LAUTERBACH
DEVELOPMENT TOOLS

APEX Debugger

MANUAL

APEX Debugger

Version 06-Jun-2024

Introduction

This manual serves as a guideline for debugging and describes all processor-specific TRACE32 settings and features.

Please keep in mind that only the **Processor Architecture Manual** (the document you are reading at the moment) is CPU specific, while all other parts of the online help are generic for all CPUs supported by Lauterbach. So if there are questions related to the CPU, the Processor Architecture Manual should be your first choice.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Training Basic Debugging”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.

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**.
- **“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.

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 of the JTAG Debugger

Starting up the debugger is done as follows:

1. Select the device prompt for the ICD Debugger and reset the system.

   ```
   B::

   RESet
   ```

   The device prompt `B::` is normally already selected in the TRACE32 command line. If this is not the case, enter `B::` to set the correct device prompt. The **RESet** command is only necessary if you do not start directly after booting the TRACE32 development tool.

2. Inform TRACE32 about the CPU under debug.

   ```
   SYStem.CPU APEX321
   ```

   The other settings in the **SYStem** window are set in such a way that it should be possible to work without modification. Please consider that this might not be the best configuration for your target.

3. Specify address ranges where the access width is restricted.

   ```
   MAP.BUS32 0x060000000++1FFFF
   ```

   If a memory range can only be accessed with a certain bus width you can use the **Map.BUSx** *<range>* command to force the debugger to use solely the according load or store instructions.

4. Establish the debug communication..

   ```
   ; issue a core reset but no SoC reset
   SYStem.Mode.Up
   ```

5. Load the program.

```
Data.LOAD.auto myproject /Long    ;load the compiler output
                                  ;the option /Long advises the
                                  ;debugger to use 32-bit accesses
                                  ;while loading the code to the
                                  ;target
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler.

A detailed description of the **Data.LOAD** command and all available options is given in the **"General Commands Reference"**.

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.BUS32 0x50000000++0x1ffff       ; Force the debugger to access the
                                    ; specified address range 32-bit wide

SYStem.Up                           ; Establish the debug communication

Data.LOAD.elf apex_project          ; Load the application

Register.Set PC _ResetVector        ; Set the PC to start point

Register.Set A1 0x63FFFFFC          ; Set the stack pointer to address
                                    ; 0x63FFFFFC

List.Mix                            ; Open source code window     *)

Register.view /SpotLight            ; Open register window        *)

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

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

Troubleshooting

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 target is in reset.
- The APEX core is not enabled.
- There is logic added to the JTAG state machine.
- There are additional loads or capacities on the JTAG lines.
- There is a short circuit on at least one of the output lines of the core.

FAQ

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

APEX Specific Implementations

Breakpoints

Software Breakpoints

If a software breakpoint is used, the original code at the breakpoint location is patched by a breakpoint code.

On-chip Breakpoints for Instructions

If on-chip breakpoints are used, the resources to set the breakpoints are provided by the CPU. On-chip breakpoints are usually needed to set Program breakpoints to instructions in FLASH. The APEX supports up to four On-chip breakpoints on single addresses, address ranges are not supported.

Example for Standard Breakpoints

TRACE32 automatically uses the appropriate breakpoint implementation.

```
Break.Set P:0x100000 /Program         ; Software breakpoint 1

Break.Set P:0x101000 /Program         ; Software breakpoint 2

Break.Set func6 /Program              ; Software breakpoint 3
```

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.

Memory Classes

The following APEX specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
V	Vector Memory
E	Run-time memory access (see **SYStem.CpuAccess** and **SYStem.MemAccess**)

To access a memory class, write the class in front of the address.

Example:

