LAUTERBACH
DEVELOPMENT TOOLS

CEVA-Oak/Teak/TeakLite Debugger and Trace

MANUAL

CEVA-Oak/Teak/TeakLite Debugger and Trace

CEVA-Oak/Teak/TeakLite Debugger and Trace

Version 06-Jun-2024

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

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

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. Specify the CPU specific settings.

   ```
   SYStem.CPU <cpu_type>
   SYStem.Option.BASE <base>
   SYStem.Option.MonBase <base>
   ```

   The default values of all other options are set in such a way that it should be possible to work without modification. Please consider that this is probably not the best configuration for your target.

3. Set the JTAG frequency

   ```
   SYStem.JtagClock <frequency>
   ```

   The default value is 1.0 MHz.

4. Inform the debugger about read-only address ranges (ROM, FLASH).

   ```
   MAP.BOnchip <range>
   ```

   The B(reak)Onchip information is necessary to decide where on-chip breakpoints must be used. On-chip breakpoints are necessary to set program breakpoints to FLASH/ROM. The sections of FLASH and ROM depend on the specific CPU and its chip selects.

5. Enter debug mode.

   ```
   SYStem.Up
   ```

   This command resets the CPU and enters debug mode. After this command is executed, it is possible to access memory and registers.

6. Load the program.

```
Data.LOAD.COFF program.a     ; COFF specifies the format, program.a
                             ; is the file name)
```

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 for the TeakLiteDev-C is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*. Other sequences can be found in the ~~/demo/... directory.

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

WinClear                      ; Clear all windows

SYS.CPU TeakLiteDev-C         ; Select CPU

SYS.JC 10MHz                  ; Choose JTAG frequency

SYStem.Up                     ; Reset the target and enter debug mode

Data.LOAD.COFF demo.a         ; Load the application with option large
                              ; memory model and verify the process

Register.Set PC start         ; Set program counter

List.Mix                      ; Open source code window          *)

Go main                       ; Run and break at main()

Register.view /SpotLight      ; Open register window             *)

