

M-Core Debugger





Release 02.2025

MANUAL

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents	
ICD In-Circuit Debugger	
Processor Architecture Manuals	
M-Core	
M-Core Debugger	1
Introduction	5
Brief Overview of Documents for New Users	5
Demo and Start-up Scripts	6
Warning	7
Quick Start JTAG/ONCE	8
Breakpoints	10
Software Breakpoints	10
On-chip Breakpoints	10
Breakpoint in ROM	11
Example for Breakpoints	11
Troubleshooting	12
SYStem.Up Errors	12
Memory Access Errors	12
FAQ	12
Configuration	13
Runtime Measurement	13
Memory Classes	13
Memory Coherency	13
M-Core specific SYStem Commands for the Debugger	14
SYStem.CONFIG	14
SYStem.CONFIG	14
Configure debugger according to target topology	14
Daisy-Chain Example	17
TapStates	18
SYStem.CONFIG.CORE	19
Assign core to TRACE32 instance	19
SYStem.CPU	20
Selects the CPU	20
SYStem.JtagClock	20
Sets JTAG clock frequency	20
SYStem.LOCK	20
Tristate the JTAG port	20

SYStem.MemAccess	Select run-time memory access method	21
SYStem.Mode	Establish the communication with the target	22
SYStem.Option.DE	Stop CPU via debug enable line	23
SYStem.Option.DUALPORT	Update all memory displays during runtime	23
SYStem.Option.IMASKASM	Disable interrupts while single stepping	23
SYStem.Option.IMASKHLL	Disable interrupts while HLL single stepping	24
SYStem.Option.PC	Not supported command	24
SYStem.Option.TRST	Use TRST line to reset the TAP controller	24
Trigger On-chip Commands		25
TrOnchip.CYcle	Define access type	25
TrOnchip.A.Address	Define address selector	26
TrOnchip.EXTernal	Generate a trigger for trace on high pulse on in0 or in1	26
TrOnchip.Mode	Configure unit A and B	27
TrOnchip.RESet	Set on-chip trigger to default state	28
TrOnchip.state	Display on-chip trigger window	28
JTAG Connector		29
Technical Data		30
Operation Voltage		30

Introduction

This document describes the processor specific settings and features for TRACE32-ICD for the following CPU families:

- MCORE xx (2001, 2107, 2112, ...)

Please note that only the **Processor Architecture Manual** (the document you are currently reading) is specific to the core architecture. All other parts of the online help are general and independent of any core architecture. Therefore, if you have questions related to the core architecture, the **Processor Architecture Manual** should be your primary reference.

If some of the described functions, options, signals or connections in this Processor Architecture Manual are only valid for a single CPU or for specific families, the name(s) of the family(ies) is added in brackets.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Debugger Tutorial”** (debugger_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“General Commands”** (general_ref_<x>.pdf): Alphabetic list of debug commands.
- **“OS Awareness Manuals”** (rtos_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your Debug Cable. To access the manual for your processor architecture, proceed as follows:
 - Choose **Help** menu > **Processor Architecture Manual**.

-
- **“General Commands”** (general_ref_<x>.pdf): Alphabetic list of debug commands.
 - **“OS Awareness Manuals”** (rtos_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your Debug Cable. To access the manual for your processor architecture, proceed as follows:
 - Choose **Help** menu > **Processor Architecture Manual**.

Demo and Start-up Scripts

Lauterbach provides ready-to-run start-up scripts for known M-Core based hardware.

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.

You can also manually navigate in the `~/demo/mcore/` subfolder of the system directory of TRACE32.

WARNING:

To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF.

Recommendation for the software start:

1. Disconnect the Debug Cable from the target while the target power is off.
2. Connect the host system, the TRACE32 hardware and the Debug Cable.
3. Power ON the TRACE32 hardware.
4. Start the TRACE32 software to load the debugger firmware.
5. Connect the Debug Cable to the target.
6. Switch the target power ON.
7. Configure your debugger e.g. via a start-up script.

Power down:

1. Switch off the target power.
2. Disconnect the Debug Cable from the target.
3. Close the TRACE32 software.
4. Power OFF the TRACE32 hardware.

Quick Start JTAG/ONCE

Starting up the Debugger is done as follows:

5. Select the device prompt B: for the ICD Debugger.

```
B:
```

If you are working with the PODPC card, a Podbus-Ethernet interface or a Podbus-Parallel adapter device B: : is already selected.