```
Data.dump D:0x0--0x3
```

Bus Width Mapping

MAP.BUS8	Forces the debugger to access the specified range with Load / Store 8-bit commands.
MAP.BUS16	Forces the debugger to access the specified range with Load / Store 16-bit commands.
MAP.BUS32	Forces the debugger to access the specified range with Load / Store 32-bit commands.

CPU specific SYStem Commands

Please be aware that if the APEX is used in a multicore debug environment with an Arm core, most **SYSTem** settings are already done by this core. But since the APEX debugger is using the same debug interface, the settings are also visible in the APEX debugger GUI.

SYStem.CONFIG.state — Display target configuration

Format:	**SYStem.CONFIG.state** [**/***<tab>*]
<tab>:	**DebugPort** | **Jtag** | **DAP** | **COmponents**

Opens the **SYStem.CONFIG.state** window, where you can view and modify most of the target configuration settings. The configuration settings tell the debugger how to communicate with the chip on the target board and how to access the on-chip debug and trace facilities in order to accomplish the debugger's operations.

Alternatively, you can modify the target configuration settings via the TRACE32 command line with the **SYStem.CONFIG** commands. Note that the command line provides *additional* **SYStem.CONFIG** commands for settings that are *not* included in the **SYStem.CONFIG.state** window.

<tab>	Opens the **SYStem.CONFIG.state** window on the specified tab. For tab descriptions, see below.
DebugPort (default)	The **DebugPort** tab informs the debugger about the debug connector type and the communication protocol it shall use. For descriptions of the commands on the **DebugPort** tab, see **DebugPort**.
Jtag	The **Jtag** tab informs the debugger about the position of the Test Access Ports (TAP) in the JTAG chain which the debugger needs to talk to in order to access the debug and trace facilities on the chip. For descriptions of the commands on the **Jtag** tab, see **Jtag**.
DAP	The **DAP** tab informs the debugger about an ARM CoreSight Debug Access Port (DAP) and about how to control the DAP to access chip-internal memory busses (AHB, APB, AXI) or chip-internal JTAG interfaces. For descriptions of the commands on the **DAP** tab, see **DAP**.

COmponents	The **COmponents** tab informs the debugger (a) about the existence and interconnection of on-chip CoreSight debug and trace modules and (b) informs the debugger on which memory bus and at which base address the debugger can find the control registers of the modules. For descriptions of the commands on the **COmponents** tab, see **COmponents**.

Format:	**SYStem.CONFIG** *<parameter>* **SYStem.MultiCore** *<parameter>* (deprecated)
<parameter>: **(DebugPort)**	**CJTAGFLAGS** *<flags>* **CJTAGTCA** *<value>* **CORE** *<core>* *<chip>* **DEBUGPORTTYPE** [**JTAG** | **SWD** | **CJTAG**]
<parameter>: (DebugPort cont.)	**Slave** [**ON** | **OFF**] **SWDP** [**ON** | **OFF**] **SWDPIDLEHIGH** [**ON** | **OFF**] **SWDPTargetSel** *<value>* **DAP2SWDPTargetSel** *<value>* **TriState** [**ON** | **OFF**] **DAPDRPOST** *<bits>* **DAPDRPRE** *<bits>* **DAPIRPOST** *<bits>* **DAPIRPRE** *<bits>* **DRPOST** *<bits>* **DRPRE** *<bits>* **IRPOST***<bits>* **IRPRE** *<bits>* **Slave** [**ON** | **OFF**] **TAPState** *<state>* **TCKLevel** *<level>*
<parameter>: **(DAP)**	**AHBACCESSPORT** *<port>* **APBACCESSPORT** *<port>* **AXIACCESSPORT** *<port>* **DEBUGACCESSPORT** *<port>* **MEMORYACCESSPORT** *<port>* **JTAGACCESSPORT** *<port>*
<parameter>: **(COmponents)**	**COREDEBUG.Base** *<address>* **COREDEBUG.RESet** **COREDEBUG.view** **CTI.Base** *<address>* **CTI.Config** [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**] **CTI.RESet** **CTI.view** **ETB.ATBSource** *<source>*

<parameter>:
(COmponents cont.)

ETB.Base *<address>*
ETB.Name *<string>*
ETB.NoFlush [**ON** | **OFF**]
ETB.RESet
ETB.Size *<size>*
ETB.STackMode [**NotAvailbale** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULL STOP** | **FULLCTI**]
ETB.view

ETF.ATBSource *<source>*
ETF.Base *<address>*
ETF.Name *<string>*
ETF.NoFlush [**ON** | **OFF**]
ETF.RESet
ETF.Size *<size>*
ETF.STackMode [**NotAvailbale** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULL STOP** | **FULLCTI**]
ETF.view

ETR.ATBSource *<source>*
ETR.Base *<address>*
ETR.CATUBase *<address>*
ETR.Name *<string>*
ETR.NoFlush [**ON** | **OFF**]
ETR.RESet
ETR.Size *<size>*
ETR.STackMode [**NotAvailbale** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULL STOP** | **FULLCTI**]
ETR.view

ETS.ATBSource *<source>*
ETS.Base *<address>*
ETS.Name *<string>*
ETS.NoFlush [**ON** | **OFF**]
ETS.RESet
ETS.Size *<size>*
ETS.STackMode [**NotAvailbale** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULL STOP** | **FULLCTI**]
ETS.view

FUNNEL.ATBSource *<sourcelist>*
FUNNEL.Base *<address>*
FUNNEL.Name *<string>*
FUNNEL.PROGrammable [**ON** | **OFF**]
FUNNEL.RESet
FUNNEL.view

REP.ATBSource *<source>*
REP.Base *<address>*
REP.Name *<string>*
REP.RESet
REP.view

The **SYStem.CONFIG** commands inform the debugger about the available on-chip debug and trace components and how to access them.

Ideally you can select with **SYStem.CPU** the chip you are using which causes all setup you need and you do not need any further **SYStem.CONFIG** command.

The **SYStem.CONFIG** command information shall be provided after the **SYStem.CPU** command, which might be a precondition to enter certain **SYStem.CONFIG** commands, and before you start up the debug session e.g. by **SYStem.Up**.

CJTAGFLAGS *<flags>*	Activates bug fixes for "cJTAG" implementations. Bit 0: Disable scanning of cJTAG ID. Bit 1: Target has no "keeper". Bit 2: Inverted meaning of SREDGE register. Bit 3: Old command opcodes. Bit 4: Unlock cJTAG via APFC register. Default: 0
CJTAGTCA *<value>*	Selects the TCA (TAP Controller Address) to address a device in a cJTAG Star-2 configuration. The Star-2 configuration requires a unique TCA for each device on the debug port.
CORE *<core> <chip>*	The command helps to identify debug and trace resources which are commonly used by different cores. The command might be required in a multicore environment if you use multiple debugger instances (multiple TRACE32 PowerView GUIs) to simultaneously debug different cores on the same target system. Because of the default setting of this command debugger#1: *<core>*=1 *<chip>*=1 debugger#2: *<core>*=1 *<chip>*=2 ... each debugger instance assumes that all notified debug and trace resources can exclusively be used. But some target systems have shared resources for different cores, for example a common trace port. The default setting causes that each debugger instance controls the same trace port. Sometimes it does not hurt if such a module is controlled twice. But sometimes it is a must to tell the debugger that these cores share resources on the same *<chip>*. Whereby the "chip" does not need to be identical with the device on your target board: debugger#1: *<core>*=1 *<chip>*=1 debugger#2: *<core>*=2 *<chip>*=1