Var.Local                     ; Open window with local variables *)
```

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

Troubleshooting

No information available

FAQ

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

CPU Specific Implementations

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 supported on-chip breakpoints:

- **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.

Family	Onchip Breakpoints	Program Breakpoints	Read/Write Breakpoints	Data Value Breakpoints
OAK **TeakLite** **TeakLite II** **Teak**	3 instruction 1 read/write	3 single address	1 single address or range as bit mask	1
TeakLite III	2 instruction 1 read/write	2 single address	2 single address or 1 range	1

SYStem.CONFIG.state — Display target configuration

Format:	**SYStem.CONFIG.state** [**/**<*tab*>]
<*tab*>:	**DebugPort** | **Jtag** | **AccessPorts** | **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**.
AccessPorts	This tab informs the debugger about an Arm CoreSight Access Port (AP) and about how to control the AP to access chip-internal memory busses (AHB, APB, AXI) or chip-internal JTAG interfaces. For a descriptions of a corresponding commands, refer to **AP**.

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**.

SYStem.CONFIG Configure debugger according to target topology

Format:	**SYStem.CONFIG** *<parameter>* **SYStem.MultiCore** *<parameter>* (deprecated)
<parameter>: **(DebugPort)**	**CJTAGFLAGS** *<flags>* **CJTAGTCA** *<value>* **CORE** *<core> <chip>* **DEBUGPORT** [**DebugCable0** | **DebugCableA** | **DebugCableB**] **DEBUGPORTTYPE** [**JTAG** | **SWD** | **CJTAG**] **Slave** [**ON** | **OFF**] **SWDPIDLEHIGH** [**ON** | **OFF**] **SWDPTargetSel** *<value>* **DAP2SWDPTargetSel** *<value>* **TriState** [**ON** | **OFF**]
<parameter>: **(JTAG)**	**DRPOST** *<bits>* **DRPRE** *<bits>* **IRPOST***<bits>* **IRPRE** *<bits>* **DAPDRPOST** *<bits>* **DAPDRPRE** *<bits>* **DAPIRPOST** *<bits>* **DAPIRPRE** *<bits>* **DAP2DRPOST** *<bits>* **DAP2DRPRE** *<bits>* **DAP2IRPOST** *<bits>* **DAP2IRPRE** *<bits>* **Slave** [**ON** | **OFF**] **TAPState** *<state>* **TCKLevel** *<level>* **TriState** [**ON** | **OFF**]

<parameter>: **(AccessPorts)**

AHBAPn.Base *<address>*
AHBAPn.HPROT [*<value>* | *<name>*]
AHBAPn.Port *<port>*
AHBAPn.RESet
AHBAPn.view
AHBAPn.XtorName *<name>*

APBAPn.Base *<address>*
APBAPn.Port *<port>*
APBAPn.RESet
APBAPn.view
APBAPn.XtorName *<name>*

AXIAPn.ACEEnable [**ON** | **OFF**]
AXIAPn.Base *<address>*
AXIAPn.CacheFlags *<value>*
AXIAPn.HPROT [*<value>* | *<name>*]
AXIAPn.Port *<port>*
AXIAPn.RESet
AXIAPn.view
AXIAPn.XtorName *<name>*

DEBUGAPn.Port *<port>*
DEBUGAPn.RESet
DEBUGAPn.view
DEBUGAPn.XtorName *<name>*

JTAGAPn.Base *<address>*
JTAGAPn.Port *<port>*
JTAGAPn.CorePort *<port>*
JTAGAPn.RESet
JTAGAPn.view
JTAGAPn.XtorName *<name>*

MEMORYAPn.HPROT [*<value>* | *<name>*]
MEMORYAPn.Port *<port>*
MEMORYAPn.RESet
MEMORYAPn.view
MEMORYAPn.XtorName *<name>*

<parameter>: **(COmponents)**

COREDEBUG.Base *<address>*
COREDEBUG.RESet
COREDEBUG.view

CTI.Base *<address>*
CTI.Config *<config>*
CTI.RESet
CTI.view

ETB.ATBSource *<source>*
