m* BYTEs)
..	..	..
3 + 2*m	BYTEs	*n* expected TDO bits (*m* BYTEs)
..	..	..
3 + 3*m	BYTEs	*n* TDO mask bits (*m* BYTEs)
..	..	..
3 + 4*m	BYTE	Repeat count *x*
4 + 4*m	BYTE	TDI repeat value
5 + 4*m	BYTE	Field width *p* in bytes
6 + 4*m	BYTE	Number of steps (bits) *q* for repeat sequence
(7 + 4*m)..((7 + 4*m) + p -1)	BYTEs	*q* TMS bits for repeat sequence (*p* BYTEs)

A.2. Examples

JPL step command

Table 86 Step example

JPL command	Parameters
03 00 02 0E 18 03	Command identifier = 0x03
	TDI value = 0x00
	Field width m in bytes = 0x02
	Number of steps (bits) n = 0x0E
	n TMS bits (m bytes) = 0x1803

Generated JTAG signals (TDO is don't care):

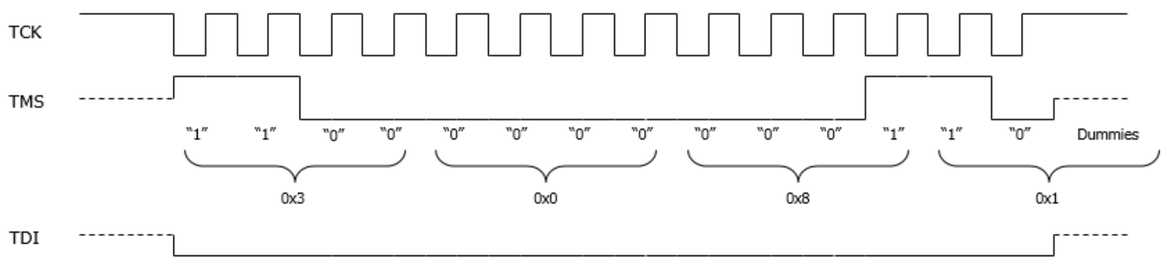

Figure 5 Step signals.

JPL data command

Table 87 Data example

JPL command	**Parameters**
04 01 08 80 48 2A 0F 0A 00 01 04 03	Command identifier = 0x04
	Field width m in bytes = 0x01
	Number of data bits n = 0x08
	n TMS bits (m bytes) = 0x80
	n TDI bits (m bytes) = 0x48
	n expected TDO bits (m bytes) = 0x2A
	n TDO mask bits (m bytes) = 0x0F
	Repeat count x = 0x0A
	TDI repeat value = 0x00
	Field width p in bytes = 0x01
	Number of steps (bits) q for repeat sequence = 0x04
	q TMS bits for repeat sequence (p bytes) = 0x03

Generated JTAG signals if expected TDO matches:

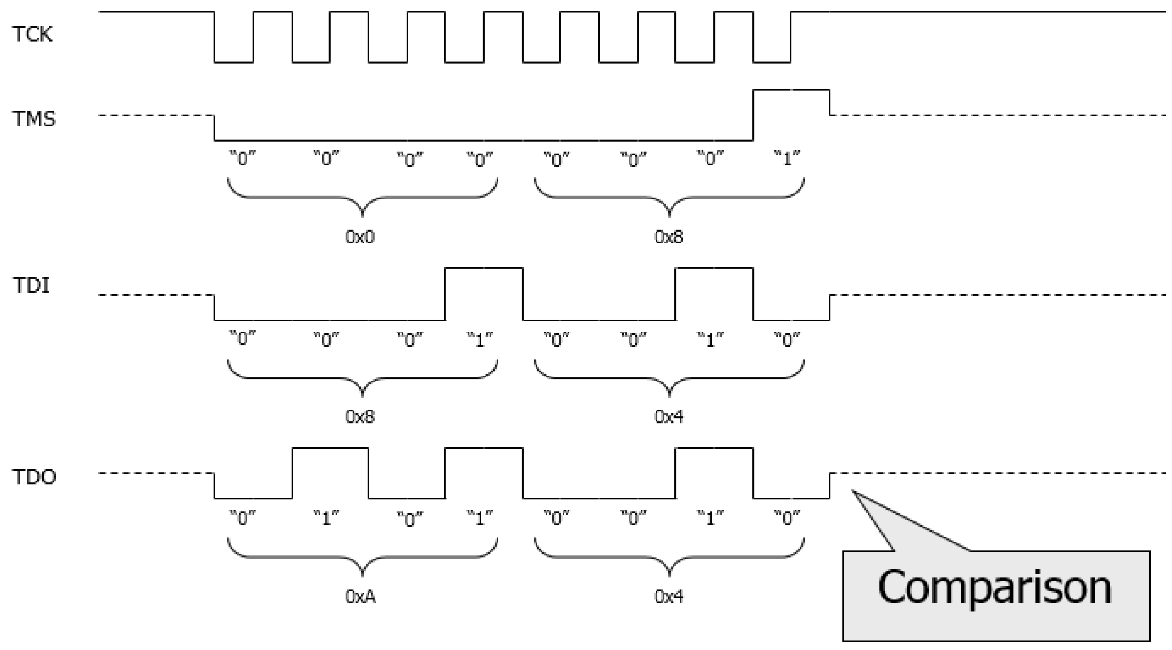

Figure 6 Data example if expected TDO matches.

Generated JTAG signals if expected TDO does not match:

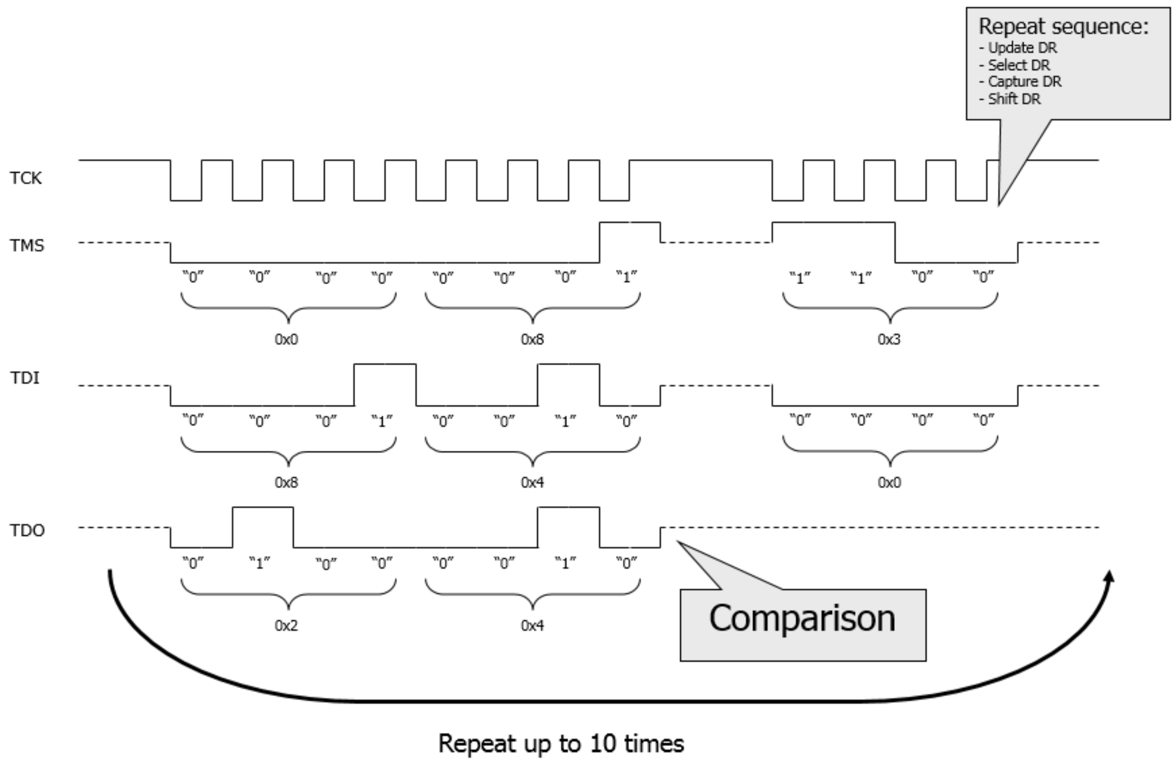

Figure 7 Data example if expected TDO does not match.