CORE *<core> <chip>* (cont.)	For cores on the same *<chip>,* the debugger assumes that the cores share the same resource if the control registers of the resource have the same address. Default: *<core>* depends on CPU selection, usually 1. *<chip>* derived from `CORE=` parameter in the configuration file (config.t32), usually 1. If you start multiple debugger instances with the help of t32start.exe, you will get ascending values (1, 2, 3,...).

DEBUGPORTTYPE [**JTAG** | **SWD** | **CJTAG**]

It specifies the used debug port type “JTAG”, “SWD”, “CJTAG”, “CJTAG-SWD”. It assumes the selected type is supported by the target.

Default: JTAG.

Slave [**ON** | **OFF**]

If several debuggers 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). The other debuggers need to have the setting **Slave ON**.

Default: OFF.
Default: ON if `CORE=`... >1 in the configuration file (e.g. config.t32).

SWDP [**ON** | **OFF**]

With this command you can change from the normal JTAG interface to the serial wire debug mode. SWDP (Serial Wire Debug Port) uses just two signals instead of five. It is required that the target and the debugger hard- and software supports this interface.

Default: OFF.

SWDPIdleHigh [**ON** | **OFF**]

Keep SWDIO line high when idle. Only for Serialwire Debug mode. Usually the debugger will pull the SWDIO data line low, when no operation is in progress, so while the clock on the SWCLK line is stopped (kept low).

You can configure the debugger to pull the SWDIO data line high, when no operation is in progress by using
SYStem.CONFIG SWDPIdleHigh ON

Default: OFF.

SWDPTargetSel *<value>*

Device address in case of a multidrop serial wire debug port.

Default: none set (any address accepted).

TriState [**ON** | **OFF**]

TriState has to be used if several debug cables are connected to a common JTAG port. 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.

Default: OFF.

With the JTAG interface you can access a Test Access Port controller (TAP) which has implemented a state machine to provide a mechanism to read and write data to an Instruction Register (IR) and a Data Register (DR) in the TAP. The JTAG interface will be controlled by 5 signals:

- nTRST (reset)
- TCK (clock)
- TMS (state machine control)
- TDI (data input)
- TDO (data output)

Multiple TAPs can be controlled by one JTAG interface by daisy-chaining the TAPs (serial connection). If you want to talk to one TAP in the chain, you need to send a BYPASS pattern (all ones) to all other TAPs. For this case the debugger needs to know the position of the TAP it wants to talk to. The TAP position can be defined with the first four commands in the table below.

… **DRPOST** *<bits>*	Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TDI signal and the TAP you are describing. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below. Default: 0.
… **DRPRE** *<bits>*	Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TAP you are describing and the TDO signal. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below. Default: 0.
… **IRPOST** *<bits>*	Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between TDI signal and the TAP you are describing. See possible TAP types and example below. Default: 0.
… **IRPRE** *<bits>*	Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between the TAP you are describing and the TDO signal. See possible TAP types and example below. Default: 0.

NOTE:	If you are not sure about your settings concerning **IRPRE**, **IRPOST**, **DRPRE**, and **DRPOST**, you can try to detect the settings automatically with the **SYStem.DETECT.DaisyChain** command.

Slave [**ON** | **OFF**]	If several debuggers 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). The other debuggers need to have the setting **Slave OFF**. Default: OFF. Default: ON if `CORE=`... >1 in the configuration file (e.g. config.t32). For CortexM: Please check also **SYStem.Option.DISableSOFTRES [ON | OFF]**
TAPState *<state>*	This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable. 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 Default: 7 = Select-DR-Scan.
TCKLevel *<level>*	Level of TCK signal when all debuggers are tristated. Normally defined by a pull-up or pull-down resistor on the target. Default: 0.

TAP types:

Core TAP providing access to the debug register of the core you intend to debug.
-> DRPOST, DRPRE, IRPOST, IRPRE.

DAP (Debug Access Port) TAP providing access to the debug register of the core you intend to debug. It might be needed additionally to a Core TAP if the DAP is only used to access memory and not to access the core debug register.
-> DAPDRPOST, DAPDRPRE, DAPIRPOST, DAPIRPRE.

<parameters> configuring a CoreSight Debug Access Port “DAP”

A Debug Access Port (DAP) is a CoreSight module from ARM which provides access via its debugport (JTAG, cJTAG, SWD) to:

1. Different memory busses (AHB, APB, AXI). This is especially important if the on-chip debug register needs to be accessed this way. You can access the memory buses by using certain access classes with the debugger commands: “AHB:”, “APB:”, “AXI:, “DAP”, “E:”. The interface to these buses is called Memory Access Port (MEM-AP).

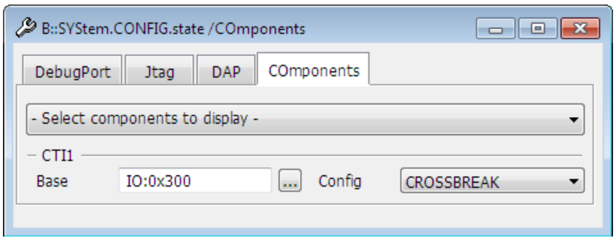
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number between 0 and 7 denotes the JTAG interfaces to be addressed.

3. At emulation or simulation system with using bus transactors the access to the busses must be specified by using the transactor identification name instead using the access port commands. For emulations/simulations with a DAP transactor the individual bus transactor name don’t need to be configured. Instead of this the DAP transactor name need to be passed and the regular access ports to the busses.

AHBACCESSPORT *<port>*	DAP access port number (0-255) which shall be used for “AHB:” access class. Default: *<port>*=0.
APBACCESSPORT *<port>*	DAP access port number (0-255) which shall be used for “APB:” access class. Default: *<port>*=1.
AXIACCESSPORT *<port>*	DAP access port number (0-255) which shall be used for “AXI:” access class. Default: port not available
DEBUGACCESSPORT *<port>*	DAP access port number (0-255) where the debug register can be found (typically on APB). Used for “DAP:” access class. Default: *<port>*=1.