ETB.Base *<address>*

<parameter>: (COmponents cont.)	**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** **ETM.Base** *<address>* **ETM.RESet** **ETM.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**

<parameter>: (COmponents cont.)	**HTM.Base** *<address>*
	HTM.RESet
	HTM.Type [**CoreSight** | **WPT**]
	REP.ATBSource *<source>*
	REP.Base *<address>*
	REP.Name *<string>*
	REP.RESet
	REP.view
	TPIU.ATBSource *<source>*
	TPIU.Base *<address>*
	TPIU.Name *<string>*
	TPIU.RESet
	TPIU.Type [**CoreSight** | **Generic**]
	TPIU.view
<parameter>: **(Deprecated)**	**CTIBASE** *<address>*
	CTICONFIG *<config>*
<parameter>: (Deprecated cont.)	**DEBUGBASE** *<address>*
	ETBBASE *<address>*
	ETBFUNNELBASE *<address>*
	ETMBASE *<address>*
	ETMETBFUNNELPORT *<port>*
	ETMFUNNEL2PORT *<port>*
	ETMFUNNELPORT *<port>*
	ETMTPIUFUNNELPORT *<port>*
	FUNNEL2BASE *<address>*
	FUNNELBASE *<address>*
	HTMBASE *<address>*
	HTMETBFUNNELPORT *<port>*
	HTMFUNNEL2PORT *<port>*
	HTMFUNNELPORT *<port>*
	HTMTPIUFUNNELPORT *<port>*
	TPIUBASE *<address>*
	TPIUFUNNELBASE *<address>*
	view
	AHBACCESSPORT *<port>*
	APBACCESSPORT *<port>*
	AXIACCESSPORT *<port>*
	COREJTAGPORT *<port>*
	DEBUGACCESSPORT *<port>*
	JTAGACCESSPORT *<port>*
	MEMORYACCESSPORT *<port>*

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

Some commands need a certain CPU type selection (**SYStem.CPU** *<type>*) to become active and might additionally depend on further settings.

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,...).

DEBUGPORT [**DebugCable0** | **DebugCableA** | **DebugCableB**]	It specifies which probe cable shall be used e.g. “DebugCableA” or “DebugCableB”. At the moment only the CombiProbe allows to connect more than one probe cable. Default: depends on detection.
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).
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).
DAP2SWDPTargetSel *<value>*	Device address of the second CoreSight DAP (DAP2) in case of a multidrop serial wire debug port (SWD). 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.

<parameters> describing the “JTAG” scan chain and signal behavior

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).

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.

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.

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.

DAP2 (Debug Access Port) TAP in case you need to access a second DAP to reach other memory locations.
-> DAP2DRPOST, DAP2DRPRE, DAP2IRPOST, DAP2IRPRE.

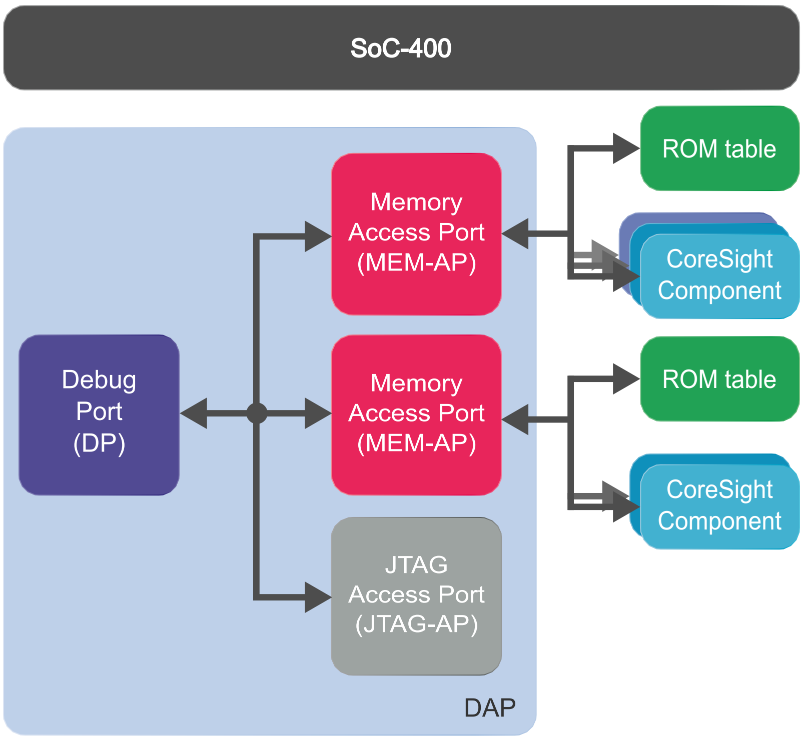
An Access Port (AP) is a CoreSight module from ARM which provides access via its debug link (JTAG, cJTAG, SWD, USB, UDP/TCP-IP, GTL, PCIe...) to:

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).

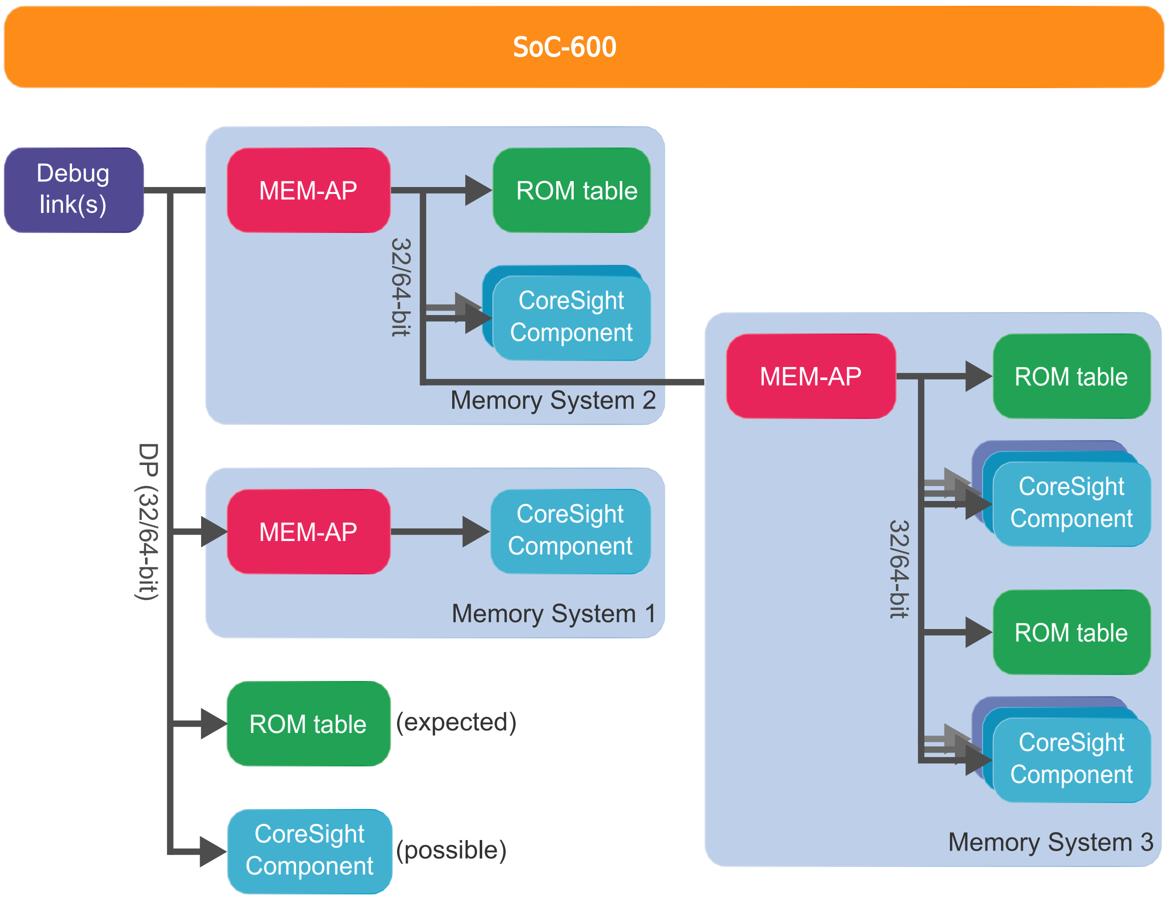
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number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

3. A transactor name for virtual connections to AMBA bus level transactors can be configured by the property **SYStem.CONFIG.*APn.XtorName** *<name>*. A JTAG or SWD transactor must be configured for virtual connections to use the property "Port" or "Base" (with "DP:" access) in case XtorName remains empty.

