LAUTERBACH
DEVELOPMENT TOOLS

C5000 Debugger

MANUAL

C5000 Debugger

C5000 Debugger

Version 06-Jun-2024

Introduction

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 C5000 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/c5000/ subfolder of the system directory of TRACE32.

Converter from GEL to PRACTICE

The General Extension Language (GEL) is an interpretive language similar to C that lets you create functions to extend Code Composer Studio's usefulness. The converter allows you to convert GEL language into PRACTICE scripts (*.cmm), which can be used directly in TRACE32.

For more detailed information on that converter please refer to **"Converter from GEL to PRACTICE"** (converter_gel.pdf).

Warning

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

DSP specific Implementations

Trigger

A bidirectional trigger system allows the following two events:

- Trigger an external system (e.g. logic analyzer) if the program execution is stopped.
- Stop the program execution if an external trigger is asserted.

For more information refer to the **TrBus** command.

Breakpoints

Software Breakpoints

If a software breakpoint is used, the original code at the breakpoint location is temporarily patched by a breakpoint code. There is no restriction in the number of software breakpoints.

On-chip Breakpoints for Instructions

If on-chip breakpoints are used, the resources to set the breakpoints are provided by the CPU. Those CPU resources only allow to set single address instruction breakpoints.

On-chip Breakpoints for Data

To stop the CPU after a read or write access to a memory location on-chip breakpoints are required. In the DSP notation these breakpoints are called watch points (WP).

Overview

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

Core	On-chip breakpoints	Instruction breakpoints	Read/Write breakpoint	Data Value breakpoints
C54x	2	2 single address	—	—
C55x	4	up to 4 single address	up to 3 data, 1 breakpoint range and 2 bit masks	up to 3

Memory Classes

The following DSP specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
IO	Input/Output Area
VM	Virtual Memory (memory on the debug system)
E	Emulation Memory, Pseudo Dualport Access to Memory (see **SYStem.CpuAccess**)

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

Example:

```
Data.dump IO:0- -3
```

SYStem.Option.IMASKASM Disable interrupts while single stepping

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

Default: OFF.

Interrupts are disabled during an assembler single-step operations, if this option is "ON".

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

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

Default: OFF.

Interrupts are disabled during HLL single-step operations, if this option is "ON".

SYStem.CPU Select the used CPU

Format: **SYStem.CPU** *<cpu>*

<cpu>: **C55XX** | **C5510** | **OMAP1510** | **OMAP1610** | **LEAD3PH2** | **LEAD3PH3** | …

Default selection: C55XX. Selects the processor type. If your ASIC is not listed, select the type of the integrated DSP core.

Format: **SYStem.JtagClock** [<*frequency*> | **RTCK** | **RTCK** <*frequency*>] | **ARTCK** <*frequency*>]

<*frequency*>: **10000.** … **80000000.**

Default frequency: 10 MHz.

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

<*frequency*>
The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the "System Settings" 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 clock is controlled by the RTCK signal (Returned TCK).
On some processor derivatives including an ARM core (e.g. OMAP) 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.

When RTCK is selected, the maximum reachable frequency is limited to 10 MHz. This limit can be changed by adding the frequency parameter. A limitation is required that the JTAG clock speed can not become higher than the physical interface can manage.

Example:
```
SYStem.JtagClock RTCK 20MHz
```

ARTCK
Accelerated method to control the JTAG clock by the RTCK signal (Accelerated Returned TCK).

RTCK mode allows theoretical frequencies up to 1/6 of the processor clock. For designs using a very low processor clock we offer a different mode (ARTCK) which does not work as recommended by ARM and might not work on all target systems. In ARTCK mode the debugger uses a fixed JTAG frequency for TCK, independent of the RTCK signal. This frequency must be specified by the user and has to be below 1/2 of the processor clock speed. The signal RTCK clocks TDI and TMS and controls the sampling of TDO.

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.

SYStem.MemAccess Select run-time memory access method

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

<*mode*>: **Enable**
Denied
StopAndGo

Default: Denied.

Enable Access is made without CPU intervention. This is only possible on the CPU (deprecated) instruction set simulator.

StopAndGo Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

Denied No memory access is possible without stopping the CPU.

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
Prepare
Go
Attach
StandBy
Up

Default: Down.

Configures how the debugger connects to the target and how the target is handled.

Down Disables the debugger. 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.

Prepare The debugger initializes the debug port (JTAG, SWD, cJTAG) and CoreSight DAP interface, but does not connect to the CPU.
This debug mode is used if the CPU shall not be debugged or bypassed, i.e. the debugger can access the memory busses, such as AXI, AHB and APB, directly through the memory access ports of the CoreSight DAP.

Typical use cases:
- The debugger accesses (physical) memory and bypasses the CPU if a mapping exists. Memory might require initialization before it can be accessed.
- The debugger accesses peripherals, e.g. for configuring registers prior to stopping the CPU in debug mode. Peripherals might need to be clocked and powered before they can be accessed.
- Third-party software or proprietary debuggers use the TRACE32 API (application programming interface) to access the debug port and DAP via the TRACE32 debugger hardware.

Go Resets the target via the reset line, initializes the debug port (JTAG, SWD, cJTAG), and starts the program execution. For a reset, the reset line has to be connected to the debug connector.
Program execution can, for example, be stopped by the **Break** command.

Attach

No reset happens, the mode of the core (running or halted) does not change. The debug port (JTAG, SWD, cJTAG) will be initialized.
After this command has been executed, the user program can, for example, be stopped with the **Break** command.

StandBy

Keeps the target in reset via the reset line and waits until power is detected. For a reset, the reset line has to be connected to the debug connector.

Once power has been detected, the debugger restores as many debug registers as possible (e.g. on-chip breakpoints, vector catch events, trace control) and releases the CPU from reset to start the program execution.

When a CPU power-down is detected, the debugger switches automatically back to the **StandBy** mode. This allows debugging of a power cycle because debug registers will be restored on power-up.

NOTE: Usually only on-chip breakpoints and vector catch events can be set while the CPU is running. To set a software breakpoint, the CPU has to be stopped.

Up

Resets the target via the reset line, initializes the debug port (JTAG, SWD, cJTAG), stops the CPU, and enters debug mode.
For a reset, the reset line has to be connected to the debug connector.
The current state of all registers is read from the CPU.

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**.