MEMORYACCESSPORT *<port>*	DAP access port number where system memory can be accessed even during runtime (typically on AHB). Used for “E:” access class while running, assuming “SYStem.MemoryAccess DAP”. Default: *<port>*=0.
JTAGACCESSPORT *<port>*	DAP access port number (0-255) of the JTAG Access Port.

<parameters> describing debug and trace “Components”

On the **Components** tab in the **SYStem.CONFIG.state** window, you can comfortably add the debug and trace components your chip includes and which you intend to use with the debugger’s help.


B::SYStem.CONFIG.state /COmponents
DebugPort
Jtag
DAP
COmponents
- Select components to display -
...

CTI1
Base IO:0x300 Config CROSSBREAK

Each configuration can be done by a command in a script file as well. Then you do not need to enter everything again on the next debug session. If you press the button with the three dots you get the corresponding command in the command line where you can view and maybe copy it into a script file.

You can have several of the following components: ETB, ETF, ETR, FUNNEL.
Example: FUNNEL1, FUNNEL2, FUNNEL3,...

The *<address>* parameter can be just an address (e.g. 0x80001000) or you can add the access class in front (e.g. AHB:0x80001000). Without access class it gets the command specific default access class which is “EDAP:” in most cases.

… **.ATBSource** *<source>*	Specify for components collecting trace information from where the trace data are coming from. This way you inform the debugger about the interconnection of different trace components on a common trace bus. You need to specify the “... .Base *<address>*” or other attributes that define the amount of existing peripheral modules before you can describe the interconnection by “... .ATBSource *<source>*”. A CoreSight trace FUNNEL has eight input ports (port 0-7) to combine the data of various trace sources to a common trace stream. Therefore you can enter instead of a single source a list of sources and input port numbers. For a list of possible components including a short description see **Components and Available Commands**.
… **.BASE** *<address>*	This command informs the debugger about the start address of the register block of the component. And this way it notifies the existence of the component. An on-chip debug and trace component typically provides a control register block which needs to be accessed by the debugger to control this component. **Example**: SYStem.CONFIG ETB.Base APB:0x8011c000 Meaning: The control register block of the Embedded Trace Buffer (ETB) starts at address 0x8011c000 and is accessible via APB bus. For a list of possible components including a short description see **Components and Available Commands**.
... **.Name**	The name is a freely configurable identifier to describe how many instances exists in a target systems chip. TRACE32 PowerView GUI shares with other opened PowerView GUIs settings and the state of components identified by the same name and component type. Components using different names are not shared. Other attributes as the address or the type are used when no name is configured.

	Example 1: **Shared None-Programmable Funnel:** PowerView1: SYStem.CONFIG.FUNNEL.PROGramable OFF SYStem.CONFIG.FUNNEL.Name "shared-funnel-1" PowerView2: SYStem.CONFIG.FUNNEL.PROGramable OFF SYStem.CONFIG.FUNNEL.Name "shared-funnel-1" SYStem.CONFIG.Core 2. 1. ; merge configuration to describe a target system with one chip containing a single none-programmable FUNNEL. **Example 2: Cluster ETFs**: 1. Configures the ETF base address and access for each core SYStem.CONFIG.ETF.Base DAP:0x80001000 \\ APB:0x80001000 DAP:0x80001000 APB:0x80001000 2. Tells the system the core 1 and 3 share cluster-etf-1 and core 2 and 4 share cluster-etf-2 despite using the same address for all ETFs SYStem.CONFIG.ETF.Name "cluster-etf-1" "cluster-etf-2" \\ "cluster-etf-1" "cluster-etf-2"
… **.NoFlush** [**ON** | **OFF**]	Deactivates a component flush request at the end of the trace recording. This is a workaround for a bug on a certain chip. You will loose trace data at the end of the recording. Don’t use it if not needed. Default: OFF.
… **.RESet**	Undo the configuration for this component. This does not cause a physical reset for the component on the chip. For a list of possible components including a short description see **Components and Available Commands**.
… **.Size** *<size>*	Specifies the size of the component. The component size can normally be read out by the debugger. Therefore this command is only needed if this can not be done for any reason.
… **.STackMode** [**NotAvailbale** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULLSTOP** | **FULLCTI**]	Specifies the which method is used to implement the Stack mode of the on-chip trace. **NotAvailable**: stack mode is not available for this on-chip trace. **TRGETM**: the trigger delay counter of the onchip-trace is used. It starts by a trigger signal that must be provided by a trace source. Usually those events are routed through one or more CTIs to the on-chip trace. **FULLTIDRM**: trigger mechanism for TI devices. **NOTSET**: the method is derived by other GUIs or hardware. detection. **FULLSTOP**: on-chip trace stack mode by implementation. **FULLCTI**: on-chip trace provides a trigger signal that is routed back to on-chip trace over a CTI.

… **.view**	Opens a window showing the current configuration of the component. For a list of possible components including a short description see **Components and Available Commands**.
CTI.Config *<type>*	Informs about the interconnection of the core Cross Trigger Interfaces (CTI). Certain ways of interconnection are common and these are supported by the debugger e.g. to cause a synchronous halt of multiple cores. NONE: The CTI is not used by the debugger. ARMV1: This mode is used for ARM7/9/11 cores which support synchronous halt, only. ARMPostInit: Like ARMV1 but the CTI connection differs from the ARM recommendation. OMAP3: This mode is not yet used. TMS570: Used for a certain CTI connection used on a TMS570 derivative. CortexV1: The CTI will be configured for synchronous start and stop via CTI. It assumes the connection of DBGRQ, DBGACK, DBGRESTART signals to CTI are done as recommended by ARM. The CTIBASE must be notified. “CortexV1” is the default value if a Cortex-A/R core is selected and the CTIBASE is notified. QV1: This mode is not yet used. ARMV8V1: Channel 0 and 1 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only. ARMV8V2: Channel 2 and 3 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only. ARMV8V3: Channel 0, 1 and 2 of the CTM are used to distribute start/stop events. Implemented on request. ARMv8 only.