Example 1: SoC-400



Example 2: SoC-600



AHBAPn.HPROT [*<value>* | *<name>*] **SYStem.Option.AHBHPROT** [*<value>* | *<name>*] (deprecated)	Default: 0. Selects the value used for the HPROT bits in the Control Status Word (CSW) of a CoreSight AHB Access Port, when using the AHB: memory class.
AXIAPn.HPROT [*<value>* | *<name>*] **SYStem.Option.AXIHPROT** [*<value>* | *<name>*] (deprecated)	Default: 0. This option selects the value used for the HPROT bits in the Control Status Word (CSW) of a CoreSight AXI Access Port, when using the AXI: memory class.
MEMORYAPn.HPROT [*<value>* | *<name>*] **SYStem.Option.MEMORYHPROT** [*<value>* | *<name>*] (deprecated)	Default: 0. This option selects the value used for the HPROT bits in the Control Status Word (CSW) of a CoreSight Memory Access Port, when using the E: memory class.

AXIAPn.ACEEnable [**ON** | **OFF**]	Default: OFF. Enables ACE transactions on the AXI-AP, including barriers. This does only work if the debug logic of the target CPU implements coherent accesses. Otherwise this option will be without effect.
AXIAPn.CacheFlags *<value>*	Default: DeviceSYStem (=0x30: Domain=0x3, Cache=0x0). This option configures the value used for the Cache and Domain bits in the Control Status Word (CSW[27:24]->Cache, CSW[14:13]->Domain) of an Access Port, when using the AXI: memory class.

The below offered selection options are all non-bufferable. Alternatively you can enter a <value>, where value[5:4] determines the Domain bits and value[3:0] the Cache bits.

<name>	Description
DeviceSYStem	=0x30: Domain=0x3, Cache=0x0
NonCacheableSYStem	=0x32: Domain=0x3, Cache=0x2
ReadAllocateNonShareable	=0x06: Domain=0x0, Cache=0x6
ReadAllocateInnerShareable	=0x16: Domain=0x1, Cache=0x6
ReadAllocateOuterShareable	=0x26: Domain=0x2, Cache=0x6
WriteAllocateNonShareable	=0x0A: Domain=0x0, Cache=0xA
WriteAllocateInnerShareable	=0x1A: Domain=0x1, Cache=0xA
WriteAllocateOuterShareable	=0x2A: Domain=0x2, Cache=0xA
ReadWriteAllocateNonShareable	=0x0E: Domain=0x0, Cache=0xE
ReadWriteAllocateInnerShareable	=0x1E: Domain=0x1, Cache=0xE
ReadWriteAllocateOuterShareable	=0x2E: Domain=0x2, Cache=0xE

AHBAPn.XtorName *<name>*	AHB bus transactor name that shall be used for “AHBn:” access class.
APBAPn.XtorName *<name>*	APB bus transactor name that shall be used for “APBn:” access class.

AXIAPn.XtorName *<name>*	AXI bus transactor name that shall be used for “AXIn:” access class.
DEBUGAPn.XtorName *<name>*	APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP:” access class.
MEMORYAPn.XtorName *<name>*	AHB bus transactor name identifying the bus where system memory can be accessed even during runtime. Used for “E:” access class while running, assuming “**SYStem.MemAccess DAP**”.
... .RESet	Undo the configuration for this access port. This does not cause a physical reset for the access port on the chip.
... .view	Opens a window showing the current configuration of the access port.

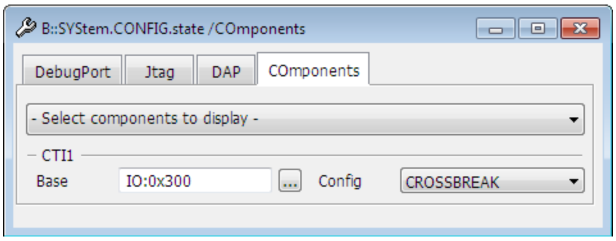