6. Select the CPU type to load the CPU specific settings.

```
SYStem.CPU MMC2107
```

7. Tell the debugger where's FLASH/ROM on the target.

```
MAP.BOnchip 0x00000000++0x0001FFFF
```

This command is necessary for the use of on-chip breakpoints.

8. Enter debug mode

```
SYStem.Up
```

This command resets the CPU (RESET) and enters debug mode. After this command is executed it is possible to access the CPU registers. Set the chip selects to get access to the target memory.

9. Load the program.

```
Data.LOAD.ELF diabc.x          ; elf specifies the format, diabc.x  
                                ; is the file name
```

The option of the **Data.LOAD** command depends on the file format generated by the compiler. A detailed description of the **Data.LOAD** command is given in the “**General Commands Reference**”.

The start-up can be automated using the programming language PRACTICE. A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** <file>.

```
B::                                ; Select the ICD device prompt

WinCLEAR                           ; Clear all windows

MAP.BOnchip                        ; Specify where's ROM
0x00000000++0x0001FFFF

SYStem.CPU MMC2107                 ; Select the processor type

SYStem.Up                          ; Reset the target and enter debug mode

Data.LOAD.ELF diabc.x              ; Load the application

List.auto                          ; Open disassembly window          *)

Register.view /SpotLight           ; Open register window          *)

Frame.view /Locals /Caller         ; Open the stack frame with
; local variables                  *)

Var.Watch %Spotlight flags ast     ; Open watch window for variables *)

PER.view                          ; Open window with peripheral register

Break.Set sieve                   ; Set breakpoint to function sieve

Break.Set 0x800000 /Program         ; Set software breakpoint to address
; 800000
; (address 800000 is in RAM)

Break.Set 0x1000 /Program           ; Set on-chip breakpoint to address
; 1000 (address 1000 is in ROM)
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

Breakpoints

There are two types of breakpoints available: Software breakpoints and on-chip breakpoints.

Software Breakpoints

Software breakpoints are the default breakpoints for program breakpoints. A software breakpoint is implemented by patching a break code into the memory.

There is no restriction in the number of software breakpoints.

On-chip Breakpoints

The resources for the on-chip breakpoints are provided by the CPU.

The following list gives an overview of the on-chip breakpoints for the MCORE:

- **On-chip breakpoints:** Total amount of available on-chip breakpoints.
- **Instruction breakpoints:** Number of on-chip breakpoints that can be used to set Program breakpoints into ROM/FLASH/EEPROM.
- **Read/Write breakpoints:** Number of on-chip breakpoints that can be used as Read or Write breakpoints.
- **Data breakpoint:** Number of on-chip data breakpoints that can be used to stop the program when a specific data value is written to an address or when a specific data value is read from an address.

	On-chip Breakpoints	Instruction Breakpoints	Read/Write Breakpoints	Data Breakpoint
MCORE	2	2	2	—

Breakpoint in ROM

By default program breakpoints are implemented as software breakpoints.

With the command **MAP.BOnchip** *<range>* it is possible to inform the debugger where you have ROM (FLASH, EPROM) on the target. If a breakpoint is set within the specified address range the debugger uses automatically the available on-chip breakpoints.

Example for Breakpoints

Assume you have a target with FLASH from 0 to 0x1FFFF and RAM from 0x800000 to 0x801fff. The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x00000000++0x00001FFF
```

The following breakpoint combinations are possible.

Software breakpoints:

```
Break.Set 0x800000 /Program          ; Software Breakpoint 1
Break.Set 0x800100 /Program          ; Software Breakpoint 2
Break.Set 0x800f00 /Program          ; Software Breakpoint 3
```

On-chip breakpoints:

```
Break.Set 0x0100 /Program          ; On-chip Breakpoint 1
Break.Set 0x0ff00 /Program         ; On-chip Breakpoint 2
```

On-chip breakpoints on read or write access:

```
Break.Set 0x5678 /Write            ; On-chip Breakpoint 1
Var.Break.Set flags[3] /Read       ; On-chip Breakpoint 2
```

SYStem.Up Errors

The **SYStem.Up** command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

- The target has no power.
- The pull-up resistor between the JTAG[VCCS] pin and the target VCC is too large.
- The target is in reset:
The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up. Therefore no external R-C combination or external reset controller is allowed.
- There is logic added to the JTAG state machine:
By default the debugger supports only one processor in one JTAG chain. If the processor is only one member of a JTAG chain the debugger has to be informed about the target JTAG chain configuration.
- There are additional loads or capacities on the JTAG lines

Memory Access Errors

After system up is completed successfully, data can be written to or read from memory. Trying to access memory not belonging to the memory map of the processor will be refused with the error message:

```
no memory mapped at address      D:0002000
```

FAQ

Please refer to <https://support.lauterbach.com/kb>.