ETR.CATUBase *<address>*	Base address of the CoreSight Address Translation Unit (CATU).
FUNNEL.Name *<string>*	It is possible that different funnels have the same address for their control register block. This assumes they are on different buses and for different cores. In this case it is needed to give the funnel different names to differentiate them.
FUNNEL.PROGrammable [**ON** | **OFF**]	Default is ON. If set to ON the peripheral is controlled by TRACE32 in order to route ATB trace data through the ATB bus network. If PROGrammable is configured to value OFF then TRACE32 will not access the FUNNEL registers and the base address doesn't need to be configured. This can be useful for FUNNELs that don't have registers or when those registers are read-only. TRACE32 need still be aware of the connected ATB trace sources and sink in order to know the ATB topology. To build a complete topology across multiple instances of PowerView the property Name should be set at all instances to a chip wide unique identifier.

See the description of the commands above. Please note that there is a common description for ... .ATBSource, ...Base, , ...RESet, ...TraceID.

COREDEBUG.Base *<address>*
COREDEBUG.RESet
Core Debug Register - ARM debug register, e.g. on Cortex-A/R
Some cores do not have a fix location for their debug register used to control the core. In this case it is essential to specify its location before you can connect by e.g. SYStem.Up.

CTI.Base *<address>*
CTI.Config [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
CTI.RESet
Cross Trigger Interface (CTI) - ARM CoreSight module
If notified the debugger uses it to synchronously halt (and sometimes also to start) multiple cores.

ETB.ATBSource *<source>*
ETB.Base *<address>*
ETB.RESet
ETB.Size *<size>*
Embedded Trace Buffer (ETB) - ARM CoreSight module
Enables trace to be stored in a dedicated SRAM. The trace data will be read out through the debug port after the capturing has finished.

ETF.ATBSource *<source>*
ETF.Base *<address>*
ETF.RESet
Embedded Trace FIFO (ETF) - ARM CoreSight module
On-chip trace buffer used to lower the trace bandwidth peaks.

ETR.ATBSource *<source>*
ETR.Base *<address>*
ETR.CATUBase *<address>*
ETR.RESet
Embedded Trace Router (ETR) - ARM CoreSight module
Enables trace to be routed over an AXI bus to system memory or to any other AXI slave.

FUNNEL.ATBSource *<sourcelist>*
FUNNEL.Base *<address>*
FUNNEL.Name *<string>*
FUNNEL.PROGrammable [**ON** | **OFF**]
FUNNEL.RESet
CoreSight Trace Funnel (CSTF) - ARM CoreSight module
Combines multiple trace sources onto a single trace bus (ATB = AMBA Trace Bus)

REP.ATBSource *<sourcelist>*
REP.Base *<address>*
REP.Name *<string>*
REP.RESet
CoreSight Replicator - ARM CoreSight module

This command group is used to configure ARM Coresight Replicators with programming interface. After the Replicator(s) have been defined by the base address and optional names the ATB sources REPlicatorA and REPlicatorB can be used from other ATB sinks to connect to output A or B to the Replicator.

SYStem.CPU Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**APEX321** | **APEX642-APU0** | **APEX642-APU1** | …

Selects the processor type. For the APEX642, consisting of 2 APUs, it is possible to select the single APUs separately. Besides, there are selections available that target-specific EVBs containing an APEX CPU.

SYStem.JtagClock

Format:	**SYStem.JtagClock** [*<frequency>* | **RTCK**]
<frequency>:	**10000.** … **40000000.** **1250000.** | **2500000.** | **5000000.** | **10000000.** (on obsolete ICD hardware)

Default frequency: 10 MHz.

Selects the JTAG port frequency (TCK) used by the debugger to communicate with the processor. The frequency affects e.g. the download speed. It could be required to reduce the JTAG frequency if there are buffers, additional loads or high capacities on the JTAG lines or if VTREF is very low. A very high frequency will not work on all systems and will result in an erroneous data transfer.

<frequency>	The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the **SYStem.state** window. Besides a decimal number like “100000.” short forms like “10kHz” or “15MHz” can also be used. The short forms imply a decimal value, although no “.” is used.
ARTCK	The JTAG interface of the APEX does not offer ARTCK (**A**ccelerated **R**eturned **TCK**). However, in multicore applications with ARM, RTCK can be used to control the JTAG clock.
CRTCK	The JTAG interface of the APEX does not offer CRTCK. However, in multicore applications with ARM, CRTCK can be used to control the JTAG clock.
CTCK	The JTAG interface of the APEX does not offer CTCK. However, in multicore applications with ARM, CTCK can be used to control the JTAG clock.
RTCK	The JTAG interface of the APEX does not offer RTCK (**R**eturned **TCK**). However, in multicore applications with ARM, RTCK can be used to control the JTAG clock.

SYStem.LOCK

Format:	**SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the JTAG port will be performed by the debugger. While locked the JTAG connector of the debugger is tristated. The intention of the **SYStem.LOCK** command is, for example, to give JTAG access to another tool. The process can also be automated, see **SYStem.CONFIG TriState**.

It must be ensured that the state of the APEX core JTAG state machine remains unchanged while the system is locked. To ensure correct hand-over, the options **SYStem.CONFIG TAPState** and **SYStem.CONFIG TCKLevel** must be set properly. They define the TAP state and TCK level which is selected when the debugger switches to tristate mode.

SYStem.MemAccess

Format:	**SYStem.MemAccess** *<mode>*
<mode>:	**DAP** **Enable** **StopAndGo** **Denied**

Default: Denied.

If **SYStem.MemAccess** is not **Denied**, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program.

DAP	A run-time memory access is done via a Memory Access Port (MEM-AP) of the Debug Access Port (DAP). This is only possible if a DAP is available on the chip and if the memory bus is connected to it (ARM Cortex, CoreSight). The debugger uses the AXI MEM-AP specified by SYStem.CONFIG AXIACCESSPORT if available, the MEM-AP (typically AHB) specified by SYStem.CONFIG MEMORYACCESSPORT otherwise.
Enable **CPU** (deprecated)	A run-time memory access is made without CPU intervention while the program is running. This is only possible on the instruction set simulator.
Denied	No memory access is possible while the CPU is executing the program.
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.

If specific windows that display memory or variables should be updated while the program is running, select the memory class E: or the format option **%E**.

