LAUTERBACH
DEVELOPMENT TOOLS

Xtensa Debugger and Trace

Release 09.2023

MANUAL

Xtensa Debugger and Trace

Xtensa Debugger and Trace

Version 10-Oct-2023

History

17-Nov-22	Chapter 'Xtensa Specific Benchmarking Commands' added.
20-Jul-22	For the MMU.SCAN ALL command, CLEAR is now possible as an optional second parameter.
18-Jan-21	Added descriptions of the commands SYStem.Option.DISableHwWatchDOG, SYStem.Option.SnoopAddressPC, and SYStem.Option.WindowVectorBase.
15-Jan-21	SYStem.TIE.ADDtiedll renamed to SYStem.TIE.AddCoreLibrary. SYStem.TIE.LIBpath renamed to SYStem.TIE.ToolLibraryPath.

Introduction

This manual serves as a guideline for debugging Xtensa cores 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.

Demo and Start-up Scripts

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

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

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

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

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

WARNING:	To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF. Recommendation for the software start: 1. Disconnect the Debug Cable from the target while the target power is off. 2. Connect the host system, the TRACE32 hardware and the Debug Cable. 3. Power ON the TRACE32 hardware. 4. Start the TRACE32 software to load the debugger firmware. 5. Connect the Debug Cable to the target. 6. Switch the target power ON. 7. Configure your debugger e.g. via a start-up script. Power down: 1. Switch off the target power. 2. Disconnect the Debug Cable from the target. 3. Close the TRACE32 software. 4. Power OFF the TRACE32 hardware.

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.Endianness [AUTO | Little | Big]

SYStem.Option.SOFTLONG [ON | OFF]
```

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

```
MAP.BOnchip 0x060000000++3FFFF
```

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.

4. Specify ranges where the access width is restricted.

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

If a memory location can only be accessed with a certain bus width you can use **MAP.BUS8** / **MAP.BUS16** / **MAP.BUS32** to force the debugger to use solely the according load or store instructions. This allows for example to have a byte-by-byte dump of a 32-bit wide memory area, where a byte access would cause an exception.

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 <file> /LONG              ;load the compiler output.
                                    ;the option /LONG tells the
                                    ;debugger to use 32 bit accesses
```

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 without EPROM simulator is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** *<file>*.

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

WinCLEAR                            ; Clear all windows

MAP.BOnchip 0x60000000++0xfffff     ; Specify where FLASH/ROM is

MAP.BUS32 0x50000000++0x1ffff       ; Force the debugger to access this ;
                                    ; area 32 bit wide

SYStem.Up                           ; Reset the target and enter debug
                                    ; mode

Data.LOAD.elf xtensa_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                  *)

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

Break.Set 0x60100000 /Program       ; Set software breakpoint to address
                                    ; 0x60100000 (address 0x60100000
                                    ; outside of BOnchip range)

Break.Set 0x60001000 /Program       ; Set on-chip breakpoint
                                    ; to address 0x60001000 (address
                                    ; 0x60001000 is within BOnchip range)
```

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

NOTE:

- Special registers can be viewed in the peripheral file with the command **PER <path>/per_xtensa.per**. To add your specific registers you can do a copy of this file and modify it using the command:
  PER.Program <path>/my_per_xtensa.per

- The **Register.view** window can be resized to view additional registers by pressing on the small field on the right bottom of the window

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 Xtensa 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.

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. The parameter NIBREAK of the Debug Option Architectural Addition defines the number of available instruction breakpoints. On-chip breakpoints are usually needed for instructions in FLASH/ROM.

With the command **MAP.BOnchip** *<range>* it is possible to tell the debugger where you have ROM / FLASH on the target.

On-chip Breakpoints for Data

To stop the CPU at a read or write access to a memory location on-chip breakpoints are required. When the CPU attempts to access a memory cell, an exception stops code execution before the memory cell is accessed. The parameter NDBREAK of the Debug Option (Architectural Option of the Xtensa core) defines the number of available data breakpoints.

Example for Standard Breakpoints

Assume you have a target with NIBREAK=2, NDBREAK=2 and

- FLASH from `0x0--0xfffff`
- RAM from `0x100000--0x11ffff`

The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x0--0xfffff
```

The following standard breakpoint combinations are possible.

1. Unlimited breakpoints in RAM and one breakpoint in ROM/FLASH

```
Break.Set 0x100000 /Program     ; Software breakpoint 1
Break.Set 0x101000 /Program     ; Software breakpoint 2
Break.Set addr /Program         ; Software breakpoint 3
Break.Set 0x100 /Program        ; On-chip instruction breakpoint
```

2. Unlimited breakpoints in RAM and one breakpoint on a read or write access

```
Break.Set 0x100000 /Program     ; Software breakpoint 1
Break.Set 0x101000 /Program     ; Software breakpoint 2
Break.Set addr /Program         ; Software breakpoint 3
Break.Set 0x108000 /Write       ; On-chip data breakpoint
```

3. Two breakpoints in ROM/FLASH

```
Break.Set 0x100 /Program        ; On-chip instruction breakpoint 1
Break.Set 0x200 /Program        ; On-chip instruction breakpoint 2
```

4. Two breakpoints on a read or write access

```
Break.Set 0x108000 /Write       ; On-chip data breakpoint 1
Break.Set 0x108010 /Read        ; On-chip data breakpoint 2
```