Runtime Measurement

The command **RunTime** allows run time measurement based on polling the CPU run status by software. Therefore the result will be about few milliseconds higher than the real value.

If the signal DE is available on the JTAG connector, the measurement will automatically be based on this hardware signal which delivers very exact results. Please do not disable the option **SYStem.Option.DE**.

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
D	Data
NC	No Cache (only physically memory)

Memory Coherency

Memory coherency on access to following memory classes.

	Physical Memory
NC:	Yes
D:	Yes
P:	Yes

Restarting the CPU afterwards.

SYStem.CONFIG

The **SYStem.CONFIG** commands are used to configure the behavior of the complete target system for debugging, e.g., the Debug Interface or the chaining of several CPUs.

This command replaces the **SYStem.MultiCore** command and uses exactly the same parameters.

SYStem.CONFIG

Configure debugger according to target topology

Format:	SYStem.CONFIG <parameter> <number_or_address> SYStem.MultiCore <parameter> <number_or_address> (deprecated)
<parameter>:	CORE <core>
<parameter>: (JTAG):	DRPRE <bits> DRPOST <bits> IRPRE <bits> IRPOST <bits> TAPState <state> TCKLevel <level> TriState [ON OFF] Slave [ON OFF]

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. Arm + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See **Daisy-chain Example**.

For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

TriState has to be used if several debuggers (“via separate cables”) are connected to a common JTAG port at the same time in order to ensure that always only one debugger drives the signal lines. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode. Please note: nTRST must have a pull-up resistor on the target, TCK can have a pull-up or pull-down resistor, other trigger inputs need to be kept in inactive state.



Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).

CORE

For multicore debugging one TRACE32 PowerView GUI has to be started per core. To bundle several cores in one processor as required by the system this command has to be used to define core and processor coordinates within the system topology. Further information can be found in [SYSstem.CONFIG.CORE](#).

DRPRE

(default: 0) *<number>* of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.

DRPOST

(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.

IRPRE

(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.

IRPOST

(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.

TAPState

(default: 7 = Select-DR-Scan) This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.

TCKLevel

(default: 0) Level of TCK signal when all debuggers are tristated.

TriState

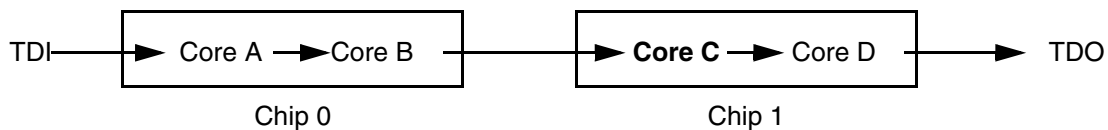
(default: OFF) If several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.

Slave

(default: OFF) If more than one debugger share the same debug port, all except one must have this option active.

JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).

Daisy-Chain Example



Below, configuration for core C.

Instruction register length of

- Core A: 3 bit
- Core B: 5 bit
- Core D: 6 bit

```
SYStem.CONFIG.IRPRE 6. ; IR Core D
SYStem.CONFIG.IRPOST 8. ; IR Core A + B
SYStem.CONFIG.DRPRE 1. ; DR Core D
SYStem.CONFIG.DRPOST 2. ; DR Core A + B
SYStem.CONFIG.CORE 0. 1. ; Target Core C is Core 0 in Chip 1
```

0	Exit2-DR
1	Exit1-DR
2	Shift-DR
3	Pause-DR
4	Select-IR-Scan
5	Update-DR
6	Capture-DR
7	Select-DR-Scan
8	Exit2-IR
9	Exit1-IR
10	Shift-IR
11	Pause-IR
12	Run-Test/Idle
13	Update-IR
14	Capture-IR
15	Test-Logic-Reset

Format: **SYStem.CONFIG.CORE** <core_index> <chip_index>
 SYStem.MultiCore.CORE <core_index> <chip_index> (deprecated)

<chip_index>: 1 ... i

<core_index>: 1 ... k

Default *core_index*: depends on the CPU, usually 1. for generic chips

Default *chip_index*: derived from CORE= parameter of the configuration file (config.t32). The CORE parameter is defined according to the start order of the GUI in T32Start with ascending values.