Format: **SYStem.CONFIG** *<parameter>*
SYStem.MultiCore *<parameter>* (deprecated)

<parameter>: **(DebugPort)**
CJTAGFLAGS *<flags>* (C7000 only)
CONNECTOR [**MIPI34** | **MIPI20T**] (C7000 only)
CORE *<core>* *<chip>*
CoreNumber *<number>*
DEBUGPORT [**DebugCable0** | **DebugCableA** | **DebugCableB**]
DEBUGPORTTYPE [**JTAG** | **SWD** | **CJTAG**]
Slave [**ON** | **OFF**]
SWDP [**ON** | **OFF**] (C7000 only)
SWDPIdleHigh [**ON** | **OFF**]
SWDPTargetSel *<value>*
TriState [**ON** | **OFF**]

<parameter>: (JTAG cont.)
DAPDRPOST *<bits>*
DAPDRPRE *<bits>*
DAPIRPOST *<bits>*
DAPIRPRE *<bits>*

DRPOST *<bits>*
DRPRE *<bits>*

ETBDRPOST *<bits>* (C5000 only)
ETBDRPRE *<bits>* (C5000 only)
ETBIRPOST*<bits>* (C5000 only)
ETBIRPRE *<bits>* (C5000 only)

<parameter>: (JTAG cont.)
IRPOST*<bits>*
IRPRE *<bits>*

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

<parameter>: **(MultiTap)**
DAPTAP *<tap>*
DEBUGTAP *<tap>*
ETBTAP *<tap>* (C5000 only)
MULTITAP [**NONE** | **IcepickA** | **IcepickB** | **IcepickC** | **IcepickD** | **IcepickBB** | **IcepickBC** | **IcepickCC** | **IcepickDD** | **JtagSEQuence** *<sub_cmd>*]
NJCR *<tap>*
SLAVETAP *<tap>*

<parameter>: **(AccessPorts)**	**AHBAPn.Base** *<address>* **AHBAPn.HPROT** [*<value>* | *<name>*] **AHBAPn.Port** *<port>* **AHBAPn.RESet** **AHBAPn.view** **AHBAPn.XtorName** *<name>* **APBAPn.Base** *<address>* **APBAPn.Port** *<port>* **APBAPn.RESet** **APBAPn.view** **APBAPn.XtorName** *<name>* **AXIAPn.ACEEnable** [**ON** | **OFF**] **AXIAPn.Base** *<address>* **AXIAPn.CacheFlags** *<value>* **AXIAPn.HPROT** [*<value>* | *<name>*] **AXIAPn.Port** *<port>* **AXIAPn.RESet** **AXIAPn.view** **AXIAPn.XtorName** *<name>* **DEBUGAPn.Port** *<port>* **DEBUGAPn.RESet** **DEBUGAPn.view** **DEBUGAPn.XtorName** *<name>* **JTAGAPn.Base** *<address>* **JTAGAPn.Port** *<port>* **JTAGAPn.CorePort** *<port>* **JTAGAPn.RESet** **JTAGAPn.view** **JTAGAPn.XtorName** *<name>*
<parameter>: (AccessPorts cont.)	**MEMORYAPn.HPROT** [*<value>* | *<name>*] **MEMORYAPn.Port** *<port>* **MEMORYAPn.RESet** **MEMORYAPn.view** **MEMORYAPn.XtorName** *<name>*
<parameter>: **(COmponents)**	**ADTF.Base** *<address>* **ADTF.RESet** **ADTF.Type** [**NONE** | **ADTF** | **ADTF2** | **GEM**] **ADTF.view** **AET.Base** *<address>* (C5000, C6000, C7000 only) **AET.RESet** (C5000, C6000, C7000 only) **AET.view** (C5000, C6000, C7000 only)

<parameter>: (COmponents cont.)

CMI.Base *<address>*
CMI.RESet
CMI.TraceID *<id>*
CMI.view

COREDEBUG.Base *<address>* (C7000 only)
COREDEBUG.RESet (C7000 only)
COREDEBUG.view (C7000 only)

CTI.Base *<address>*
CTI.Config [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
CTI.RESet
CTI.view

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

EPM.Base *<address>*
EPM.RESet
EPM.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

<parameter>: (COmponents cont.)

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

OCP.Base *<address>*
OCP.RESet
OCP.TraceID *<id>*
OCP.view

PMI.Base *<address>*
PMI.RESet
PMI.TraceID *<id>*
PMI.view

<parameter>: (Components cont.)

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

SC.Base *<address>*
SC.RESet
SC.TraceID *<id>*
SC.view

STM.Base *<address>*
STM.Mode [**None** | **SDTI** | **STP** | **STP64** | **STPv2** | **STPv2LE**]
STM.Name *<string>*
STM.RESet
STM.Type [**None** | **GenericARM** | **SDTI** | **TI**]
STM.view

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

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

<parameter>: (Components cont.)

TRACEPORT.Name
TRACEPORT.RESet
TRACEPORT.TraceSource
TRACEPORT.Type
TRACEPORT.view

TRC.Base *<address>* (C7000 only)
TRC.RESet (C7000 only)
TRC.view (C7000 only)

<parameter>: **(Deprecated)**

COREBASE *<address>*
CTIBASE *<address>*
DEBUGBASE *<address>*
ETBBASE *<address>*
ETBFUNNELBASE *<address>*
ETFBASE *<address>*
ETMBASE *<address>*

<parameter>: (Deprecated cont.)	**FUNNEL2BASE** *<address>* **FUNNELBASE** *<address>* **HTMBASE** *<address>* **ITMBASE** *<address>* **RTPBASE** *<address>* **SDTIBASE** *<address>* **STMBASE** *<address>* **TIADTFBASE** *<address>* **TIDRMBASE** *<address>* **TIEPMBASE** *<address>* **TIOCPBASE** *<address>* **TIOCPTYPE** *<type>* **TIPMIBASE** *<address>* **TISCBASE** *<address>* **TISTMBASE** *<address>* **TPIUBASE** *<address>* **TPIUFUNNELBASE** *<address>* **TRACEETBFUNNELPORT** *<port>* **TRACEFUNNELPORT***<port>* **TRACETPIUFUNNELPORT** *<port>* **view** **AHBACCESSPORT** *<port>* **APBACCESSPORT** *<port>* **AXIACCESSPORT** *<port>* **COREJTAGPORT** *<port>* **DEBUGACCESSPORT** *<port>* **JTAGACCESSPORT** *<port>* **MEMORYACCESSPORT** *<port>*

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

This is a common description of the **SYStem.CONFIG** command group for the TI C2000, C5000, C6000 and C7000 DSPs. Each debugger will provide only a subset of these commands. Some commands need a certain CPU type selection (**SYStem.CPU** *<type>*) to become active and it might additionally depend on further settings.