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 ARM specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
VM	Virtual Memory (memory on the debug system)
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:0--3
```

Normally there is no need to use the following memory classes: P, D since program and data memory space are not separated.

MAP.BUS8 Bus width mapping

Format: **MAP.BUS8** [*<address_range>*]

This command is used to force the debugger to access the specified range with Load / Store 8-bit commands. So if you do a 32-bit wide memory dump (**Data.dump** *<address>* **/Long**) the debugger reads byte-by-byte while the window shows the information in 32-bit words.

Format: **MAP.BUS16** [*<address_range>*]

This command is used to force the debugger to access the specified range with Load / Store 16-bit commands. So if you do a 8-bit wide memory dump (**Data.dump** *<address>* **/Byte**) the debugger reads word-by-word while the window shows the information byte-by-byte.

As a follow the debugger might read more than the dump window shows, so if a memory cell is sensitive on read accesses you might touch it unintentional.

Format: **MAP.BUS32** [*<address_range>*]

This command is used to force the debugger to access the specified range with Load / Store 32-bit commands. So if you do a 8-bit wide memory dump (**Data.dump** *<address>* **/Byte**) the debugger reads 32-bit values while the window shows the information byte-by-byte.

As a follow the debugger might read more than the dump window shows, so if a memory cell is sensitive on read accesses you might touch it unintentional.

SYStem.CONFIG.state — Display target configuration

Format:	**SYStem.CONFIG.state** [**/***<tab>*]
<tab>:	**DebugPort** | **Jtag** | **MultiTap** | **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**.
MultiTap	Informs the debugger about the existence and type of a System/Chip Level Test Access Port. The debugger might need to control it in order to reconfigure the JTAG chain or to control power, clock, reset, and security of different chip components. For descriptions of the commands on the **MultiTap** tab, see **MultiTap**.

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>*
CoreNumber *<number>*
DEBUGPORT [**DebugCable0** | **DebugCableA** | **DebugCableB**]
DEBUGPORTTYPE [**JTAG** | **SWD** | **CJTAG**]
Slave [**ON** | **OFF**]
SWDPIDLEHIGH [**ON** | **OFF**]
SWDPTargetSel *<value>*
DAP2SWDPTargetSel *<value>*
TriState [**ON** | **OFF**]

<parameter>: **(JTAG)**
DAP2DRPOST *<bits>*
DAP2DRPRE *<bits>*
DAP2IRPOST *<bits>*
DAP2IRPRE *<bits>*
DAPDRPOST *<bits>*
DAPDRPRE *<bits>*
DAPIRPOST *<bits>*
DAPIRPRE *<bits>*
DRPOST *<bits>*
DRPRE *<bits>*

<*parameter*>: (JTAG cont.)

IRPOST<*bits*>
IRPRE <*bits*>

Slave [**ON** | **OFF**]
TAPState <*state*>
TCKLevel <*level*>
TriState [**ON** | **OFF**]

<*parameter*>: **(Multitap)**

MULTITAP [**NONE** | **IcepickA** | **IcepickB** | **IcepickC** | **IcepickD** | **IcepickBB** | **IcepickBC** | **IcepickCC** | **IcepickDD** | **STCLTAP1** | **STCLTAP2** | **STCLTAP3** |
MSMTAP <*irlength*> <*irvalue*> <*drlength*> <*drvalue*>
JtagSEQuence <*sub_cmd*>]

<*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*>

DAP2JTAGPORT <*port*>

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*>

<*parameter*>: (AccessPorts cont.)

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 [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
CTI.RESet
CTI.view

ETB.ATBSource <*source*>
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*>

<parameter>: (COmponents cont.)

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

TRAX.RESet
TRAX.view

TPIU.ATBSource *<source>*
TPIU.Base *<address>*
TPIU.Name *<string>*
TPIU.RESet
TPIU.Type [**CoreSight** | **Generic**]
TPIU.view

<parameter>: **(Deprecated)**

COREBASE *<address>*
CTIBASE *<address>*
CTICONFIG [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
DEBUGBASE *<address>*
ETBFUNNELBASE *<address>*
ETFBASE *<address>*
FUNNEL2BASE *<address>*
FUNNELBASE *<address>*
TPIUBASE *<address>*
TPIUFUNNELBASE *<address>*
view

AHBACCESSPORT *<port>*
APBACCESSPORT *<port>*
AXIACCESSPORT *<port>*
COREJTAGPORT *<port>*
DAP2COREJTAGPORT *<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.

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.

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.

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.

<parameters> describing a system level TAP “MultiTap”

A “Multitap” is a system level or chip level test access port (TAP) in a JTAG scan chain. It can for example provide functions to re-configure the JTAG chain or view and control power, clock, reset and security of different chip components.