Appendix: B. Mapping between Low Level Telegram and Infineon DAP Telegram

The XCP slave may use the Device Access Port (DAP), see [4], for communication with the target. All DAP communication between the XCP slave and the target is based on telegrams whereas the telegrams differ per direction. In the following, the mapping between DAP and the Low Level Telegram is briefly explained.

<u>Direction debug XCP master to target</u>

In Figure 8 the basic structure of a DAP telegram when sent from the XCP slave to the target is given. A telegram is transmitted from right hand side to left hand side. It always starts with the start bit and is followed by the sections command (CMD), length (LEN), data (Data) and a checksum (CRC) - the least significant bit of a section is always towards the right-hand side.

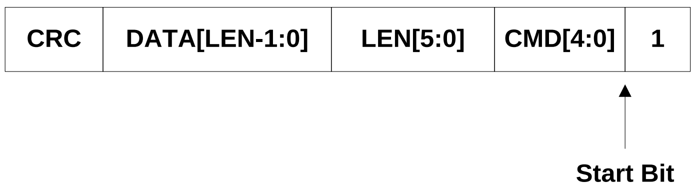

Figure 8 DAP telegram structure for direction tool to device

The debug XCP master uses the LLT command (see chapter 4.1.11) to transfer DAP telegrams to the XCP slave. Each DAP telegram is coded by a Low Level Telegram based on the mapping given in Figure 9.

It must be considered that the length information given in the LLT command is binary coded. The length section of the DAP telegram instead has a DAP protocol specific coding. The XCP slave shall convert the LLT length information into the protocol specific coding.

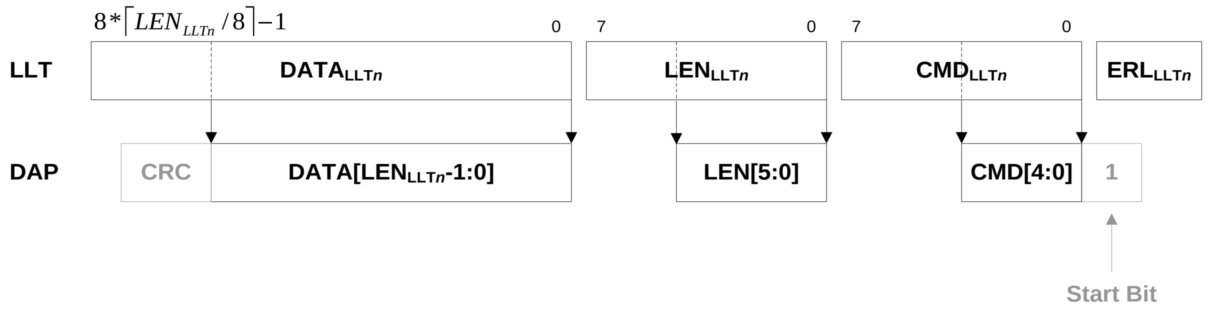

Figure 9 mapping between a LLT and a DAP telegram in target direction

Unused bits in the Low Level Telegram shall be zero filled. Neither the start bit nor the CRC are sent by the debug XCP master. The XCP slave shall generate this information when sending the telegram to the target.

<u>Direction target to debug XCP master</u>

When the target replies data it uses the telegram structure shown in Figure 10.