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

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

CJTAGFLAGS *<flags>*

Activates bug fixes for "cJTAG" implementations.
Bit 0: Disable scanning of cJTAG ID.
Bit 1: Target has no "keeper".
Bit 2: Inverted meaning of SREDGE register.
Bit 3: Old command opcodes.
Bit 4: Unlock cJTAG via APFC register.

Default: 0

CONNECTOR
[**MIPI34** | **MIPI20T**]

Specifies the connector "MIPI34" or "MIPI20T" on the target. This is mainly needed in order to notify the trace pin location.

Default: MIPI34 if CombiProbe is used, MIPI20T if µTrace (MicroTrace) is used.

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,...).

CoreNumber *<number>*

Number of cores to be considered in an SMP (symmetric multiprocessing) debug session. There are core types which can be used as a single core processor or as a scalable multicore processor of the same type. If you intend to debug more than one such core in an SMP debug session you need to specify the number of cores you intend to debug.

Default: 1.

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.

What is NIDnT?

NIDnT is an acronym for “Narrow Interface for Debug and Test”. NIDnT is a standard from the MIPI Alliance, which defines how to reuse the pins of an existing interface (like for example a microSD card interface) as a debug and test interface.

To support the NIDnT standard in different implementations, TRACE32 has several special options:

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

If several debuggers share the same debug port, all except one must have this option active.

JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET). The other debuggers need to have the setting **Slave ON**.

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

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

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

Default: OFF.

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

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

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

Default: OFF.

SWDPTargetSel *<value>*

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

Default: none set (any address accepted).

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

TriState has to be used if several debug cables are connected to a common JTAG port. TAPState and TCKLevel define the TAP state and TCK level which is selected when the debugger switches to tristate mode.
Please note:

- nTRST must have a pull-up resistor on the target.
- TCK can have a pull-up or pull-down resistor.
- Other trigger inputs need to be kept in inactive state.

Default: OFF.

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

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

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

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

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

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

If several debuggers share the same debug port, all except one must have this option active.

JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET). The other debuggers need to have the setting **Slave OFF**.

Default: OFF.
Default: ON if `CORE=... >1` in the configuration file (e.g. config.t32).
For CortexM: Please check also
SYStem.Option.DISableSOFTRES [ON | OFF]

TAPState *<state>*

This is the state of the TAP controller when the debugger switches to tristate mode. All states of the JTAG TAP controller are selectable.

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

Default: 7 = Select-DR-Scan.

TCKLevel *<level>*

Level of TCK signal when all debuggers are tristated. Normally defined by a pull-up or pull-down resistor on the target.

Default: 0.

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.

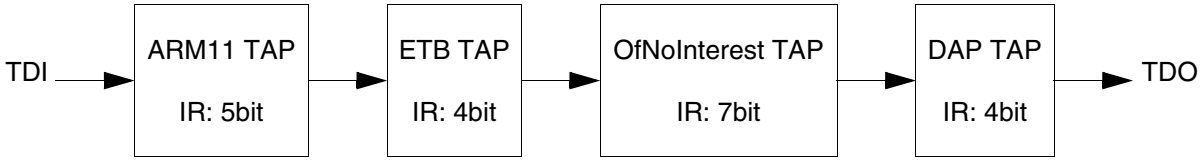
ETB (Embedded Trace Buffer) TAP if the ETB has its own TAP to access its control register (typical with ARM11 cores).
-> ETBDRPOST, ETBDRPRE, ETBIRPOST, ETBIRPRE.

NEXT: If a memory access changes the JTAG chain and the core TAP position then you can specify the new values with the NEXT... parameter. After the access for example the parameter NEXTIRPRE will replace the IRPRE value and NEXTIRPRE becomes 0. Available only on ARM11 debugger.
-> NEXTDRPOST, NEXTDRPRE, NEXTIRPOST, NEXTIRPRE.

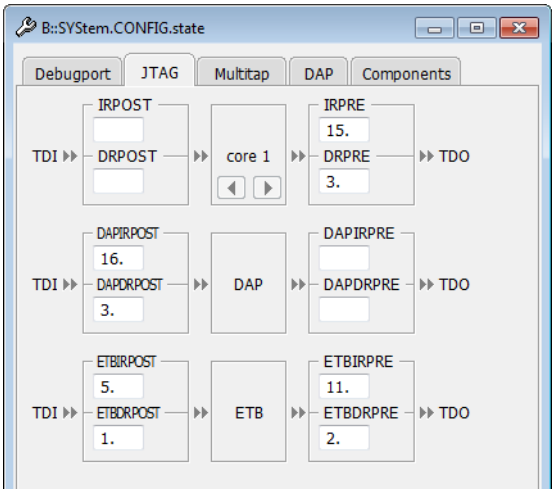
RTP (RAM Trace Port) TAP if the RTP has its own TAP to access its control register.
-> RTPDRPOST, RTPDRPRE, RTPIRPOST, RTPIRPRE.

CHIP: Definition of a TAP or TAP sequence in a scan chain that needs a different Instruction Register (IR) and Data Register (DR) pattern than the default BYPASS (1...1) pattern.
-> CHIPDRPOST, CHIPDRPRE, CHIPIRPOST, CHIPIRPRE.