MULTITAP
[**NONE** | **IcepickA** | **IcepickB** | **IcepickC** | **IcepickD** | **IcepickM** | **IcepickBB** | **IcepickBC** | **IcepickCC** | **IcepickDD** | **STCLTAP1** | **STCLTAP2** | **STCLTAP3** | **MSMTAP** *<irlength> <irvalue> <drlength> <drvalue>* **JtagSEQuence** *<sub_cmd>*]

Selects the type and version of the MULTITAP.

In case of MSMTAP you need to add parameters which specify which IR pattern and DR pattern needed to be shifted by the debugger to initialize the MSMTAP. Please note some of these parameters need a decimal input (dot at the end).

IcepickXY means that there is an Icepick version “X” which includes a subsystem with an Icepick of version “Y”.

For a description of the **JtagSEQuence** subcommands, see **SYStem.CONFIG.MULTITAP JtagSEQuence**.

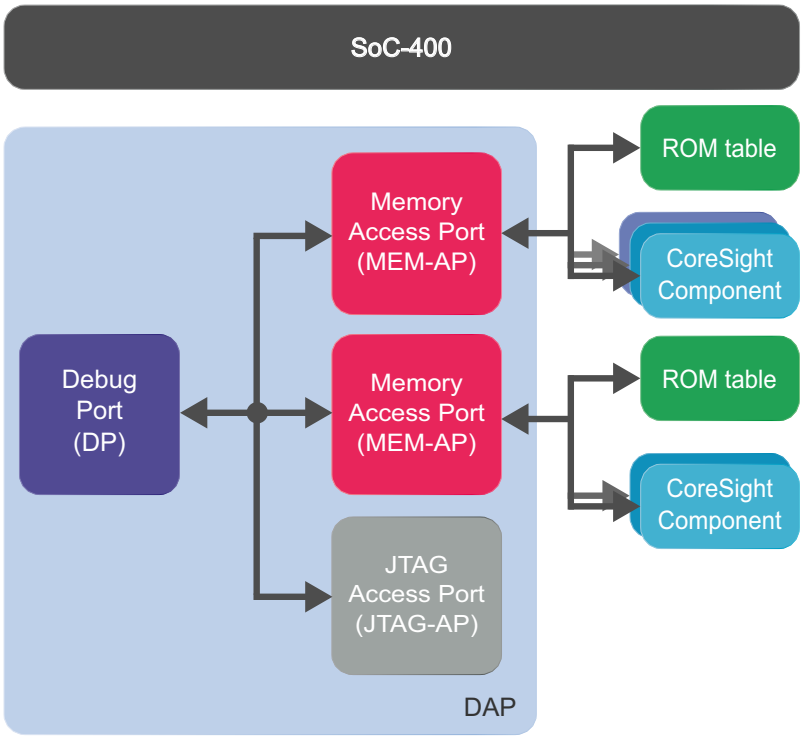
An Access Port (AP) is a CoreSight module from ARM which provides access via its debug link (JTAG, cJTAG, SWD, USB, UDP/TCP-IP, GTL, PCIe...) to:

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).

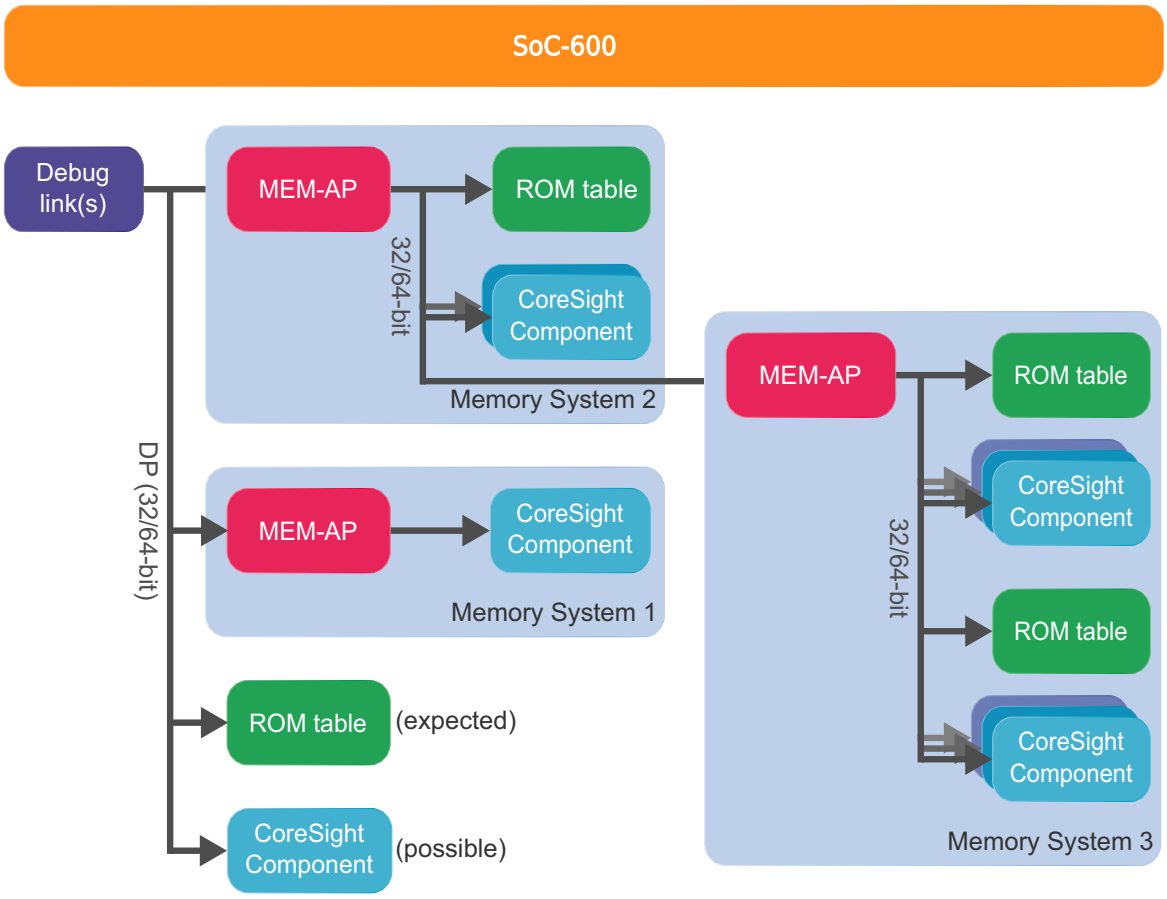
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number 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.AHBH-PROT [*<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>*]

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**]
SYStem.Option.AXIACEEnable [**ON** | **OFF**] (deprecated)

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>*
SYStem.Option.AXICACHEFLAGS *<value>* (deprecated)

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.

DAP2JTAGPORT *<port>*

JTAG-AP port number (0-7) for an (other) DAP which is connected to a JTAG-AP.

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)
DAP2COREJTAGPORT *<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.

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.

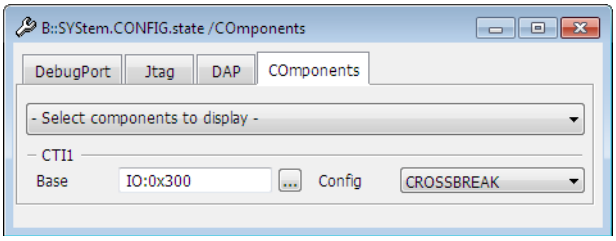
Example: SYStem.CONFIG.AXIAP1.Base DP:0x80004000
Meaning: The control register block of the AXI access ports starts at address 0x80004000.

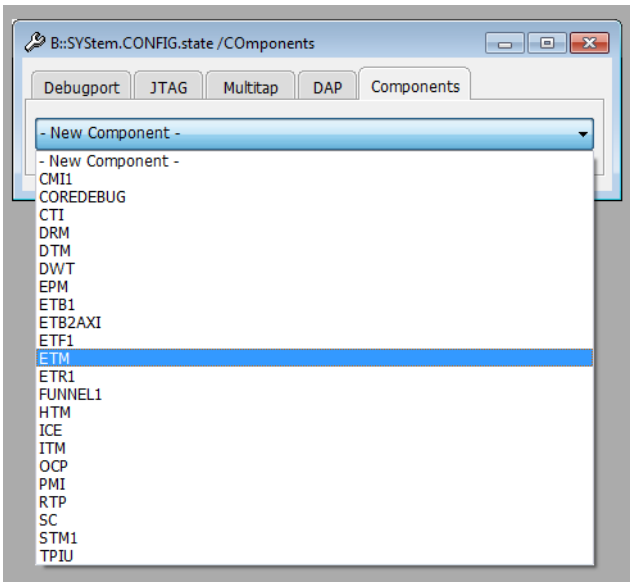
JTAGAPn.Base *<address>*

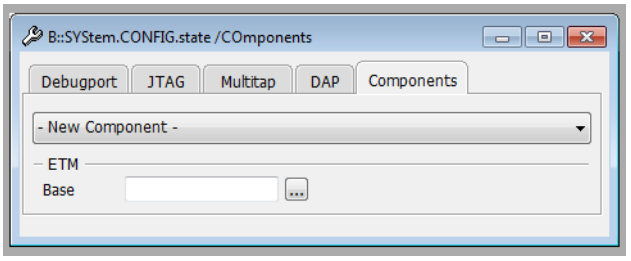
This command informs the debugger about the start address of the register block of the “JTAGAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus 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.

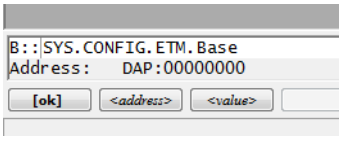
On the **Components** tab in the **SYStem.CONFIG.state** window, you can comfortably add the debug and trace components your chip includes and which you intend to use with the debugger's help.







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.

In an SMP (Symmetric MultiProcessing) debug session you can enter for the components COREBEBUG, CTI, ETB, ETF, 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 Embedded Trace Buffer. The ETB 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.

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, 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.

ETS.ATBSource *<source>*
ETS.Base *<address>*
ETS.RESet
Embedded Trace Streamer (ETS) - ARM CoreSight module

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.

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).

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.

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.

On an SMP debug session some of these commands can have a list of *<port>* parameter.

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.

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)
COREBASE *<address>*	**COREDEBUG.Base** *<address>*
CTIBASE *<address>*	**CTI.Base** *<address>*
DEBUGBASE *<address>*	**COREDEBUG.Base** *<address>*
ETBBASE *<address>*	**ETB1.Base** *<address>*
ETBFUNNELBASE *<address>*	**FUNNEL4.Base** *<address>*
ETFBASE *<address>*	**ETF1.Base** *<address>*
FUNNEL2BASE *<address>*	**FUNNEL2.Base** *<address>*
FUNNELBASE *<address>*	**FUNNEL1.Base** *<address>*
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>:	**XTENSA** | **DC108MINI** | **DC212GP** | **DC232L** | **DC330HIFI** | **DC545CK** | **DC570T**

Selects the processor type. IF XTENSA is selected the debugger detects the architectural options from the CPU.

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

Default frequency: 1 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.