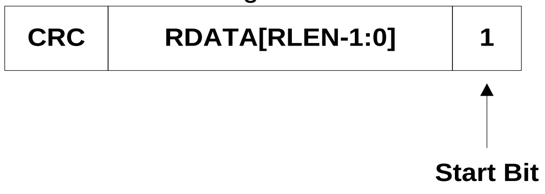

Figure 10 DAP telegram structure for direction device to tool

Again, a telegram is transmitted from right hand side to left hand side and starts with a start bit followed by the data section (RDATA) and an optional CRC section. The length of the received data must be known by the receiving side. Since the XCP slave is not aware of the expected response length related to a DAP telegram sent by the debug XCP master, the length is made available to the XCP slave by the *expected response length in bits* parameter for each LLT (see Table 36).

Additional specifics are coded by the debug XCP master using the LLT Mode parameter.

Table 88 LLT Mode parameter bitmask structure for DAP

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
×	×	×	×	×	×	×	Response on single lane

Table 89 LLT Mode parameter bitmask coding for DAP

Flag	Description
Response on single lane	0: target might send the response on several lanes in parallel 1: target replies on DAP1 only

The target might send the response solely on DAP 1 lane for specific DAP commands. The XCP slave might not be aware of these DAP commands. For that reason, the debug XCP master shall notify the XCP slave on which DAP lanes the response is sent by means of the LLT Mode parameter.

When the DAP telegram has been received the XCP slave maps this to the positive response for a LLT as shown in Figure 11. The start bit and the CRC are not sent to the debug XCP master. Unused bits in the RDATA section shall be zero filled by the XCP slave.

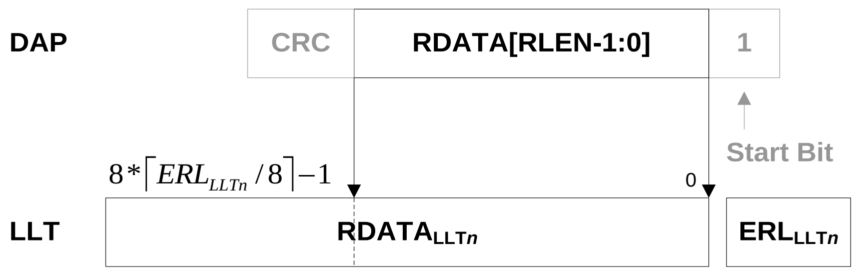
DAP
CRC
RDATA[RLEN-1:0]
1
Start Bit
$8*\lceil ERL_{LLTn}/8\rceil-1$
0
LLT
$\mathbf{RDATA}_{\mathbf{LLT}n}$
$\mathbf{ERL}_{\mathbf{LLT}n}$

Figure 11 mapping between a DAP and LLT for data received from target

Appendix: C. Explanation of Target Resource Identifier

When accessing memory using the commands defined in chapters 4.1.12 - 4.1.20, the Target Resource Identifier is used to distinguish different cores, access units, access modes, etc. It is specific to different microcontroller device families. Below, the Target Resource Identifiers (TRI) are defined for different targets.

C.1. Infineon DAP Devices

For Infineon devices with DAP interface (TriCore and XC2000 families), the TRI is used only for determining the Cerberus that shall be used for memory accesses. The following table lists the supported options:

Table 90 Target Resource Identifier classification for Infineon devices

Value	Description
1	Cerberus 1
2	Cerberus 2
Others	Reserved

C.2. MPC5xxx/SPC5xxx PowerPC Devices

For Freescale/NXP MPC5xxx and STMicroelectronics SPC5xxx devices, the TRI determines the Nexus client that shall be used for performing memory accesses. The XCP slave has to select the specified Nexus client using the appropriate JTAGC instruction. The following table lists the supported options:

Table 91 Target Resource Identifier classification for Power PC devices

Value	Description
0	Production Device (PD) area access - Core0
1	Production Device (PD) area access - Core1
2	Production Device (PD) area access - Core2
3	Production Device (PD) area access - Core3
128	Buddy Device (BD) area access - Read Write Client
Others	Reserved

C.3. Renesas RH850 Devices

For Renesas RH850 devices, it is necessary to differentiate up to 10 cores. Each core might have access to 3 different busses. In addition, a special mapping allows for improved performance when accessing debug registers. The following table lists the TRI mapping for Renesas RH850 devices:

Table 92 Target Resource Identifier classification for Renesas RH850 devices

Value	Description
0	Access to AXI bus via PE1 MAU
1	Access to AXI bus via PE2 MAU
…	
9	Access to AXI bus via PE10 MAU
32	Access to ELB bus via PE1 MAU
33	Access to ELB bus via PE2 MAU
…	
41	Access to ELB bus via PE10 MAU
64	Access to NT bus via PE1 MAU
65	Access to NT bus via PE2 MAU
…	
73	Access to NT bus via PE10 MAU
255	Debug register access
Others	Reserved

When the Debug register access mapping is used (TRI = 255) the byte address determines the parameters that are used for accessing a target debug register according to the following mapping:

Table 93 Debug register access address mapping

Bit numbers	Description
63..27	(reserved)
26	Access to DCU register set of PE10
25	Access to DCU register set of PE9
..	
17	Access to DCU register set of PE1
16	Access to JCU register set
15..12	Bank number
11..10	(reserved)
9..2	IR register number
1..0	(must be zero)

When the Debug register access is used the element width (EW) of the related commands must be set to value 0x4 (DWORD). It is allowed to access multiple, consecutive debug registers with one `DBG_READ` or `DBG_WRITE` command by setting the number of data elements (N) parameter to a value larger than 1.

Figure Directory

Figure 1 XCP Master accessing the ECU via a POD using XCP 6
Figure 2 XCP Master directly connected to the ECU using XCP....................................... 7
Figure 3 Tools accessing the ECU in parallel via a POD 7
Figure 4 Debugging in parallel with other XCP tools... 10
Figure 5 Step signals... 79
Figure 6 Data example if expected TDO matches. 80
Figure 7 Data example if expected TDO does not match.. 81
Figure 8 DAP telegram structure for direction tool to device 82
Figure 9 mapping between a LLT and a DAP telegram in target direction 82
Figure 10 DAP telegram structure for direction device to tool 83
Figure 11 mapping between a DAP and LLT for data received from target...................... 84

Table Directory

Table 1 Comparison of debug solutions 12
Table 2 Definition of the SW-DBG specific XCP command space 14
Table 3 SW-DBG command overview 14
Table 4 SW-DBG command structure 15
Table 5 SW-DBG command response packet structure 15
Table 6 SW-DBG error packet structure 16
Table 7 SW-DBG event packet structure 16
Table 8 DBG_ATTACH command structure 17
Table 9 DBG_ATTACH response structure 17
Table 10 DBG_GET_VENDOR_INFO command structure 18
Table 11 DBG_GET_VENDOR_INFO response structure 18
Table 12 DBG_GET_MODE_INFO command structure 19
Table 13 DBG_GET_MODE_INFO response structure 19
Table 14 Dialect parameter coding 19
Table 15 Feature parameter bitmask structure 20
Table 16 Feature parameter bitmask coding 20
Table 17 Service level parameter coding 20
Table 18 DBG_GET_JTAG_ID command structure 21
Table 19 DBG_GET_JTAG_ID response structure 21
Table 20 DBG_HALT_AFTER_RESET command structure 22
Table 21 DBG_HALT_AFTER_RESET negative response structure 22
Table 22 DBG_GET_HWIO_INFO command structure 23
Table 23 DBG_GET_HWIO_INFO response structure 23
Table 24 Mode parameter bitmask structure 24
Table 25 DBG_SET_HWIO_EVENT command structure 25
Table 26 DBG_HWIO_CONTROL command structure 26
Table 27 DBG_HWIO_CONTROL response structure 27
Table 28 DBG_HWIO_CONTROL negative response structure 27
Table 29 DBG_EXCLUSIVE_TARGET_ACCESS command structure 28
Table 30 Mode parameter coding 28
Table 31 Context parameter coding 28
Table 32 DBG_SEQUENCE_MULTIPLE command structure 29
Table 33 Mode parameter bitmask structure 29
Table 34 DBG_SEQUENCE_MULTIPLE positive response structure 30
Table 35 DBG_SEQUENCE_MULTIPLE negative response structure 31
Table 36 DBG_LLT command structure 32
Table 37 DBG_LLT positive response 33
Table 38 DBG_LLT negative response 33
Table 39 Reason parameter coding 34
Table 40 DBG_READ_MODIFY_WRITE command structure 35
Table 41 Element Width parameter coding 35
Table 42 DBG_READ_MODIFY_WRITE positive response structure 36
Table 43 DBG_WRITE command structure 37
Table 44 DBG_WRITE_NEXT command structure 39
Table 45 DBG_WRITE_NEXT negative response structure 39
Table 46 DBG_WRITE_CAN1 command structure 40
Table 47 DBG_WRITE_CAN2 command structure 41
Table 48 Element Width parameter coding for DBG_WRITE_CAN2 41
Table 49 DBG_WRITE_CAN2 negative response structure 41
Table 50 DBG_WRITE_CAN_NEXT command structure 43
Table 51 DBG_READ command structure 44
Table 52 DBG_READ positive response structure 44