AHBAPn.Port *<port>* **AHBACCESSPORT** *<port>* (deprecated)	Access Port Number (0-255) of a SoC-400 system which shall be used for “AHBn:” access class. Default: *<port>*=0.
APBAPn.Port *<port>* **APBACCESSPORT** *<port>* (deprecated)	Access Port Number (0-255) of a SoC-400 system which shall be used for “APBn:” access class. Default: *<port>*=1.
AXIAPn.Port *<port>* **AXIACCESSPORT** *<port>* (deprecated)	Access Port Number (0-255) of a SoC-400 system which shall be used for “AXIn:” access class. Default: port not available.
DEBUGAPn.Port *<port>* **DEBUGACCESSPORT** *<port>* (deprecated)	AP access port number (0-255) of a SoC-400 system where the debug register can be found (typically on APB). Used for “DAP:” access class. Default: *<port>*=1.
JTAGAPn.CorePort *<port>* **COREJTAGPORT** *<port>* (deprecated)	JTAG-AP port number (0-7) connected to the core which shall be debugged.
JTAGAPn.Port *<port>* **JTAGACCESSPORT** *<port>* (deprecated)	Access port number (0-255) of a SoC-400 system of the JTAG Access Port.
MEMORYAPn.Port *<port>* **MEMORYACCESSPORT** *<port>* (deprecated)	AP access port number (0-255) of a SoC-400 system where system memory can be accessed even during runtime (typically an AHB). Used for “E:” access class while running, assuming “**SYStem.MemAccess DAP**”. Default: *<port>*=0.

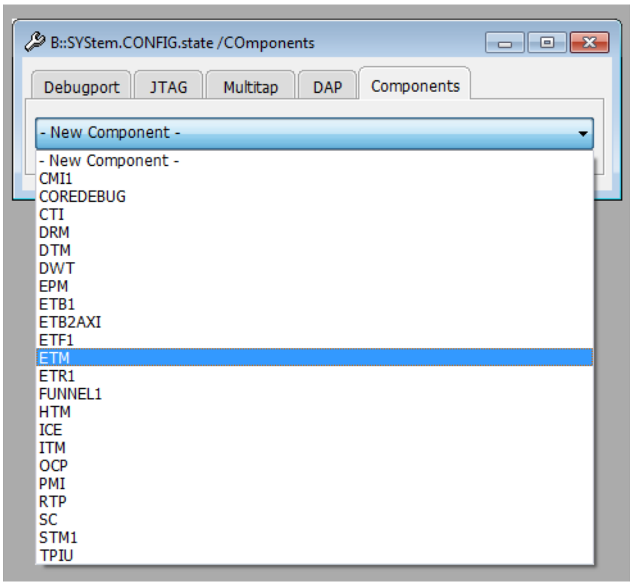
SoC-600 Specific Commands

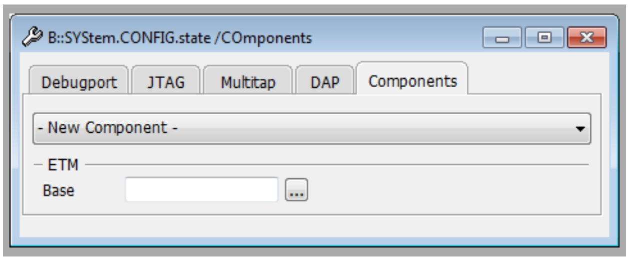
AHBAPn.Base *<address>*	This command informs the debugger about the start address of the register block of the “AHBAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus connected to the access port. **Example**: SYStem.CONFIG.AHBAP1.Base DP:0x80002000 Meaning: The control register block of the AHB access ports starts at address 0x80002000.
APBAPn.Base *<address>*	This command informs the debugger about the start address of the register block of the “APBAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus connected to the access port. **Example**: SYStem.CONFIG.APBAP1.Base DP:0x80003000 Meaning: The control register block of the APB access ports starts at address 0x80003000.
AXIAPn.Base *<address>*	This command informs the debugger about the start address of the register block of the “AXIAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus connected to the access port. **Example**: SYStem.CONFIG.AXIAP1.Base DP:0x80004000 Meaning: The control register block of the AXI access ports starts at address 0x80004000.
JTAGAPn.Base *<address>*	This command informs the debugger about the start address of the register block of the “JTAGAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus connected to the access port. **Example**: SYStem.CONFIG.JTAGAP1.Base DP:0x80005000 Meaning: The control register block of the JTAG access ports starts at address 0x80005000.

<parameters> describing debug and trace "Components"

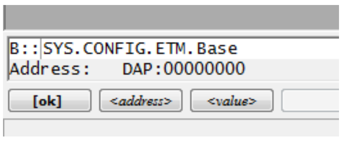
On the **Components** tab in the **SYStem.CONFIG.state** window, you can comfortably add the debug and trace components your chip includes and which you intend to use with the debugger's help.


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.