Example:




```
SYStem.CONFIG IRPRE 15.
SYStem.CONFIG DRPRE  3.
SYStem.CONFIG DAPIRPOST 16.
SYStem.CONFIG DAPDRPOST  3.
SYStem.CONFIG ETBIRPOST  5.
SYStem.CONFIG ETBDRPOST  1.
SYStem.CONFIG ETBIRPRE  11.
SYStem.CONFIG ETBDRPRE   2.
```

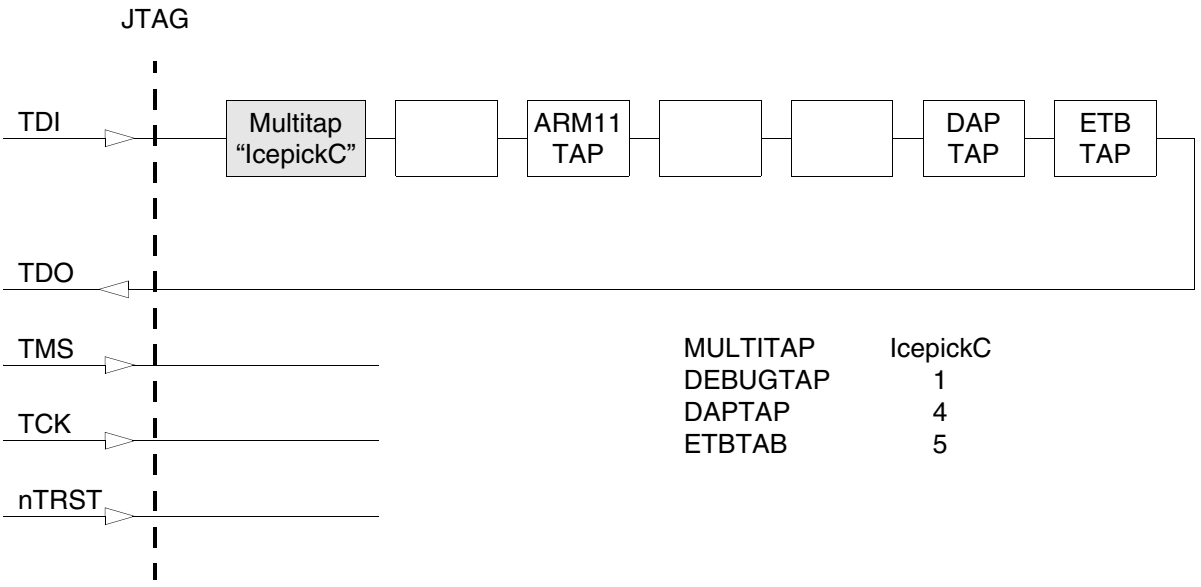

B::SYStem.CONFIG.state
Debugport
JTAG
Multitap
DAP
Components
TDI
IRPOST
DRPOST
core 1
IRPRE
15.
DRPRE
3.
TDO
TDI
DAPIRPOST
16.
DAPDRPOST
3.
DAP
DAPIRPRE
DAPDRPRE
TDO
TDI
ETBIRPOST
5.
ETBDRPOST
1.
ETB
ETBIRPRE
11.
ETBDRPRE
2.
TDO

<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.

At the moment the debugger supports three types and its different versions:
Icepickx, STCLTAPx, MSMTAP:

Example:



DAPTAP *<tap>*

Specifies the TAP number which needs to be activated to get the DAP TAP in the JTAG chain.

Used if MULTITAP=Icepickx.

DEBUGTAP *<tap>*

Specifies the TAP number which needs to be activated to get the core TAP in the JTAG chain. E.g. ARM11 TAP if you intend to debug an ARM11.

Used if MULTITAP=Icepickx.

ETBTAP *<tap>*

Specifies the TAP number which needs to be activated to get the ETB TAP in the JTAG chain.

MULTITAP
[**NONE** | **IcepickA** | **IcepickB** | **IcepickC** | **IcepickD** | **IcepickM** | **IcepickBB** | **IcepickBC** | **IcepickCC** | **IcepickDD** | **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**.

NJCR *<tap>*

Number of a Non-JTAG Control Register (NJCR) which shall be used by the debugger.

Used if MULTITAP=Icepickx.

SLAVETAP *<tap>*

Specifies the TAP number to get the Icepick of the sub-system in the JTAG scan chain.

Used if MULTITAP=IcepickXY (two Icepicks).

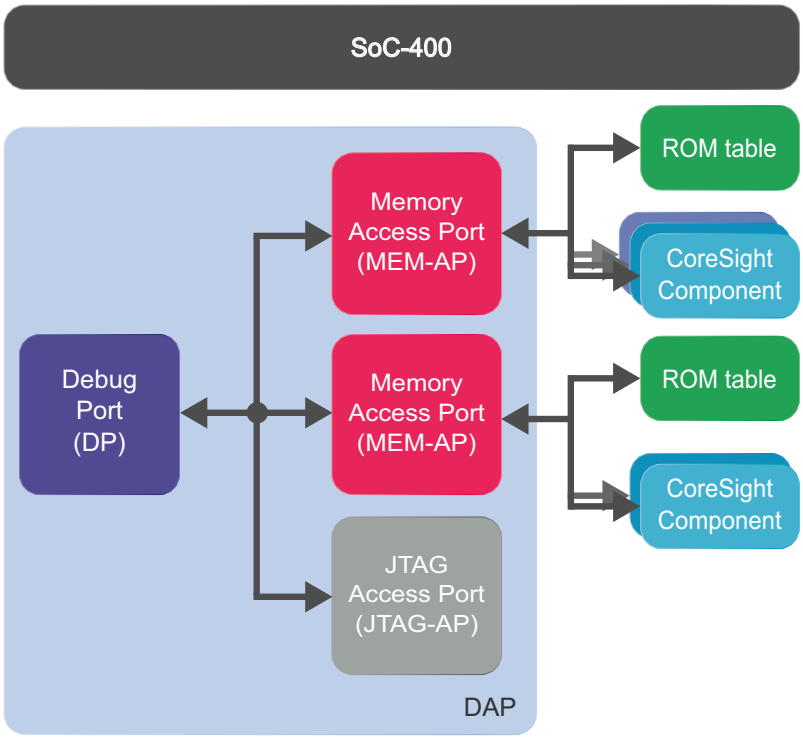
An Access Port (AP) is a CoreSight module from Arm which provides access via its debug link (JTAG, cJTAG, SWD, SWD, USB, UDP/TCP-IP, GTL, PCIe...) to:

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).