Table 53 DBG_READ_CAN1 command structure 46
Table 54 DBG_READ_CAN2 command structure 47
Table 55 Generic structure of a negative response containing a SW-DBG specific error code 48
Table 56 SW-DBG specific error codes 48
Table 57 SW-DBG specific error handling 49
Table 58 Definition of the SW-DBG specific XCP event space 58
Table 59 SW-DBG events overview 58
Table 60 EV_DBG_SERVICE_LEVEL_CHANGE event packet 58
Table 61 EV_DBG_HWIO_CHANGE event packet 59
Table 62 Notation for indicating the packet direction 60
Table 63 Getting basic information 61
Table 64 Attaching to XCP slave with SW-DBG support (BYTE_ORDER = Motorola)..... 62
Table 65 Getting information about debugging properties of the XCP slave 62
Table 66 Getting the JTAG ID of the target (BYTE_ORDER = Motorola)........ 63
Table 67 Getting information about available HW-IO pins 63
Table 68 Asserting PORST and disabling watchdog (BYTE_ORDER = Motorola) 64
Table 69 Requesting the JTAG bus and going through RST to IDLE state (BYTE_ORDER = Intel) 64
Table 70 Write a huge JTAG register with preceding IR Tap setup (BYTE_ORDER = Intel) 65
Table 71 DBG_SEQUENCE_MULTIPLE negative response structure (BYTE_ORDER = Intel) 67
Table 72 LLT Example Sequence 1........ 68
Table 73 XCP Packets for LLT Example Sequence 1........ 68
Table 74 LLT Example Sequence 2........ 68
Table 75 XCP Packets for LLT Example Sequence 2........ 69
Table 76 LLT Example Sequence 3........ 69
Table 77 XCP Packets for LLT Example Sequence 3........ 69
Table 78 LLT Example Sequence 4........ 70
Table 79 XCP Packets for Example Telegram Sequence 4........ 70
Table 80 Reading four bytes at address 0x70000000 (BYTE_ORDER = Motorola)........ 70
Table 81 Reading four bytes at address 0x70000000 (BYTE_ORDER = Motorola)........ 71
Table 82 Writing four bytes to address 0x70000000 (BYTE_ORDER = Motorola)........ 71
Table 83 Writing four bytes to address 0x70000000 (BYTE_ORDER = Motorola)........ 72
Table 84 Step command structure........ 77
Table 85 Data command structure 78
Table 86 Step example........ 79
Table 87 Data example 80
Table 88 LLT Mode parameter bitmask structure for DAP 83
Table 89 LLT Mode parameter bitmask coding for DAP 83
Table 90 Target Resource Identifier classification for Infineon devices........ 85
Table 91 Target Resource Identifier classification for Power PC devices 85
Table 92 Target Resource Identifier classification for Renesas RH850 devices 86
Table 93 Debug register access address mapping 86

ASAM

Association for Standardisation of
Automation and Measuring Systems

E-mail:	support@asam.net
Web:	www.asam.net