To provide proper interaction between different parts of the debugger, the systems topology must be mapped to the debugger's topology model. The debugger model abstracts chips and sub cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected, a generic chip or non-generic chip is created at the default *chip_index*.

Non-generic Chips

Non-generic chips have a fixed number of sub cores, each with a fixed CPU type.

Initially, all GUIs are configured with different *chip_index* values. Therefore, you have to assign the *core_index* and the *chip_index* for every core. Usually, the debugger does not need further information to access cores in non-generic chips, once the setup is correct.

Generic Chips

Generic chips can accommodate an arbitrary amount of sub-cores. The debugger still needs information how to connect to the individual cores e.g. by setting the JTAG chain coordinates.

Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a non-generic chip, two GUIs are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is valid since every new GUI uses a new *chip_index* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

Format:	SYStem.CPU <cpu>
<cpu>:	MMC2001 MMC2107 MMC2112 MMC2113 MMC2114 M310 M340 M3401

Selects the processor type.

SYStem.JtagClock

Sets JTAG clock frequency

Format:	SYStem.JtagClock <rate> SYStem.BdmClock (deprecated)
<fixed>:	1 000 000. ... 15 000 000.

Selects the frequency for the debug interface.

The default frequency is 1 MHz.

NOTE:	Buffers, additional loads or high capacities on the JTAG/COP lines can reduce the debug speed.
--------------	--

SYStem.LOCK

Tristate the JTAG port

Format:	SYStem.LOCK [ON OFF]
---------	-------------------------------

Default: OFF.

LOCK must be switched on, if several debuggers are used to debug several Cores using the same JTAG connector. By locking the debug lines for certain cores another debugger can own mastership on the JTAG interface.

It must be ensured that the state of the MCore JTAG state machine remains unchanged while the system is locked.

Format:	SYStem.MemAccess Enable StopAndGo Denied <cpu_specific> SYStem.ACCESS (deprecated)
---------	---

Enable CPU (deprecated)	Memory access during program execution to target is enabled.
Denied (default)	Memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

This option declares if and how a non-intrusive memory access can take place while the CPU is executing code. Although the CPU is not halted, run-time memory access creates an additional load on the processor's internal bus.

The run-time memory access has to be activated for each window by using the memory class E: (e.g. Data.dump E:0x100) or by using the format option %E (e.g. Var.View %E var1). It is also possible to activate this non-intrusive memory access for all memory ranges displayed on the TRACE32 screen by setting **SYStem.Option.DUALPORT ON**.

Format: **SYStem.Mode** <mode>

<mode>:
Down
NoDebug
Go
Attach
Up
StandBy

The communication between the debugger and the target is established.

Down	Disables the debugger. The state of the CPU remains unchanged.
NoDebug	Resets the target with debug mode disabled. The CPU acts like no debugger is connected.
Go	Resets the target with debug mode enabled. Afterwards the CPU starts executing the code. The CPU can be stopped with the break command or until any break condition occurs.
Attach	User program remains running (no reset) and the debug mode is activated. After this command the user program can be stopped with the break command or if any break condition occurs.
Up	Resets the target (HRESET line) and sets the CPU to debug mode. The CPU stops at the reset vector afterwards.
StandBy	Not implemented.

Format:	SYStem.Option.DE [ON OFF]
---------	------------------------------------

- ON (default)

The debugger stops the CPU via the Debug Enable line.
- OFF

The debugger stops the CPU by transferring a special command via the JTAG interface.
OFF is only recommended if stopping the CPU via the Debug Enable line is not possible.

SYStem.Option.DUALPORT

Update all memory displays during runtime

Format:	SYStem.Option.DUALPORT [ON OFF]
---------	--

Default: OFF.

- ON
(NEXUS only)

Switches the run-time memory access to ON for all windows that display memory. Use this option if you want all windows to be updated while the processor is running. This setting has no effect if **SYStem.Option.MemAccess** is denied.

SYStem.Option.IMASKASM

Disable interrupts while single stepping

Format:	SYStem.Option.IMASKASM [ON OFF]
---------	--

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during assembler single-step operations. The result is that interrupts are not accepted during single-step operations. After each single step the interrupt mask bits are restored to the value before the step.

Format:

SYStem.Option.IMASKHLL [ON | OFF]

Default: OFF.