RTCK	The JTAG interface of Xtensa does not offer RTCK (**R**eturned **TCK**). However, in multicore applications with ARM, RTCK can be used to control the JTAG clock. On some processor derivatives, there is the need to synchronize the processor clock and the JTAG clock. In this case RTCK shall be selected. Synchronization is maintained, because the debugger does not progress to the next TCK edge until after an RTCK edge is received. In case you have a processor derivative requiring a synchronization of the processor clock and the JTAG clock, but your target does not provide an RTCK signal, you need to select a fix JTAG clock below 1/6 of the processor clock (ARM7, ARM9), below 1/8 of the processor clock (ARM11), respectively. When RTCK is selected, the frequency depends on the processor clock and on the propagation delays. The maximum reachable frequency is about 16 MHz.

ARTCK	Accelerated method to control the debug clock by the RTCK signal (Accelerated Returned TCK). This option is only relevant for JTAG debug ports. For designs using a very low processor clock we offer a different mode (ARTCK). In **ARTCK** mode, the debugger uses a fixed frequency for TCK, independent of the RTCK signal. This frequency must be specified by the user and has to be below 1/3 of the processor clock speed. TDI and TMS will be delayed by 1/2 TCK clock cycle. TDO will be sampled with RTCK.
CTCK	With this option higher debug port speeds can be reached. The TDO/SWDIO signal will be sampled by a signal which derives from TCK/SWCLK, but which is timely compensated regarding the debugger-internal driver propagation delays (**C**ompensation by **TCK**). This feature can be used with a debug cable version 3 or newer. If it is selected, although the debug cable is not suitable, a fixed frequency will be selected instead (minimum of 10 MHz and selected clock).
CRTCK	With this option higher debug port speeds can be reached. The TDO/SWDIO signal will be sampled by the RTCK signal. This compensates the debugger-internal driver propagation delays, the delays on the cable and on the target (**C**ompensation by **RTCK**). This feature requires that the target provides an RTCK signal. In contrast to the **RTCK** option, the TCK/SWCLK is always output with the selected, fixed frequency.

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 Xtensa 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 Real-time memory access (non-intrusive)

Format: **SYStem.MemAccess Enable | StopAndGo | Denied | DAP**
SYStem.ACCESS (deprecated)

Enable
CPU (deprecated)
Real-time memory access during program execution to target is enabled.

Denied (default)
Real-time 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.

DAP
A run-time memory access is done via the ARM CoreSight v2 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.

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. The automatic endian detection does not work in this case. Set the **SYStem.Option.Endianness** to Little or Big before executing **SYStem.Mode Attach**.
StandBy	Not available for Xtensa.
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*> (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.

Format: **SYStem.Option.DAP2DBGPWRUPREQ** [**ON** | **AlwaysON**]

Default: ON.

This option controls the DBGPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port 2 (DAP2) before and after the debug session. Debug power will always be requested by the debugger on a debug session start.

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 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.DAP2DBGPWRUPREQ** is set to **AlwaysON**.

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.

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

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.DAP2SYSPWRUPREQ Force system power in DAP2

Format: **SYStem.Option.DAP2SYSPWRUPREQ** [**AlwaysON** | **ON** | **OFF**]

Default: ON.

This option controls the SYSPWRUPREQ bit of the CTRL/STAT register of the Debug Access Port 2 (DAP2) 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.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.

SYStem.Option.DISableHwWatchDOG Disable watchdog when core stops

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

Some SoCs contain a hardware watchdog. If this option is active, the debugger disables the watchdog when the core stops, e.g. due to breakpoint.

ON Disables watchdog.

OFF Idle.

Format: **SYStem.Option.DisMode** *<mode>*

<mode>: **INTernal**
LIBrary
INTbeforeLIB
LIBbeforeINT

Defines the disassembler mode.

INTernal Use TRACE32 internal disassembler.

LIBrary Use disassembler from TIE library defined with **SYStem.TIE** commands, see example below.

INTbeforeLIB Prefer TRACE32 internal disassembler, use TIE library as fallback.

LIBbeforeINT Prefer TIE library, use TRACE32 internal disassembler as fallback.

Example:

```
SYStem.TIE.DELete                       ; Delete all already added files

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll  ; Add TIE library files
SYStem.TIE.AddCoreLibrary libisa-core.dll
SYStem.TIE.AddCoreLibrary libisa-DC_330HiFi.dll

SYStem.TIE.ENAble                      ; Load and enable TIE Instructions

SYStem.Option.DisMode LIBbeforeINT   ; set disassembler preference
```

SYStem.Option.Endianness Specify the byte ordering

Format: **SYStem.Option.Endianness** [**AUTO** | **Little** | **Big**]

Default: AUTO.

The instructions for the JTAG connection to the Xtensa core depend on the byte ordering. If AUTO is selected, the debugger detects the endianness when leaving down state. This does not work for **SYStem.Mode Attach**.

SYStem.Option.EnReset Allow the debugger to drive nRESET (nSRST)

[SYStem.state window> EnReset]

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

Default: ON.

If this option is disabled the debugger will never drive the nRESET (nSRST) line on the JTAG connector. This is necessary if nRESET (nSRST) is no open collector or tristate signal.

From the view of the 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.EnTRST Allow debugger to drive TRST

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

Default: ON.

If this option is disabled, the nTRST line is never driven by the debugger (permanent high). Instead the debugger attempts to capture the same effect to the TAP controller by consecutive TCK pulses with TMS high.

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.

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.IntelSOC Slave core is part of Intel® SoC

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

Default: OFF.

Informs the debugger that the core is part of an Intel® SoC. When enabled, all IR and DR pre/post settings are handled automatically, no manual configuration is necessary.

Requires that the debugger for this core is slave in a multicore setup with x86 as the master debugger and that **SYStem.Option.CLTAPOnly** is enabled in the x86 debugger.

SYStem.Option.MMUSPACES Separate address spaces by space IDs

Format: **SYStem.Option.MMUSPACES** [**ON** | **OFF**]
SYStem.Option.MMUspaces [**ON** | **OFF**] (deprecated)
SYStem.Option.MMU [**ON** | **OFF**] (deprecated)

Default: OFF.

Enables the use of space IDs for logical addresses to support **multiple** address spaces.

For an explanation of the TRACE32 concept of address spaces (zone spaces, MMU spaces, and machine spaces), see **"TRACE32 Concepts"** (trace32_concepts.pdf).

NOTE:	**SYStem.Option.MMUSPACES** should not be set to **ON** if only one translation table is used on the target. If a debug session requires space IDs, you must observe the following sequence of steps: 1. Activate **SYStem.Option.MMUSPACES**. 2. Load the symbols with **Data.LOAD**. Otherwise, the internal symbol database of TRACE32 may become inconsistent.

Examples:

```
;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x012A:
Data.dump D:0x012A:0xC00208A

;Dump logical address 0xC00208A belonging to memory space with
;space ID 0x0203:
Data.dump D:0x0203:0xC00208A
```

SYStem.Option.PWROVR — Specifies power override bit

Format: **SYStem.Option.PWROVR** [**ON** | **OFF**] (deprecated)

Specifies the power override bit when a certain derivative providing this function is selected.

SYStem.Option.SOFTLONG — Use 32-bit access to set breakpoint

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

Default: OFF.

This option instructs the debugger to use 32-bit accesses to patch the software breakpoint code.

MAP.BUS8 / **MAP.BUS16** / **MAP.BUS32** does not influence the access used for patching the software breakpoint code. So if you use **MAP.BUS32** for code area you have to activate this option.

SYStem.Option.ResetDetection Supervise reset

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

Default: ON.

Selects if the debugger takes account of an external target reset.

ON	External resets are supervised.
OFF	External resets are ignored.

SYStem.Option.RUNSTALLMASKASM Disable RunStall while step

Format: **SYStem.Option.RUNSTALLMASKASM** [**AUTO** | **ON** | **OFF**]

Default: AUTO.

If enabled, the RunStallInEn bit of the Debug Control Register (DCR) is cleared during assembler single-step operations. This is required for some multicore CPUs.

AUTO	**AUTO** enables this option for some pre-defined CPUs.
ON, **OFF**	Use **ON** or **OFF** to explicitly enable or disable this option.

SYStem.Option.SnoopAddressPC Program counter snoop address

Format: **SYStem.Option.SnoopAddressPC** *<address>* | *<addressrange>* | *<name>*

Some SoCs allow to read the program counter during runtime. Use this option to tell the debugger where to read the program counter.

Example:

```
SYStem.Option.SnoopAddressPC EAXI:0x12345678
```

SYStem.Option.SPILLLOC Temporary memory

Format: **SYStem.Option.SPILLLOC** *<start_address>*

Tells the debugger where to find memory which can be used to store data and to execute small pieces of code (max. 256 bytes).

Some configurations contain registers which cannot be accessed directly. They can only be accessed by executing a sequence of instructions. For this task, a small area of RAM is required. The debugger saves the contents before the memory is used and restores the original contents after usage. With this option, you can specify the first address of the memory range the debugger can use.

SYStem.Option.TriggerHwWatchDOG Trigger hardware watchdog

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

Default: OFF.

Some SOCs contain a hardware watchdog. If this option is active, the debugger triggers the watchdog while real time execution is stopped. Make sure to keep the watchdog timer period long enough, to give the debugger a chance to meet the timing.

ON Idle.

OFF Trigger.

SYStem.Option.WindowVectorBase VECBASE initial value

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

If the Relocatable Vector Option was added to a Xtensa configuration, the core contains a special register VECBASE.

Specifies initial value of the special register VECBASE for simulation purposes.

Format: **SYStem.Option.WinRegOption** [*/<option>*]

<option>: **AUTO** | **OFF** | **32** | **64**

Tells the debugger if the Architectural Option with the name Windowed Register Option was configured. It tells you the number of physical registers. You can have:

32 32 core registers if this Architectural Option is not configured to have 32 registers.

64 64 core registers if this Architectural Option is not configured to have 32 registers.

AUTO The option **AUTO** tells the debugger to detect the used setting from the hardware.

OFF 16 core registers if this Architectural Option is not configured.

The **SYStem.TIE** command group is used to configure TRACE32 to deal with architectural extensions. One important extension, the Tensilica Instruction Extension gave the name for this set of commands.

The Tensilica tool chain generates libraries for a custom configuration. These libraries can be used to extract information on the usage of architectural options, additional instructions and registers.

SYStem.TIE.AddCoreLibrary Add library file

Format: **SYStem.TIE.AddCoreLibrary** *<file>*
SYStem.TIE.ADDtiedll *<file>* (deprecated)

Adds TIE library file to the TRACE32, which can be used to improve disassembly for custom configurations. It is important to add all needed library files. The TIE library files need to be added in the correct order, since they are internally dependent. If any file is missing an error may appear after executing **SYStem.TIE.ENAble** command.

On Linux systems it might be required to add the path to the LD_LIBRARY_PATH environment variable *before* starting TRACE32.

Example:

```
SYStem.TIE.AddCoreLibrary libisa-core.dll
```

For a complete example see **SYStem.TIE.ENAble** command.

SYStem.TIE.ADDALLtiedll Add all library files

Format: **SYStem.TIE.ADDALLtiedll** *<directory>* (deprecated)

Adds all TIE library files within the specified folder to TRACE32. It is recommended to use **SYStem.TIE.AddCoreLibrary** instead, since the library files need to be added in the correct order.

Format: **SYStem.TIE.ADPerdll** *<file>*

Adds TIE library file for peripheral file generation (per file), see example at **SYStem.TIE.GENper** command.

On Linux systems it might be required to add the path to the LD_LIBRARY_PATH environment variable *before* starting TRACE32.

SYStem.TIE.CMList Instructions to display custom registers

Format: **SYStem.TIE.CMList** *<file>*

Generates the instructions the debugger needs to display custom registers, see example at **SYStem.TIE.GENper** command.

SYStem.TIE.DELete Remove all library files

Format: **SYStem.TIE.DELete**

Removes all added TIE library files from TRACE32. This command is recommended before **SYStem.TIE.AddCoreLibrary** to be sure that there are no other library files added.

Example:

```
SYStem.TIE.DELete
```

SYStem.TIE.DEPerdll Remove all library files for per file

Format: **SYStem.TIE.DEPerdll**

Removes all TIE library files added with **SYStem.TIE.ADPerdll** for peripheral file generation (per file). This command is recommended before **SYStem.TIE.ADPerdll** to be sure that there are no other library files added.

Format: **SYStem.TIE.DISable**

All loaded TIE library files are unloaded from disassembler decoder. Instructions are decoded only by the internal TRACE32 decoder. To restart decoding with TIE library files use the command **SYStem.TIE.ENAble**.

Example:

```
SYStem.TIE.DISable
```

SYStem.TIE.ENAble Load and enable TIE instructions

Format: **SYStem.TIE.ENAble**

Loads all added TIE library files to the TRACE32 disassembler. From this moment all instructions are decoded by internal TRACE32 decoder and TIE library files. Before you execute this command, it is necessary to add all needed library files to the TRACE32 otherwise an error will appear and TIE library files will not be loaded. To add the file use **SYStem.TIE.AddCoreLibrary** command. To set you disassembler preference use **SYStem.Option.DisMode** command.

Example:

```
SYStem.TIE.DELete                       ; Delete all already added files

SYStem.TIE.AddCoreLibrary libisa-core-hw.dll  ; Add TIE library files
SYStem.TIE.AddCoreLibrary libisa-core.dll
SYStem.TIE.AddCoreLibrary libisa-DC_330HiFi.dll

SYStem.TIE.ENAble                      ; Load and enable TIE Instructions

SYStem.Option.DisMode LIBbeforeINT   ; Set disassembler preference
```

SYStem.TIE.GENper Generate peripheral file

Format: **SYStem.TIE.GENper** *<file>*

Generates a custom peripheral file from the loaded libraries.

Example:

```
SYStem.TIE.ToolLibraryPath "./tools/lib"

SYStem.TIE.ADPerdll "./core/libisa-core-hw.dll"
SYStem.TIE.ADPerdll "./core/libisa-core.dll"

SYStem.TIE.GENper "my_hifi2.per"
SYStem.TIE.CMList "my_hifi2.cmm"

DO "my_hifi2.cmm"
PER.view my_hifi2.per"
```

SYStem.TIE.GETArchOPTions Detect architectural options from libraries

Format: **SYStem.TIE.GETArchOPTions** *<file>*

Detects architectural options from the loaded libraries.

Example:

```
SYStem.TIE.ToolLibraryPath "./tools/lib"

SYStem.TIE.ADPerdll "./core/libisa-core-hw.dll"
SYStem.TIE.ADPerdll "./core/libisa-core.dll"