```
B::SYS.CONFIG.ETM.Base
Address:    DAP:00000000
```

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. **Example**: Four cores with ETM modules. SYStem.CONFIG ETM.Base 0x1000 0x2000 0x3000 0x4000 SYStem.CONFIG FUNNEL1.ATBSource ETM.0 0 ETM.1 1 ETM.2 2 ETM.3 3 "...2" of "ETM.2" indicates it is the third ETM module which has the base address 0x3000. The indices of a list are 0, 1, 2, 3,... If the numbering is accelerating, starting from 0, without gaps, like the example above then you can shorten it to SYStem.CONFIG FUNNEL1.ATBSource ETM **Example**: Four cores, each having an ETM module and an ETB module. SYStem.CONFIG ETM.Base 0x1000 0x2000 0x3000 0x4000 SYStem.CONFIG ETB.Base 0x5000 0x6000 0x7000 0x8000 SYStem.CONFIG ETB.ATBSource ETM.2 2 The third "ETM.2" module is connected to the third ETB. The last "2" in the command above is the index for the ETB. It is not a port number which exists only for FUNNELs. 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 ETMBASE APB:0x8011c000

Meaning: The control register block of the Embedded Trace Macrocell (ETM) starts at address 0x8011c000 and is accessible via APB bus.

In an SMP (Symmetric MultiProcessing) debug session you can enter for the components COREBEBUG, CTI, ETB, ETF, ETM, ETR a list of base addresses to specify one component per core.

Example assuming four cores: SYStem.CONFIG COREDEBUG.Base 0x80001000 0x80003000 0x80005000 0x80007000

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 an ETB 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** [**NotAvailable** | **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**.

… **.TraceID** *<id>*	Identifies from which component the trace packet is coming from. Components which produce trace information (trace sources) for a common trace stream have a selectable ".TraceID *<id>*". If you miss this SYStem.CONFIG command for a certain trace source (e.g. ETM) then there is a dedicated command group for this component where you can select the ID (ETM.TraceID *<id>*). The default setting is typically fine because the debugger uses different default trace IDs for different components. 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.
HTM.Type [**CoreSight** | **WPT**]	Selects the type of the AMBA AHB Trace Macrocell (HTM). CoreSight is the type as described in the ARM CoreSight manuals. WPT is a NXP proprietary trace module.
TPIU.Type [**CoreSight** | **Generic**]	Selects the type of the Trace Port Interface Unit (TPIU). CoreSight: Default. CoreSight TPIU. TPIU control register located at TPIU.Base *<address>* will be handled by the debugger. Generic: Proprietary TPIU. TPIU control register will not be handled by the debugger.

Components and Available Commands

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
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.

ETM.Base *<address>*
ETM.RESet
Embedded Trace Macrocell (ETM) - ARM CoreSight module
Program Trace Macrocell (PTM) - ARM CoreSight module
Trace source providing information about program flow and data accesses of a core.
The ETM commands will be used even for PTM.

ETR.ATBSource *<source>*
ETR.CATUBase *<address>*
ETR.Base *<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.

HTM.Base *<address>*
HTM.RESet
HTM.Type [**CoreSight** | **WPT**]
AMBA AHB Trace Macrocell (HTM) - ARM CoreSight module
Trace source delivering trace data of access to an AHB bus.

TPIU.ATBSource *<source>*
TPIU.Base *<address>*
TPIU.RESet
TPIU.Type [**CoreSight** | **Generic**]
Trace Port Interface Unit (TPIU) - ARM CoreSight module
Trace sink sending the trace off-chip on a parallel trace port (chip pins).

<parameters> which are "Deprecated"

In the last years the chips and its debug and trace architecture became much more complex. Especially the CoreSight trace components and their interconnection on a common trace bus required a reform of our commands. The new commands can deal even with complex structures.

… **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 ETMBASE APB:0x8011c000 Meaning: The control register block of the Embedded Trace Macrocell (ETM) starts at address 0x8011c000 and is accessible via APB bus. In an SMP (Symmetric MultiProcessing) debug session you can enter for the components CORE, CTI, ETB, ETF, ETM, ETR a list of base addresses to specify one component per core. For a list of possible components including a short description see **Components and Available Commands**.
… **PORT** *<port>*	Informs the debugger about which trace source is connected to which input port of which funnel. A CoreSight trace funnel provides 8 input ports (port 0-7) to combine the data of various trace sources to a common trace stream. For a list of possible components including a short description see **Components and Available Commands**.

CTICONFIG *<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.
view	Opens a window showing most of the SYStem.CONFIG settings and allows to modify them.

Deprecated and New Commands

In the following you find the list of deprecated commands which can still be used for compatibility reasons and the corresponding new command.

SYStem.CONFIG *<parameter>*

<parameter>: (Deprecated)	*<parameter>*: (New)
CTIBASE *<address>*	**CTI.Base** *<address>*
CTICONFIG *<type>*	**CTI.Config** *<type>*
DEBUGBASE *<address>*	**COREDEBUG.Base** *<address>*
ETBBASE *<address>*	**ETB1.Base** *<address>*
ETBFUNNELBASE *<address>*	**FUNNEL4.Base** *<address>*
ETMBASE *<address>*	**ETM.Base** *<address>*
ETMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource ETM** *<port>* (1)
ETMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource ETM** *<port>* (1)
ETMFUNNELPORT *<port>*	**FUNNEL1.ATBSource ETM** *<port>* (1)
ETMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource ETM** *<port>* (1)
FUNNEL2BASE *<address>*	**FUNNEL2.Base** *<address>*