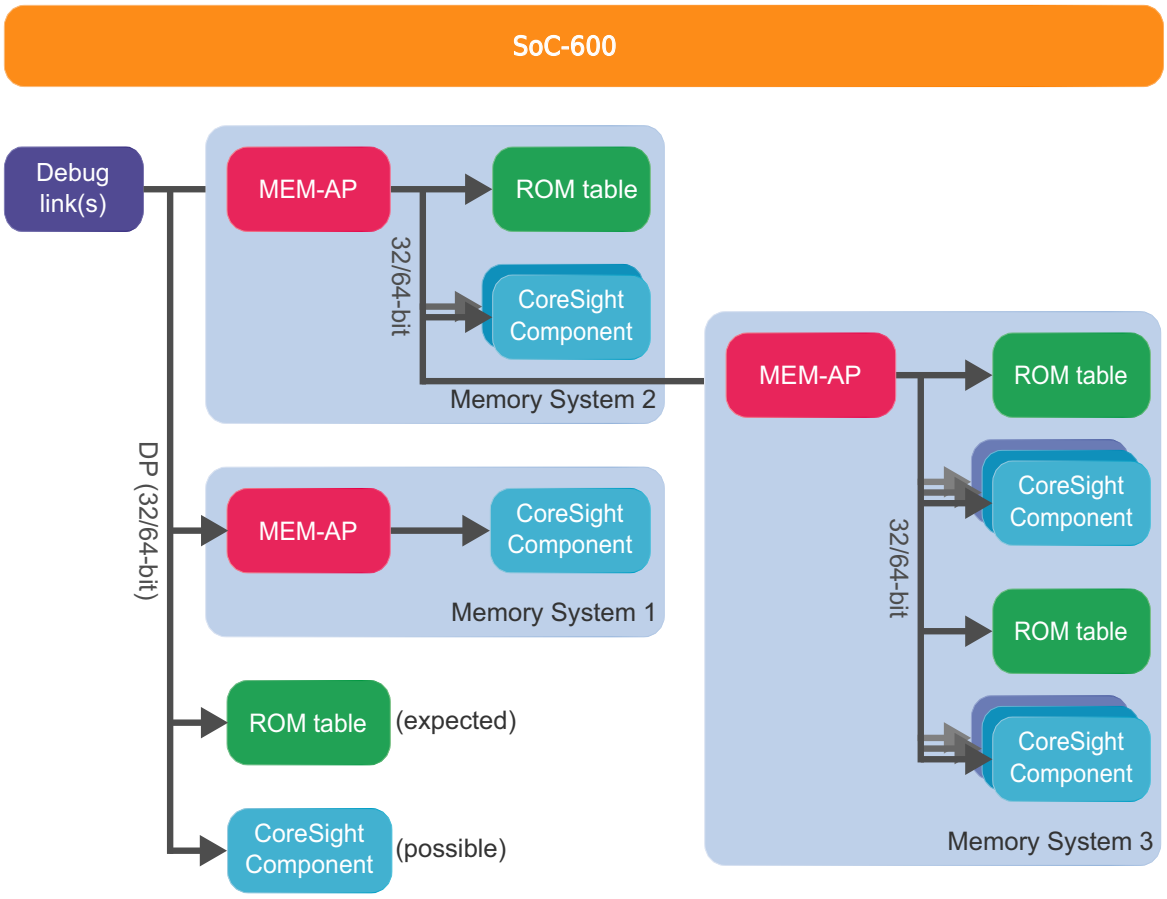
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number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

DEBUGAPn.Port *<port>*
DEBUGACCESSPORT *<port>* (deprecated)

AP access port number (0-255) of a SoC-400 system where the debug register can be found (typically on APB). Used for “DAP:” access class. Default: *<port>*=1.

JTAGAPn.CorePort *<port>*
COREJTAGPORT *<port>* (deprecated)

JTAG-AP port number (0-7) connected to the core which shall be debugged.

JTAGAPn.Port *<port>*
JTAGACCESSPORT *<port>* (deprecated)

Access port number (0-255) of a SoC-400 system of the JTAG Access Port.

MEMORYAPn.Port *<port>*
MEMORYACCESSPORT *<port>* (deprecated)

AP access port number (0-255) of a SoC-400 system where system memory can be accessed even during runtime (typically an AHB). Used for “E:” access class while running, assuming “**SYStem.MemAccess DAP**”. Default: *<port>*=0.

SoC-600 Specific Commands

AHBAPn.Base *<address>*

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

Example: SYStem.CONFIG.AHBAP1.Base DP:0x80002000
Meaning: The control register block of the AHB access ports starts at address 0x80002000.

APBAPn.Base *<address>*

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

Example: SYStem.CONFIG.APBAP1.Base DP:0x80003000
Meaning: The control register block of the APB access ports starts at address 0x80003000.

AXIAPn.Base *<address>*

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

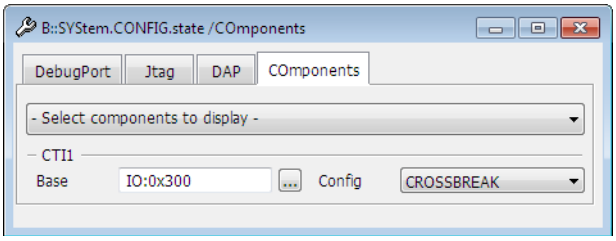
Example: SYStem.CONFIG.AXIAP1.Base DP:0x80004000
Meaning: The control register block of the AXI access ports starts at address 0x80004000.

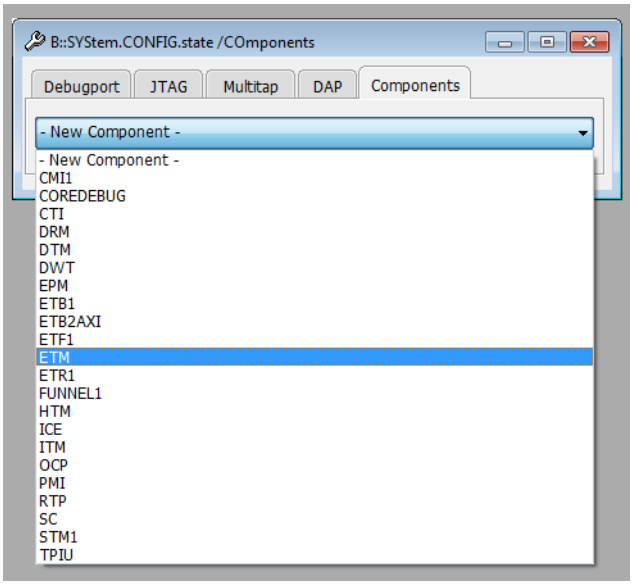
JTAGAPn.Base *<address>*

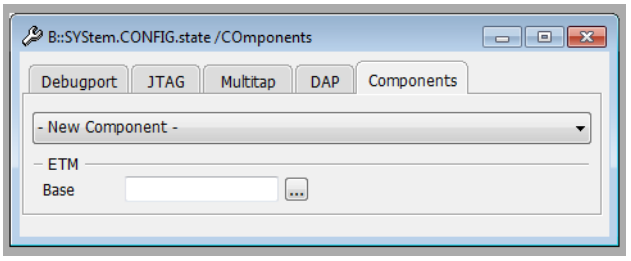
This command informs the debugger about the start address of the register block of the “JTAGAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus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

Example: SYStem.CONFIG.JTAGAP1.Base DP:0x80005000
Meaning: The control register block of the JTAG access ports starts at address 0x80005000.