SYStem.TIE.GETArchOPTions

SYStem.Mode Up
```

SYStem.TIE.ToolLibraryPath Specify path for library tools

Format: **SYStem.TIE.ToolLibraryPath** *<directory>*
SYStem.TIE.LIBpath *<directory>* (deprecated)

Tells the debugger where to search for the tools to handle core specific libraries.

To extract information from delivered libraries a set of tools is needed. These tools can be found within additional libraries like xtisa.dll, xtparams.dll and xtdebug.dll. TRACE32 needs to know where to find these files.

Be aware that the same release revision is needed as you selected for your XGP request to Cadence®.

On Linux systems it might be required to add the path to the LD_LIBRARY_PATH environment variable *before* starting TRACE32.

SYStem.TIE.REGlist Internal use only

Format: **SYStem.TIE.REGlist** *<file>*

SYStem.TIE.RESet Reset TIE

Format: **SYStem.TIE.RESet**

Resets TIE to initial state.

Xtensa Specific Benchmarking Commands

The **BMC** (**B**ench**M**ark **C**ounter) commands provide control of the on-chip performance counters. The counters can be configured to count certain events in order to get statistics on the operation of the processor and the memory system.

The counters of Xtensa cores can be read at run-time.

For further Information please refer to section “Performance Monitor for Xtensa LX Processors” or “Performance Monitor for Xtensa NX Processors” within “xtensa_debug_guide.pdf”.

- Please note, due to influence of the Debugger the derived counter values could be slightly higher.
- Please note, the feature can only be used, if the performance monitor counters are configured within the core.

BMC.<counter>.EVENT — Assign event to counter

[build 147523 - DVD 09/2022]

Format:	**BMC.***<counter>***.EVENT** *<event>*
<counter>:	**PM0** | **PM1** | … (depending on configuration)
<event>:	**OFF** | **CYCLE** | **OVFL_PREV_COUNTER** | … (depending on NX/LX core, cmp xtensa_debug_guide.pdf)

Performance Monitors - short PM - are implemented as 32-bit hardware counter. They collect information about the throughput of the target processor and its pipeline stages. They count certain events, like cache misses or CPU cycles. Further, they deliver information about the efficiency of the instruction or data cache, the TLBs (translation look aside buffers) and some other performance values. This information may be helpful in finding bottlenecks and tuning the application.

A list for countable events can be found in the document “xtensa_debug_guide.pdf” provided by Cadence.

Format:	**BMC.***<counter>***.KRNLCNT** *<mode>*
<counter>:	**PM0** | **PM1** | … (depending on configuration)
<mode>:	**LESSEQUAL** | **GREATER**

Depending on the selected mode, the performance monitors only count if

KRNLCNT	on LX cores	on NX cores
LESSEQUAL	CINTLEVEL <= TRACELEVEL	EXECLEVEL <= TRACESCOPE
GREATER	CINTLEVEL > TRACELEVEL	EXECLEVEL > TRACESCOPE

For further details please refer to “xtensa_debug_guide.pdf” provided by Cadence.

BMC.<counter>.TRACELEVEL Set counting threshold

[build 147783 - DVD 09/2022]

Format:	**BMC.***<counter>***.TRACELEVEL** *<val>*
<counter>:	**PM0** | **PM1** | … (depending on configuration)
<val>:	**0** | **1** | … | **7**

LX core only.

- CINTLEVEL is the Xtensa core’s current interrupt level as defined in the Xtensa Instruction Set Architecture (ISA) Reference Manual

Format:	**BMC.***<counter>***.TRACESCOPE** *<val>*
<counter>:	**PM0** | **PM1** | … (depending on configuration)
<val>:	**0** | **1** | … | **7**

NX core only.

- EXECLEVEL is the Xtensa core's current execution level as reported by the traceport. Refer to operating system relationship with EXECLEVEL for more details.
- The Xtensa processor's current execution level (EXECLEVEL) is a value reported on the Traceport that classifies the code currently running in categories useful to track for profiling or tracing purposes, such as: application thread, exception handler, interrupt handler, or dispatch code. For example, this allows performance counters to selectively count at specified execution levels, using TRACESCOPE and KRNLCNT fields described in Control Register.For further details please refer to "xtensa_debug_guide.pdf" provided by Cadence.

TERM.METHOD.BRK1_14 Define communication protocol

Format: **TERM.METHOD.BRK1_14** [*<address>*]

BRK1_14 The command **TERM.METHOD.BRK1_14** tells the debugger to use GNU SYSCALL operations for terminal communication.
Use **TERM.view** to open the terminal window and to activate the communication.
When an application reaches “break 1,14”, the application is stopped and the debugger checks for the type of SYSCALL.

- When receiving a SYSCALL_WRITE operation, the debugger writes the relevant information to the terminal window and returns to **Go** state, i.e. starts to execute user code again.
- When receiving a SYSCALL_READ, the application remains stopped. The debugger is waiting for some input to the terminal window. When you press the **Enter** key, the application resumes operation.

The handling of "break 1,14" is only active when the **TERM.view** window is open while **TERM.METHOD BRK1_14** is selected. In all other cases, "break 1,14" is treated as a normal software break instruction.

For a description of the other options, see **TERM.METHOD**.

CPU specific TrOnchip Commands

The **TrOnchip** command group provides full access to both ICE Breaker units called A and B. Most of the features can also utilized easier by setting regular breakpoints (**Break.Set** command).

The TrOnchip commands are only visible if the debugger detects TRAX-PC hardware. They cannot be modified when the onchip trace is disabled.

For the bit descriptions of the control registers, please refer to the *Trace Solutions User's Guide* of the chip/core manufacturer.

TrOnchip.BIEN

Break-out relay enable

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

TrOnchip.BOEN

Break-in relay enable

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

TrOnchip.CTOWS Cross-trigger output enable when trace stop completes

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

TrOnchip.CTOWT Cross-trigger output enable when trace stop triggered

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

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

Default: OFF.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

TrOnchip.PTOWS Processor trigger output enable

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

Default: OFF.

Enables the processor trigger output when trace stop completes.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

TrOnchip.PTOWT Processor trigger output enable

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

Default: OFF.

Enables the processor trigger output when trace stop triggered.

Only available, when the TRAX Onchip Trace is configured and not disabled.

ON Enable.

OFF Disable.

Format: **TrOnchip.RESet**

Resets all TrOnchip settings.

TrOnchip.state

Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

MMU.DUMP Page wise display of MMU translation table

Format:	**MMU.DUMP** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.dump** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** *<cpu_specific_tables>*

Displays the contents of the CPU specific MMU translation table.

- If called without parameters, the complete table will be displayed.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Displays the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: displays the translation table of the specified process • else, this command displays the table the CPU currently uses for MMU translation.

KernelPageTable	Displays the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and displays its table entries.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Displays the MMU translation table entries of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and displays its table entries. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

CPU specific Tables

TLB	Displays the contents of the Translation Lookaside Buffer.

Format:	**MMU.List** *<table>* [*<range>* | *<address>* | *<range>* *<root>* | *<address>* *<root>*] **MMU.***<table>***.List** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**

Lists the address translation of the CPU-specific MMU table.

- If called without address or range parameters, the complete table will be displayed.
- If called without a table specifier, this command shows the debugger-internal translation table. See **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed if their **logical** address matches with the given parameter.

<root>	The *<root>* argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.