FUNNELBASE *<address>*	**FUNNEL1.Base** *<address>*
HTMBASE *<address>*	**HTM.Base** *<address>*
HTMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource HTM** *<port>* (1)
HTMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource HTM** *<port>* (1)
HTMFUNNELPORT *<port>*	**FUNNEL1.ATBSource HTM** *<port>* (1)
HTMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource HTM** *<port>* (1)
TPIUBASE *<address>*	**TPIU.Base** *<address>*
TPIUFUNNELBASE *<address>*	**FUNNEL3.Base** *<address>*
view	**state**

(1) Further "*<component>*.ATBSource *<source>*" commands might be needed to describe the full trace data path from trace source to trace sink.

SYStem.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**OAK** | **PMB8870P** | **PMB8870S** (OAK cores) **TeakLiteDev-A** | **TeakLiteDev-B** | **TeakLiteDev-C** | **PMB8875** | **88i6523** (TeakLite cores) **TEAK-REVA** | **TEAK-RTL2_0** | **TEAK_REVB** | **XPERTTEAK** (Teak cores)

Selects the processor type. If your ASIC is not listed, select the type of the integrated core.

SYStem.JtagClock — Define JTAG clock

Format:	**SYStem.JtagClock** *<frequency>* **SYStem.BdmClock** *<frequency>* (deprecated)

Default: 1 MHz

Selects the frequency for the debug interface.

SYStem.LOCK Lock and tristate the debug port

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

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool.

SYStem.MemAccess Select run-time memory access method

Format:	**SYStem.MemAccess Enable** | **StopAndGo** | **Denied** | **DAP** **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.
DAP	Memory access is done via CoreSight Debug Access Port (DAP).

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** **Up** **Attach** **Go** **NoDebug**

Down	Disables the debugger (default). The state of the CPU remains unchanged. The JTAG port is tristated.
NoDebug	The debugger gets tristated. The state of the CPU remains unchanged. Debug mode is not active. In this mode the target behaves as if the debugger is not connected.
Up	Resets the target, sets the CPU to debug mode and stops the CPU.
Attach	No reset happens, the mode of the core (running or halted) does not change. The debug port will be initialized. After this command, the user program can be stopped e.g. with the **Break** command.
Go	Resets the target, sets the CPU to debug mode and starts the program execution. After this command, the user program can be stopped e.g. with the **Break** command.
StandBy	Not available for CEVA-Oak/Teak/TeakLite.

SYStem.Option.AHBHPROT Select AHB-AP HPROT bits

Format:	**SYStem.Option.AHBHPROT** *<value>* (deprecated) **Use SYStem.CONFIG.AHBAPn.HPROT instead.**

Default: 0

Selects the value used for the HPROT bits in the Control Status Word (CSW) of a CoreSight AHB Access Port, when using the AHB: memory class.

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

Format: **SYStem.Option.AXIACEEnable** [**ON** | **OFF**] (deprecated)
Use SYStem.CONFIG.AXIAPn.ACEEnable instead.

Default: OFF.

Enables ACE transactions on the DAP AXI-AP, including barriers. This does only work if the debug logic of the target CPU implements coherent AXI accesses. Otherwise this option will be without effect.

SYStem.Option.AXICACHEFLAGS Configure AXI-AP cache bits

Format: **SYStem.Option.AXICACHEFLAGS** *<value>* (deprecated)
Use SYStem.CONFIG.AXIAPn.CacheFlags instead.

Default: DeviceSYStem (=0x30: Domain=0x3, Cache=0x0).

This option configures the value used for the Cache and Domain bits in the Control Status Word (CSW[27:24]->Cache, CSW[14:13]->Domain) 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>* (deprecated)
Use SYStem.CONFIG.AXIAPn.HPROT instead.

Default: 0

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

SYStem.Option.BackPC Keep core running except for debugger access

Format: **SYStem.Option.BackPC** *<address>*

Default: 0x0 which equals OFF.

If enabled, the core is hold in running state when debugger does not access the core. This option is only needed for multi core debugging of some special cases. *<address>* has to be a RAM address the debugger can use for the idle loop.

SYStem.Option.BASE Setup MAILBOX or MMIO base address

Format: **SYStem.Option.BASE** *<base>*

Default: 0x4000.

The base address specifies the Mailbox address for TeakLite Rev A/B or Teak Rev A cores. For newer revisions like TeakLite Rev C or Teak Rev B cores this is the MMIO base address.

SYStem.Option.BigEndian Enable big endian mode

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

Default: OFF.

Switches the endianness of memory access to Big Endian.

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** | **DormantToSwd** | **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 reset the processor

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

Default: ON.

If this option is disabled, the debugger will never reset the processor when establishing a debug connection. This is only needed for the XGOLD213 ES1.

SYStem.Option.EXTDBGTRAP — Writing debug trap in external memory

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

When this option is set to OFF, the debug trap is not written in external program memory. Debug trap will only be written in internal memory.

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 interrupt routine is not executed during single-step operations. After single step the interrupt mask bits are restored to the value before the step. For 56800E processors IMASKASM ON is necessary for HLL stepping and stepping from software breakpoints.

SYStem.Option.IMASKHLL — Disable interrupts while HLL single stepping

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 single step the interrupt mask bits are restored to the value before the step.

SYStem.Option.MonBase — Set up monitor base address

Format:	**SYStem.Option.MonBase** *<base>*

Default: 0x2100.