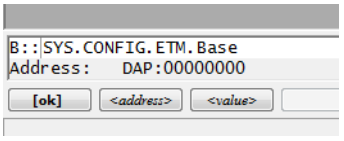
On the **Components** tab in the **SYStem.CONFIG.state** window, you can comfortably add the debug and trace components your chip includes and which you intend to use with the debugger's help.







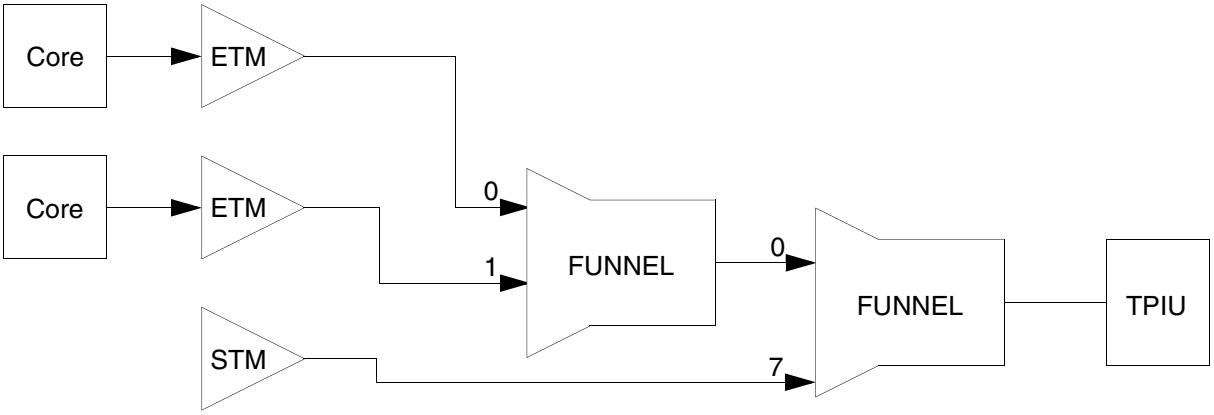
Each configuration can be done by a command in a script file as well. Then you do not need to enter everything again on the next debug session. If you press the button with the three dots you get the corresponding command in the command line where you can view and maybe copy it into a script file.



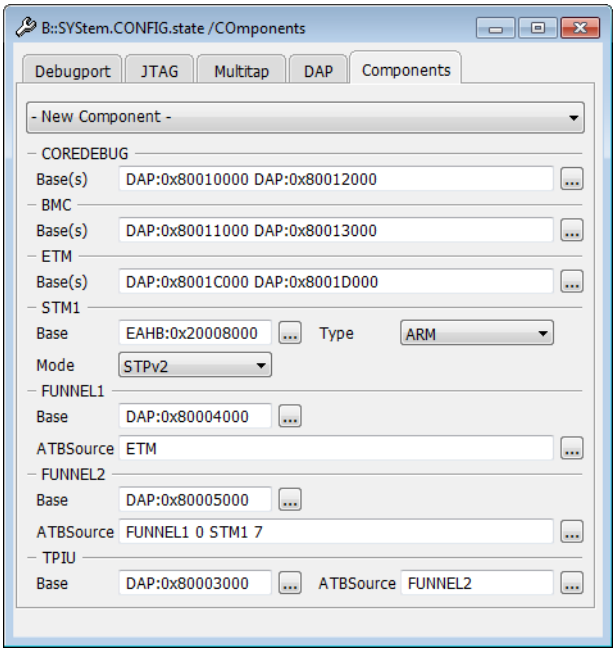
You can have several of the following components: CMI, ETB, ETF, ETR, FUNNEL, STM.
Example: FUNNEL1, FUNNEL2, FUNNEL3,...

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.

Example:




```
SYStem.CONFIG.COREDEBUG.Base 0x80010000 0x80012000
SYStem.CONFIG.BMC.Base 0x80011000 0x80013000
SYStem.CONFIG.ETM.Base 0x8001c000 0x8001d000
SYStem.CONFIG.STM1.Base EAHB:0x20008000
SYStem.CONFIG.STM1.Type ARM
SYStem.CONFIG.STM1.Mode STPv2
SYStem.CONFIG.FUNNEL1.Base 0x80004000
SYStem.CONFIG.FUNNEL2.Base 0x80005000
SYStem.CONFIG.TPIU.Base 0x80003000
SYStem.CONFIG.FUNNEL1.ATBSource ETM.0 0 ETM.1 1
SYStem.CONFIG.FUNNEL2.ATBSource FUNNEL1 0 STM1 7
SYStem.CONFIG.TPIU.ATBSource FUNNEL2
```



… **.ATBSource** *<source>*

Specify for components collecting trace information from where the trace data are coming from. This way you inform the debugger about the interconnection of different trace components on a common trace bus.

You need to specify the "... .Base *<address>*" or other attributes that define the amount of existing peripheral modules before you can describe the interconnection by "... .ATBSource *<source>*".

A CoreSight trace FUNNEL has eight input ports (port 0-7) to combine the data of various trace sources to a common trace stream. Therefore you can enter instead of a single source a list of sources and input port numbers.

Example:
SYStem.CONFIG FUNNEL.ATBSource ETM 0 HTM 1 STM 7

Meaning: The funnel gets trace data from ETM on port 0, from HTM on port 1 and from STM on port 7.

In an SMP (Symmetric MultiProcessing) debug session where you used a list of base addresses to specify one component per core you need to indicate which component in the list is meant:

Example: Four cores with ETM modules.
SYStem.CONFIG ETM.Base 0x1000 0x2000 0x3000 0x4000
SYStem.CONFIG FUNNEL1.ATBSource ETM.0 0 ETM.1 1 ETM.2 2 ETM.3 3
"...2" of "ETM.2" indicates it is the third ETM module which has the base address 0x3000. The indices of a list are 0, 1, 2, 3,...
If the numbering is accelerating, starting from 0, without gaps, like the example above then you can shorten it to
SYStem.CONFIG FUNNEL1.ATBSource ETM

Example: Four cores, each having an ETM module and an ETB module.
SYStem.CONFIG ETM.Base 0x1000 0x2000 0x3000 0x4000
SYStem.CONFIG ETB.Base 0x5000 0x6000 0x7000 0x8000
SYStem.CONFIG ETB.ATBSource ETM.2 2
The third "ETM.2" module is connected to the third ETB. The last "2" in the command above is the index for the ETB. It is not a port number which exists only for FUNNELs.