<range> *<address>*	Limit the address range displayed to either an address range or to addresses larger or equal to *<address>*. For most table types, the arguments *<range>* or *<address>* can also be used to select the translation table of a specific process if a space ID is given.
PageTable	Lists the entries of an MMU translation table. • if *<range>* or *<address>* have a space ID: list the translation table of the specified process • else, this command lists the table the CPU currently uses for MMU translation.
KernelPageTable	Lists the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the MMU translation table of the kernel and lists its address translation.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Lists the MMU translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and lists its address translation. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manuals**.

Format:	**MMU.SCAN** *<table>* [*<range>* *<address>*] **MMU.***<table>***.SCAN** (deprecated)
<table>:	**PageTable** **KernelPageTable** **TaskPageTable** *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0** **ALL** [**Clear**] *<cpu_specific_tables>*

Loads the CPU-specific MMU translation table from the CPU to the debugger-internal static translation table.

- If called without parameters, the complete page table will be loaded. The list of static address translations can be viewed with **TRANSlation.List**.
- If the command is called with either an address range or an explicit address, page table entries will only be loaded if their **logical** address matches with the given parameter.

Use this command to make the translation information available for the debugger even when the program execution is running and the debugger has no access to the page tables and TLBs. This is required for the real-time memory access. Use the command **TRANSlation.ON** to enable the debugger-internal MMU table.

PageTable	Loads the entries of an MMU translation table and copies the address translation into the debugger-internal static translation table. • if *<range>* or *<address>* have a space ID: loads the translation table of the specified process • else, this command loads the table the CPU currently uses for MMU translation.

KernelPageTable	Loads the MMU translation table of the kernel. If specified with the **MMU.FORMAT** command, this command reads the table of the kernel and copies its address translation into the debugger-internal static translation table.
TaskPageTable *<task_magic>* | *<task_id>* | *<task_name>* | *<space_id>***:0x0**	Loads the MMU address translation of the given process. Specify one of the **TaskPageTable** arguments to choose the process you want. In MMU-based operating systems, each process uses its own MMU translation table. This command reads the table of the specified process, and copies its address translation into the debugger-internal static translation table. • For information about the first three parameters, see **"What to know about the Task Parameters"** (general_ref_t.pdf). • See also the appropriate **OS Awareness Manual**.
ALL [**Clear**]	Loads all known MMU address translations. This command reads the OS kernel MMU table and the MMU tables of all processes and copies the complete address translation into the debugger-internal static translation table. See also the appropriate **OS Awareness Manual**. **Clear:** This option allows to clear the static translations list before reading it from all page translation tables.

The Xtensa trace protocol is based on Nexus (R) commands, so the NEXUS command group is used to configure the TRAX trace properties.

NEXUS.CLOCK Specify the frequency of the timestamp counter

Format: **NEXUS.CLOCK** *<frequency>*

Default: 0.

Specifies the frequency of the timestamp counter clock. An external logic block supplies the 64 lines DebugExtTime (input lines of the Xttop Module) with a monotonically increasing 64-bit time value. The clock of this counter is expected to be synchronous to the Xtensa clock (Signal CLK of the Xttop Module).

To calculate time values from the timestamp value within a trace packet, TRACE32 needs to know how much time one counter tick takes.

Within a CoreSight environment, the external logic block typically is a CoreSight timestamp Interpolator.

NEXUS.ON Switch the NEXUS trace port on

Format: **NEXUS.ON**

Default: OFF.

This option controls the ATB enable (ATEN) in the Trax Control Register. The output of the TRAX trace compressor is sent to the ATB interface to feed the funnel of a Coresight environment for example.

ON	Send trace data out on the ATB interface.
OFF	No trace data is sent to the ATB interface.

Format: **NEXUS.RESet**

Resets NEXUS trace port settings to default settings.

NEXUS.TraceID Specify the trace ID

Format: **NEXUS.TraceID** *<ID>* [0...127]

Default:1.

selects a unique number within your trace sources.

Default: AUTO.

The command **NEXUS.TraceID** sets the ATB-ID used by the TRAX trace logic, when emitting the trace stream to the CoreSight ATB. The value gets written to the bitfield ATID of the TRAX Control register.

Every trace stream must have a different ID inside the same CoreSight ATB network. It is especially important with AMP multicore configurations to ensure that every trace producer uses a different ID.

In SMP multicore configurations this command sets the ID of the first core, while the ID is incremented for each consecutive core in the same SMP cluster.

NEXUS.TImeMode Generate timestamps to the trace data

Format: **NEXUS.TImeMode** *<mode>*

<mode> **NexusTimeStamps**
OFF

This option controls the TSEN bit of the Trax Control Register. It can be set only, if the TRAX Trace Buffer Module of the Xtensa configuration is configured to generate timestamp information.

NexusTimeStamps The TRAX trace encoder generates time information

OFF No time information.

IDC20A Debug Cable

Mechanical Description of the IDC20A Debug Cable:

Signal	Pin	Pin	Signal
VTREF	1	2	VSUPPLY(not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
SRST-	15	16	GND
(DBGRQ)	17	18	GND
(DBGACK)	19	20	GND

Electrical Description of the IDC20A Debug Cable:

- TCK, TMS, TDI and nTRST are driven by CMOS drivers which are supplied with a voltage following the level at VCCS. Therefore the ICD can work in an voltage range of 1.8 … 5.0 V. In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the JTAG port. In environments where multiple tools can access the JTAG port, it is absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a hand-over between different tools.
- TDO is ICD input only.
- VCCS is used as a sense line for the target voltage. It is also used to define the level which is generated to supply the output drivers of the ICD interface to make an adaptation to the target voltage (I(VCCS) appr. 3 mA).
- nRESET (= nSRST) is used by the debugger to reset the target CPU or to detect a reset on the target. It is driven by an open collector buffer. The debugger will only assert a pulse on nRESET when the **SYStem.Up** command is executed.

Be careful with Pin17 and Pin19.

- They are intended to carry the signals DBGRQ and DBGACK, when the Xtensa core is embedded into an Arm based debug environment.
- Do not connect Pin17 and Pin 19 on the target side in other cases. Otherwise the hardware of the debugger can get damaged. Pin 2, originally intended to supply debugger hardware with power, is not in use and should not be connected to the target hardware either. Termination circuitry on the debugger hardware makes sure that these pins are not floating.

Mechanical Description of the 14-pin Debug Cable:

Signal	Pin	Pin	Signal
TDI	1	2	GND
TDO	3	4	GND
TCK	5	6	GND
N/C	7	-	KEY PIN
RESET-	9	10	TMS
VCCS	11	12	N/C
N/C	13	14	TRST-

Tensilica has specified Pin 8 as a mechanical KEY Pin to define the orientation of the connector. This is a standard 14 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.).

Electrical Description of the 14-pin Debug Cable:

- TCK, TMS, TDI and nTRST are driven by CMOS drivers which are supplied with a voltage following the level at VCCS. Therefore the ICD can work in an voltage range of 1.8 … 5.0 V. In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the JTAG port. In environments where multiple tools can access the JTAG port, it is absolutely required that there is a pull down resistor at TCK. This is to ensure that TCK is low during a hand-over between different tools.
- TDO is ICD input only.
- VCCS is used as a sense line for the target voltage. It is also used to define the level which is generated to supply the output drivers of the ICD interface to make an adaptation to the target voltage (I(VCCS) appr. 3 mA).
- nRESET (= nSRST) is used by the debugger to reset the target CPU or to detect a reset on the target. It is driven by an open collector buffer. The debugger will only assert a pulse on nRESET when the **SYStem.Up** command is executed.