```
Data.dump ED:0x100

Var.View %E first
```

SYStem.Mode — Establish the communication with the target

Format:	**SYStem.Mode** *<mode>*
	SYStem.Attach (alias for SYStem.Mode Attach)
	SYStem.Down (alias for SYStem.Mode Down)
	SYStem.Up (alias for SYStem.Mode Up)
<mode>:	**Down**
	NoDebug
	Go
	Attach
	Up

Down	Disables the debugger (default). The state of the CPU remains unchanged. The JTAG port is tristated.
NoDebug	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
Go	Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or external trigger.
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.
StandBy	Not available for APEX.
Up	Resets the target, sets the CPU to debug mode and stops the CPU. After the execution of this command the CPU is stopped and all register are set to the default level.

SYStem.Option.AHBHPROT — Select AHB-AP HPROT bits

Format:	**SYStem.Option.AHBHPROT** *<value>*

Default: 0

This option selects the value used for the HPROT bits in the Control Status Word (CSW) of an AHB Access Port of a DAP, when using the AHB: memory class.

SYStem.Option.AXIACEEnable ACE enable flag of the AXI-AP

Format:	**SYStem.Option.AXIACEEnable** [**ON** | **OFF**]

Default: OFF

Enable ACE transactions on the DAP AXI-AP, including barriers.

SYStem.Option.AXICACHEFLAGS Select AXI-AP CACHE bits

Format:	**SYStem.Option.AXICACHEFLAGS** *<value>*

Default: 0

This option selects the value used for the CACHE bits in the Control Status Word (CSW) of an AXI Access Port of a DAP, when using the AXI: memory class.

SYStem.Option.AXIHPROT Select AXI-AP HPROT bits

Format:	**SYStem.Option.AXIHPROT** *<value>*

Default: 0

This option selects the value used for the HPROT bits in the Control Status Word (CSW) of an AXI Access Port of a DAP, when using the AXI: memory class.

SYStem.Option.DAPDBGPWRUPREQ   Force debug power in DAP

Format:	**SYStem.Option.DAPDBGPWRUPREQ** [**ON** | **AlwaysON** | **OFF**]

Default: ON.

This option controls the DBGPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port (DAP) before and after the debug session. Debug power will always be requested by the debugger on a debug session start because debug power is mandatory for debugger operation.

ON	Debug power is requested by the debugger on a debug session start, and the control bit is set to 1. The debug power is released at the end of the debug session, and the control bit is set to 0.
AlwaysON	Debug power is requested by the debugger on a debug session start, and the control bit is set to 1. The debug power is **not** released at the end of the debug session, and the control bit is set to 0.
OFF	Only for test purposes: Debug power is **not** requested and **not** checked by the debugger. The control bit is set to 0.

Use case:

Imagine an AMP session consisting of at least of two TRACE32 PowerView GUIs, where one GUI is the master and all other GUIs are slaves. If the master GUI is closed first, it releases the debug power. As a result, a debug port fail error may be displayed in the remaining slave GUIs because they cannot access the debug interface anymore.

To keep the debug interface active, it is recommended that **SYStem.Option.DAPDBGPWRUPREQ** is set to **AlwaysON**.

SYStem.Option.DAPNOIRCHECK   No DAP instruction register check

Format:	**SYStem.Option.DAPNOIRCHECK** [**ON** | **OFF**]

Default: OFF.

Bug fix for derivatives which do not return the correct pattern on a DAP (Arm CoreSight Debug Access Port) instruction register (IR) scan. When activated, the returned pattern will not be checked by the debugger.

SYStem.Option.DAPREMAP Rearrange DAP memory map

Format:	**SYStem.Option.DAPREMAP** {*<address_range> <address>*}

The Debug Access Port (DAP) can be used for memory access during runtime. If the mapping on the DAP is different than the processor view, then this re-mapping command can be used

NOTE:	Up to 16 *<address_range>*/*<address>* pairs are possible. Each pair has to contain an address range followed by a single address.

SYStem.Option.DAPSYSPWRUPREQ Force system power in DAP

Format:	**SYStem.Option.DAPSYSPWRUPREQ** [**AlwaysON** | **ON** | **OFF**]

Default: ON.

This option controls the SYSPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port (DAP) during and after the debug session

AlwaysON	System power is requested by the debugger on a debug session start, and the control bit is set to 1. The system power is **not** released at the end of the debug session, and the control bit remains at 1.
ON	System power is requested by the debugger on a debug session start, and the control bit is set to 1. The system power is released at the end of the debug session, and the control bit is set to 0.