For a list of possible components including a short description see **Components and Available Commands**.

… **.BASE** *<address>*

This command informs the debugger about the start address of the register block of the component. And this way it notifies the existence of the component. An on-chip debug and trace component typically provides a control register block which needs to be accessed by the debugger to control this component.

Example: SYStem.CONFIG ETMBASE APB:0x8011c000

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

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

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

For a list of possible components including a short description see **Components and Available Commands**.

... **.Name**

The name is a freely configurable identifier to describe how many instances exists in a target systems chip. TRACE32 PowerView GUI shares with other opened PowerView GUIs settings and the state of components identified by the same name and component type. Components using different names are not shared. Other attributes as the address or the type are used when no name is configured.

Example: **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.

... **.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**.

… **.view**

Opens a window showing the current configuration of the component.

For a list of possible components including a short description see **Components and Available Commands**.

… **.TraceID** *<id>*

Identifies from which component the trace packet is coming from. Components which produce trace information (trace sources) for a common trace stream have a selectable ".TraceID *<id>*".

If you miss this SYStem.CONFIG command for a certain trace source (e.g. ETM) then there is a dedicated command group for this component where you can select the ID (ETM.TraceID *<id>*).

The default setting is typically fine because the debugger uses different default trace IDs for different components.

For a list of possible components including a short description see **Components and Available Commands**.

CTI.Config *<type>*

Informs about the interconnection of the core Cross Trigger Interfaces (CTI). Certain ways of interconnection are common and these are supported by the debugger e.g. to cause a synchronous halt of multiple cores.

NONE: The CTI is not used by the debugger.
ARMV1: This mode is used for ARM7/9/11 cores which support synchronous halt, only.
ARMPostInit: Like ARMV1 but the CTI connection differs from the ARM recommendation.
OMAP3: This mode is not yet used.
TMS570: Used for a certain CTI connection used on a TMS570 derivative.
CortexV1: The CTI will be configured for synchronous start and stop via CTI. It assumes the connection of DBGRQ, DBGACK, DBGRESTART signals to CTI are done as recommended by ARM. The CTIBASE must be notified. “CortexV1” is the default value if a Cortex-A/R core is selected and the CTIBASE is notified.
QV1: This mode is not yet used.

ARMV8V1: Channel 0 and 1 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only.
ARMV8V2: Channel 2 and 3 of the CTM are used to distribute start/stop events from and to the CTIs. ARMv8 only.
ARMV8V3: Channel 0, 1 and 2 of the CTM are used to distribute start/stop events. Implemented on request. ARMv8 only.

ETB.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.

ETB.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.

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.
OCP.Type *<type>*	Specifies the type of the OCP module. The *<type>* is just a number which you need to figure out in the chip documentation.
RTP.PerBase *<address>*	PERBASE specifies the base address of the core peripheral registers which accesses shall be traced. PERBASE is needed for the RAM Trace Port (RTP) which is available on some derivatives from Texas Instruments. The trace packages include only relative addresses to PERBASE and RAMBASE.
RTP.RamBase *<address>*	RAMBASE is the start address of RAM which accesses shall be traced. RAMBASE is needed for the RAM Trace Port (RTP) which is available on some derivatives from Texas Instruments. The trace packages include only relative addresses to PERBASE and RAMBASE.
STM.Mode [**NONE** | **XTIv2** | **SDTI** | **STP** | **STP64** | **STPv2**]	Selects the protocol type used by the System Trace Module (STM).
STM.Type [**None** | **Generic** | **ARM** | **SDTI** | **TI**]	Selects the type of the System Trace Module (STM). Some types allow to work with different protocols (see STM.Mode).
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.

ADTF.Base *<address>*
ADTF.RESet
ADTF.Type [None | ADTF | ADTF2 | GEM]

AMBA trace bus DSP Trace Formatter (ADTF) - Texas Instruments
Module of a TMS320C5x or TMS320C6x core converting program and data trace information in ARM CoreSight compliant format.

CMI.Base *<address>*
CMI.RESet
CMI.TraceID *<id>*
Clock Management Instrumentation (CMI) - Texas Instruments
Trace source delivering information about clock status and events to a system trace module.

DRM.Base *<address>*
DRM.RESet
Debug Resource Manager (DRM) - Texas Instruments
It will be used to prepare chip pins for trace output.

EPM.Base *<address>*
EPM.RESet
Emulation Pin Manager (EPM) - Texas Instruments
It will be used to prepare chip pins for trace output.

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.

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.

OCP.Base *<address>*
OCP.RESet
OCP.TraceID *<id>*
OCP.Type *<type>*
Open Core Protocol watchpoint unit (OCP) - Texas Instruments
Trace source module delivering bus trace information to a system trace module.

PMI.Base *<address>*
PMI.RESet
PMI.TraceID *<id>*
Power Management Instrumentation (PMI) - Texas Instruments
Trace source reporting power management events to a system trace module.

SC.Base *<address>*
SC.RESet
SC.TraceID *<id>*
Statistic Collector (SC) - Texas Instruments
Trace source delivering statistic data about bus traffic to a system trace module.

STM.Base *<address>*
STM.Mode [**NONE** | **XTIv2** | **SDTI** | **STP** | **STP64** | **STPv2**]
STM.RESet
STM.Type [**None** | **Generic** | **ARM** | **SDTI** | **TI**]
System Trace Macrocell (STM) - MIPI, ARM CoreSight, others
Trace source delivering system trace information e.g. sent by software in printf() style.

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.

Example: SYStem.CONFIG ETMBASE APB:0x8011c000

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

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

Example assuming four cores: "SYStem.CONFIG COREBASE 0x80001000 0x80003000 0x80005000 0x80007000".

COREBASE (old syntax: DEBUGBASE): Some cores e.g. Cortex-A or Cortex-R do not have a fix location for their debug register which are used for example to halt and start the core. In this case it is essential to specify its location before you can connect by e.g. **SYStem.Up**.

PERBASE and RAMBASE are needed for the RAM Trace Port (RTP) which is available on some derivatives from Texas Instruments. PERBASE specifies the base address of the core peripheral registers which accesses shall be traced, RAMBASE is the start address of RAM which accesses shall be traced. The trace packages include only relative addresses to PERBASE and RAMBASE.