The base address specifies the monitor base address where the monitor has to be loaded to. This option is not required for core implementations with ROM monitor.

General Restrictions

Setting the PC	In cases where the program counter consists of the PC register and program page extension bits, the program counter can be set by the register PP.

SYStem.Option.MONITOR — Define user-specific debug monitor

Format:	**SYStem.Option.MONITOR** *<file>*

This option allows to define a user-specific debug monitor. Only needed for very special cases.

SYStem.Option.PALLADIUM — Extend debugger timeout

Format:	**SYStem.Option.PALLADIUM** [**ON** | **OFF**] (deprecated) **Use SYStem.CONFIG.DEBUGTIMESCALE instead.**

Default: OFF.

The debugger uses longer timeouts as might be needed when used on a chip emulation system like the Palladium from Cadence.

This option will only extend some timeouts by a fixed factor. It is recommended to extend all timeouts. This can be done with **SYStem.CONFIG.DEBUGTIMESCALE**.

SYStem.Option.RisingTDO Target outputs TDO on rising edge

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

Default: OFF.

Bug fix for chips which output the TDO on the rising edge instead of on the falling.

SYStem.Option.RomMon Disable download of monitor routine

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

Disables the download of the monitor routine into the RAM of the chip.

Default: OFF

ON	If the monitor routine used for communicating with the debugger is in the ROM of the chip, then set the command to ON.
OFF	If there is no on-chip monitor routine, then set the command to OFF to load the monitor routine into the RAM of the chip.

SYStem.Option.TKLMON Use TeakLite monitor for TeakLite-II

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

Default: OFF.

This option allows to use the TeakLite monitor for TeakLite-II.

ETM Commands

ETM.AGU32 Data trace mode for AGU

Format:	**ETM.AGU32** [**ON** | **OFF**]

Data trace mode for AGU. For details see Ceva ETM specifications.

TeakLite only.

ETM.AGU64 Data trace mode for AGU

Format:	**ETM.AGU64** [**ON** | **OFF**]

Data trace mode for AGU. For details see Ceva ETM specifications.

TeakLite only.

ETM.ISTACK Interrupt stack operation trace

Format:	**ETM.ISTACK** [**ON** | **OFF**]

Traces push/pop from interrupt entry and return.

ETM.RWM Read-write-modify

Format:	**ETM.ISTACK** [**ON** | **OFF**]

If set to ON, trace only write path of read-write-modify sequence.

ETM.WrapperFilter
Global breakpoint enable

Format: **ETM.WrapperFilter** [**ON** | **OFF**]

Disables or enabled all data and program breakpoints at once.

Default: ON.

ETM.WrapperSTALL
Enable/disable wrapper stall

Format: **ETM.WrapperSTALL** [**ON** | **OFF**]

Default: ON.

TrOnchip Commands

The OCEM registers can be used to break on several conditions.

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

Format: **TrOnchip.RESet**

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

TrOnchip.Set — Set on-chip trigger

Format: **TrOnchip.Set**

TrOnchip.Set.BKRE — Trigger on block repeat loop

Format: **TrOnchip.Set.BKRE** [**ON** | **OFF**]

When enabled indicates the breakpoint upon returning to the beginning of the block repeat loop.

TrOnchip.Set.BRE — Trigger on jump

Format: **TrOnchip.Set.BRE** [**ON** | **OFF**]

When enabled, indicates the breakpoint each time the program jumps instead of executing the next sequential address.

TrOnchip.Set.EXTRE Trigger on external register read

Format: **TrOnchip.Set.EXTRE** [**ON** | **OFF**]

When enabled, enables the breakpoint as a result of external register read transaction.

TrOnchip.Set.EXTWE Trigger on external register write

Format: **TrOnchip.Set.EXTWE** [**ON** | **OFF**]

When enabled, enables the breakpoint as a result of external register write transaction.

TrOnchip.Set.ILLE Trigger on illegal instruction access

Format: **TrOnchip.Set.ILLE** [**ON** | **OFF**]

When enabled (default), the on-chip breakpoint for illegal instruction access is activated.

TrOnchip.Set.INTE Trigger on interrupt

Format: **TrOnchip.Set.INTE** [**ON** | **OFF**]

When enabled, indicates the breakpoint upon detection of the interrupt service routine execution.

TrOnchip.Set.TBFE Trigger on trace buffer full

Format: **TrOnchip.Set.TBFE** [**ON** | **OFF**]

When set, indicates the breakpoint as a result of the program flow trace buffer being filled.

TrOnchip.state — Display “Trigger-Onchip” dialog

Format:	**TrOnchip.state**

Control panel to configure the on-chip breakpoint registers.

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

Format:	**TrOnchip.VarCONVert** [**ON** | **OFF**] (deprecated) **Use Break.CONFIG.VarConvert instead**

The on-chip breakpoints can only cover specific ranges. If you want to set a marker or breakpoint to a complex variable, the on-chip break resources of the CPU may be not powerful enough to cover the whole structure. If the option **TrOnchip.VarCONVert** is set to **ON**, the breakpoint will automatically be converted into a single address breakpoint. This is the default setting. Otherwise an error message is generated.

JTAG Connector

This connector is defined by ARM and also used by our OAK/TeakLite/Teak debuggers La-7774 and LA-7789 for easy support of multi-core processors with ARMx and OAK/TeakLite/Teak core. It is planned to make an additional adapter to 9 pin and 10 pin CEVA connectors.

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

Pins 11, 17 and 19 are not used.

This is a standard 20 pin double row connector (pin-to-pin spacing: 0.100 in.).

We strongly recommend to use a connector on your target with housing and having a center polarization (e.g. AMP: 2-827745-0). A connection the other way around indeed causes damage to the output driver of the debugger.

Memory Classes

Memory Class	Description
D	Data memory
P	Program memory