OFF	System power is **not** requested by the debugger on a debug session start, and the control bit is set to 0.

Format:	**SYStem.Option.DEBUGPORTOptions** *<option>*
<option>:	**SWITCHTOSWD.**[**TryAll** | **None** | **JtagToSwd** | **LuminaryJtagToSwd** | **Dor-mantToSwd** | **JtagToDormantToSwd**] **SWDTRSTKEEP.**[**DEFault** | **LOW** | **HIGH**]

Default: SWITCHTOSWD.TryAll, SWDTRSTKEEP.DEFault.

See Arm CoreSight manuals to understand the used terms and abbreviations and what is going on here.

SWITCHTOSWD tells the debugger what to do in order to switch the debug port to serial wire mode:

TryAll	Try all switching methods in the order they are listed below. This is the default. Normally it does not hurt to try improper switching sequences. Therefore this succeeds in most cases.
None	There is no switching sequence required. The SW-DP is ready after power-up. The debug port of this device can only be used as SW-DP.
JtagToSwd	Switching procedure as it is required on SWJ-DP without a dormant state. The device is in JTAG mode after power-up.
LuminaryJtagToSwd	Switching procedure as it is required on devices from LuminaryMicro. The device is in JTAG mode after power-up.
DormantToSwd	Switching procedure which is required if the device starts up in dormant state. The device has a dormant state but does not support JTAG.
JtagToDormantToSwd	Switching procedure as it is required on SWJ-DP with a dormant state. The device is in JTAG mode after power-up.

SWDTRSTKEEP tells the debugger what to do with the nTRST signal on the debug connector during serial wire operation. This signal is not required for the serial wire mode but might have effect on some target boards, so that it needs to have a certain signal level.

DEFault	Use nTRST the same way as in JTAG mode which is typically a low-pulse on debugger start-up followed by keeping it high.
LOW	Keep nTRST low during serial wire operation.
HIGH	Keep nTRST high during serial wire operation

SYStem.Option.EnReset Allow the debugger to drive nRESET/nSRST

Format:	**SYStem.Option.EnReset** [**ON** | **OFF**]

Default: ON.

If this option is disabled in case of a DAP configuration alongside an ARM processor, the debugger will never drive the nSRST line on the JTAG connector. This is necessary if nSRST is no open collector or tristate signal.

From the view of the ARM core it is not necessary that nRESET / nSRST becomes active at the start of a debug session (**SYStem.Up**), but there may be other logic on the target which requires a reset.

SYStem.Option.IgnoreAttributes Obey ELF attributes for breakpoints

Format:	**SYStem.Option.IgnoreAttributes** [**ON** | **OFF**]

Default: ON.

TRACE32 obeys per default ELF file attributes for onchip and software breakpoints. THis behavior can be changed by setting this option to OFF.

SYStem.Option.IMASKASM Disable interrupts while single stepping

[SYStem.state window > IMASKASM]

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 interrupt routine is not executed during single-step operations. After a single step, the interrupt mask bits are restored to the value before the step.

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

[SYStem.state window > IMASKHLL]

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 interrupt routine is not executed during single-step operations. After a single step, the interrupt mask bits are restored to the value before the step.

SYStem.Option.MEMoryMODEL — Select view of memory classes

Format:	**SYStem.Option.MEMoryMODEL** [**SMEM** | **SMEM+DMEM** | **IMEM+DMEM**]

Default: SMEM

With this option it is possible to choose at which address the memory content is shown in a **Data.dump** window. The alternatives are:

SMEM	Data and program memory are displayed in the same address space. Program memory starts after data memory.
SMEM+DMEM	The P: memory class shows SMEM addresses without an offset. The D: memory class shows the memory relatively to the DM_START register.
IMEM+DMEM	Program and data memory are displayed in different address spaces, both starting at 0.

SYStem.Option.MEMORYHPROT — Select memory-AP HPROT bits

Format:	**SYStem.Option.MEMORYHPROT** *<value>*

Default: 0

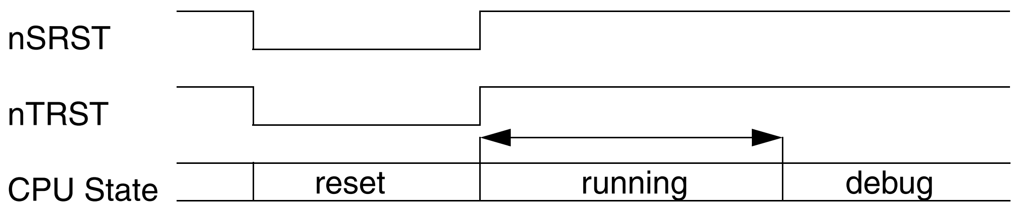
This option selects the value used for the HPROT bits in the Control Status Word (CSW) of an Memory Access Port of a DAP, when using the E: memory class.

SYStem.Option.ResBreak — Halt the core after reset

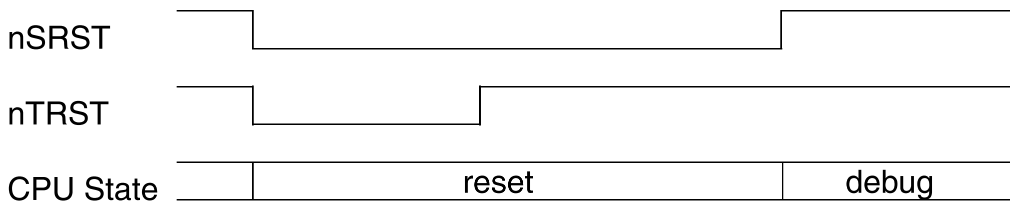
Format:	**SYStem.Option.ResBreak** [**ON** | **OFF**]

Default: ON.