If enabled, the interrupt mask bits of the CPU will be set during HLL single-step operations. The result is that interrupts are not accepted during HLL single-step operations. After each HLL single step the interrupt mask bits are restored to the value before the step.

Format:

SYStem.Option.PC <address>

Default address: 0N.

Not supported.

Format:

SYStem.Option.TRST [ON | OFF]

- ON** (default)

Use the TRST line to reset the TAP controller.
- OFF**

The TAP will be reset by shifting in an adequate TMS sequence.
OFF is only recommended if resetting the TAP controller via the TRST line is not possible.

TrOnchip.CYcle

Define access type

Format:	TrOnchip.A.CYcle <cycle> TrOnchip.B.CYcle <cycle>
<cycle>:	ANY Read Write Access Execute

Defines on which cycle the ICE breaker stops the program execution.

ANY	Cycle type doesn't matter.
Read	Stop the program execution on a read access.
Write	Stop the program execution on a write access.
Access	Stop the program execution on a read or write access.
Execute	Stop the program execution on an instruction is executed.

Format:	TrOnchip.A.Address <selector> TrOnchip.B.Address <selector>
<selector>:	OFF Alpha Beta Charly

The address/range for an address selector can not be defined directly. Set an breakpoint of the type Alpha, Beta or Charly to the address/range.

OFF	No trigger signal will be generated.
Alpha	Use Alpha breakpoint as address selector for selected unit.
Beta	Use Beta breakpoint as address selector for selected unit.
Charly	Use Charly breakpoint as address selector for selected unit.

```
Break.Set 1000 /Alpha           ; set an Alpha breakpoint to 1000
TrOnchip.A.Address Alpha       ; use Alpha breakpoint as address
                               ; selector for the unit A

Var.Break.Set flags[3] /Beta   ; set a Beta breakpoint to flags[3]
TrOnchip.B.Address Beta       ; use Beta breakpoint as address
                               ; selector for the unit B
```

TrOnchip.EXTERNAL

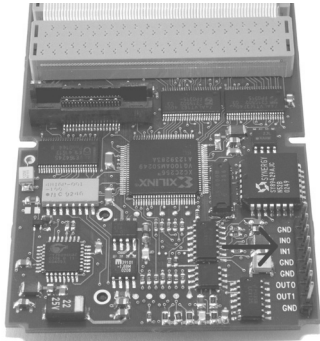
Generate a trigger for trace on high pulse on in0 or in1

Format:	TrOnchip EXTERNAL OFF IN0 IN1
---------	--

Generates a trigger for the trace on a high pulse (at least 20 ns) on the IN0 or IN1 connector of the NEXUS adapter. IN0 and IN1 are ORed for the trigger.

OFF	No trigger signal will be generated.
------------	--------------------------------------

- IN0** Generate a trigger signal on IN0.
- IN1** Generate a trigger signal on IN1.



Example: Stop the sampling to the trace if a higher pulse is recognized at IN0.

```
TrOnchip.EXTERNAL IN0      ; Enable IN0 as trigger source for the trace
Go                          ; Start the program execution
```

TrOnchip.Mode

Configure unit A and B

Format: TrOnchip.Mode <mode>

<mode>: AORB
 AANDB
 BAFTERA

Not yet implemented.

Defines the way in which unit A and B are used together.

AORB	Stop the program execution if unit A or unit B match.
AANDB	Stop the program execution if both units match.
BAFTERA	Stop the program execution if first unit A and then unit B match.

TrOnchip.RESet

Set on-chip trigger to default state

Format:	TrOnchip.RESet
---------	-----------------------

Sets the TrOnchip settings and trigger module to the default settings.

TrOnchip.state

Display on-chip trigger window

Format:	TrOnchip.state
---------	-----------------------

Opens the **TrOnchip.state** window.

JTAG Connector

Signal	Pin	Pin	Signal
TDI	1	2	GND
TDO	3	4	GND
TCLK	5	6	GND
(GPIO-SI-)	7	8	N/C
RESET-	9	10	TMS
VDD TARGET	11	12	DE-
(!GPIO-SO)	13	14	TRST-

This is a standard 14 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.).
(Signals in brackets are not strong necessary for basic debugging, but its recommended to take in consideration for future designs.)

Operation Voltage

This list contains information on probes available for other voltage ranges. Probes not noted here supply an operation voltage range of 4.5 ... 5.5 V.

Adapter	OrderNo	Voltage Range
ONCE Debugger for M-CORE (ICD)	LA-7745	2.5 .. 5.5 V

•