This option has to be disabled if the nTRST line is connected to the nRESET / nSRST line on the target. In this case the CPU executes some cycles while the **SYStem.Up** command is executed. The reason for this behavior is the fact that it is necessary to halt the core (enter debug mode) by a JTAG sequence. This sequence is only possible while nTRST is inactive. In the following figure the marked time between the deassertion of reset and the entry into debug mode is the time of this JTAG sequence plus a time delay selectable by **SYStem.Option.WaitReset** (default = 3 msec).



If nTRST is available and not connected to nRESET/nSRST it is possible to force the CPU directly after reset (without cycles) into debug mode. This is also possible by pulling nTRST fixed to VCC (inactive), but then there is the problem that it is normally not ensured that the JTAG port is reset in normal operation. If the ResBreak option is enabled the debugger first deasserts nTRST, then it executes a JTAG sequence to set the DBGRQ bit in the ICE breaker control register and then it deasserts nRESET/nSRST.



SYStem.Option.TRST Allow debugger to drive TRST

Format:	**SYStem.Option.TRST** [**ON** | **OFF**]

Default: ON.

If this option is disabled the nTRST line is never asserted by the debugger (permanent high). Instead five consecutive TCK pulses with TMS high are asserted to reset the TAP controller which have the same effect.

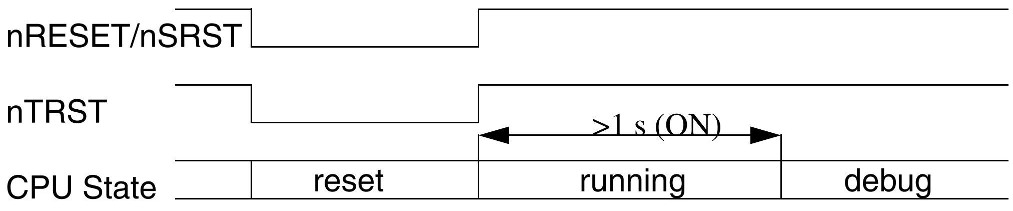
SYStem.Option.WaitReset Wait with JTAG activities after deasserting reset

Format:	**SYStem.Option.WaitReset** [**ON** | **OFF** | *<time>*]

Default: OFF.

If **SYStem.Option.ResBreak** is disabled the debugger waits after the deassertion of nRESET/nSRST and nTRST before the first JTAG activity starts (see picture below). During this time the ARM core may execute some code, e.g. to enable the JTAG port. If **SYStem.Option.ResBreak** is enabled the debugger waits after the deassertion of nTRST before the first JTAG activity starts while nSRST remains active.

ON	1 sec delay
OFF	3 msec delay
<time>	Selectable time delay, min 50 usec, max 30 sec, use 'us', 'ms, 's' as unit.



SYStem.RESetOut — Assert nRESET/nSRST on JTAG connector

[SYStem.state window > RESetOut]

Format:	**SYStem.RESetOut**

If possible (nRESET/nSRST is open collector), this command asserts the nRESET/nSRST line on the JTAG connector. While the CPU is in debug mode, this function will be ignored. Use the **SYStem.Up** command if you want to reset the CPU in debug mode.

SYStem.state — Display SYStem.state window

Format:	**SYStem.state**

Displays the **SYStem.state** window for system settings that configure debugger and target behavior.

APEX Specific TrOnchip Commands

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.RESet — Set on-chip trigger to default state

Format: **TrOnchip.RESet**

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

TrOnchip.VarCONVert — Adjust complex breakpoint in on-chip resource

Format: **TrOnchip.VarCONVert** [**ON** | **OFF**]

Not used by the APEX debugger.

Target Adaption

Interface Standards JTAG, Serial Wire Debug, cJTAG

The Debug Cable supports JTAG (IEEE 1149.1), Serial Wire Debug (CoreSight ARM), and Compact JTAG (IEEE 1149.7, cJTAG) interface standards. The different modes are supported by the same connector. Only some signals get a different function. The mode can be selected by debugger commands. This assumes of course that your target supports this interface standard.

Serial Wire Debug is activated/deactivated by **SYStem.CONFIG DEBUGPORTTYPE [SWD | JTAG]**. In a multidrop configuration you need to specify the address of your debug client by **SYStem.CONFIG SWDPTARGETSEL.**

cJTAG is activated/deactivated by **SYStem.CONFIG DEBUGPORTTYPE [CJTAG | JTAG]**. Your system might need bug fixes which can be activated by **SYStem.CONFIG CJTAGFLAGS**.

Serial Wire Debug (SWD) and Compact JTAG (cJTAG) require a Debug Cable version V4 or newer (delivered since 2008) and one of the newer base modules (Power Debug Pro, Power Debug Interface USB 2.0/USB 3.0, Power Debug Ethernet, PowerTrace or Power Debug II).

Pinout

Adaption for ARM Debug Cable: See **https://www.lauterbach.com/adarmdbg.html**.

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	VSUPPLY (not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS|TMSC|SWDIO	7	8	GND
TCK|TCKC|SWCLK	9	10	GND
RTCK	11	12	GND
TDO|-|SWO	13	14	GND
RESET-	15	16	GND
DBGRQ	17	18	GND
DBGACK	19	20	GND

For details on logical functionality, physical connector, alternative connectors, electrical characteristics, timing behavior and printing circuit design hints, refer to **“ARM JTAG Interface Specifications”** (app_arm_jtag.pdf).