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.

Example: SYStem.CONFIG STMFUNNEL2PORT 3

Meaning: The System Trace Module (STM) is connected to input port #3 on FUNNEL2.

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

In case there are dedicated funnels for the ETB and the TPIU their base addresses are specified by ETBFUNNELBASE, TPIUFUNNELBASE respectively. And the funnel port number for the ETM are declared by ETMETBFUNNELPORT, ETMTPIUFUNNELPORT respectively.

TRACE... stands for the ADTF trace source module.

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.

TIOCPTYPE *<type>*

Specifies the type of the OCP module from Texas Instruments (TI).

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)
BMCBASE *<address>*	**BMC.Base** *<address>*
BYPASS *<seq>*	**CHIPIRPRE** *<bits>* **CHIPIRLENGTH** *<bits>* **CHIPIRPATTERN.Alternate** *<pattern>*
COREBASE *<address>*	**COREDEBUG.Base** *<address>*
CTIBASE *<address>*	**CTI.Base** *<address>*
DEBUGBASE *<address>*	**COREDEBUG.Base** *<address>*
DTMCONFIG [**ON** | **OFF**]	**DTM.Type.Generic**
DTMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource DTM** *<port>* (1)
DTMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource DTM** *<port>* (1)
DTMFUNNELPORT *<port>*	**FUNNEL1.ATBSource DTM** *<port>* (1)
DTMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource DTM** *<port>* (1)
DWTBASE *<address>*	**DWT.Base** *<address>*
ETB2AXIBASE *<address>*	**ETB2AXI.Base** *<address>*
ETBBASE *<address>*	**ETB1.Base** *<address>*
ETBFUNNELBASE *<address>*	**FUNNEL4.Base** *<address>*
ETFBASE *<address>*	**ETF1.Base** *<address>*
ETMBASE *<address>*	**ETM.Base** *<address>*
ETMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource ETM** *<port>* (1)
ETMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource ETM** *<port>* (1)
ETMFUNNELPORT *<port>*	**FUNNEL1.ATBSource ETM** *<port>* (1)
ETMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource ETM** *<port>* (1)
FILLDRZERO [**ON** | **OFF**]	**CHIPDRPRE** 0 **CHIPDRPOST** 0 **CHIPDRLENGTH** *<bits_of_complete_dr_path>* **CHIPDRPATTERN.Alternate** 0
FUNNEL2BASE *<address>*	**FUNNEL2.Base** *<address>*
FUNNELBASE *<address>*	**FUNNEL1.Base** *<address>*
HSMBASE *<address>*	**HSM.Base** *<address>*

HTMBASE *<address>*	**HTM.Base** *<address>*
HTMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource HTM** *<port>* (1)
HTMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource HTM** *<port>* (1)
HTMFUNNELPORT *<port>*	**FUNNEL1.ATBSource HTM** *<port>* (1)
HTMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource HTM** *<port>* (1)
ITMBASE *<address>*	**ITM.Base** *<address>*
ITMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource ITM** *<port>* (1)
ITMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource ITM** *<port>* (1)
ITMFUNNELPORT *<port>*	**FUNNEL1.ATBSource ITM** *<port>* (1)
ITMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource ITM** *<port>* (1)
PERBASE *<address>*	**RTP.PerBase** *<address>*
RAMBASE *<address>*	**RTP.RamBase** *<address>*
RTPBASE *<address>*	**RTP.Base** *<address>*
SDTIBASE *<address>*	**STM1.Base** *<address>* **STM1.Mode SDTI** **STM1.Type SDTI**
STMBASE *<address>*	**STM1.Base** *<address>* **STM1.Mode STPV2** **STM1.Type ARM**
STMETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource STM1** *<port>* (1)
STMFUNNEL2PORT *<port>*	**FUNNEL2.ATBSource STM1** *<port>* (1)
STMFUNNELPORT *<port>*	**FUNNEL1.ATBSource STM1** *<port>* (1)
STMTPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource STM1** *<port>* (1)
TIADTFBASE *<address>*	**ADTF.Base** *<address>*
TIDRMBASE *<address>*	**DRM.Base** *<address>*
TIEPMBASE *<address>*	**EPM.Base** *<address>*
TIICEBASE *<address>*	**ICE.Base** *<address>*
TIOCPBASE *<address>*	**OCP.Base** *<address>*
TIOCPTYPE *<type>*	**OCP.Type** *<type>*
TIPMIBASE *<address>*	**PMI.Base** *<address>*
TISCBASE *<address>*	**SC.Base** *<address>*
TISTMBASE *<address>*	**STM1.Base** *<address>* **STM1.Mode STP** **STM1.Type TI**
TPIUBASE *<address>*	**TPIU.Base** *<address>*
TPIUFUNNELBASE *<address>*	**FUNNEL3.Base** *<address>*
TRACEETBFUNNELPORT *<port>*	**FUNNEL4.ATBSource ADTF** *<port>* (1)

TRACEFUNNELPORT *<port>*	**FUNNEL1.ATBSource ADTF** *<port>* (1)
TRACETPIUFUNNELPORT *<port>*	**FUNNEL3.ATBSource ADTF** *<port>* (1)
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.Option.AHBHPROT Select AHB-AP HPROT bits

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

Default: 0

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

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

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

Default: OFF.

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

SYStem.Option.AXICACHEFLAGS Configure AXI-AP cache bits

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

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

This option configures the value used for the Cache and Domain bits in the Control Status Word (CSW[27:24]->Cache, CSW[14:13]->Domain) of an AXI Access Port of a DAP, when using the AXI: memory class.

SYStem.Option.AXIHPROT Select AXI-AP HPROT bits

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

Default: 0

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

SYStem.Option.ByteMode Define byte mode

Format: **SYStem.Option.ByteMode** [**AUTO** | **ACCESS** | **WORD** | **BYTE**]

Default: AUTO.

Defines byte mode.

AUTO	Byte mode is automatically detected by TRACE32.
ACCESS	The selected byte mode depends on the CPU registers.
WORD	TRACE32 interprets code in word-pointer mode.
BYTE	TRACE32 interprets code in byte-pointer mode.

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.DAPSYSPWRUPREQ Force system power in DAP

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

Default: ON.

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

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

SYStem.Option.DAPREMAP Rearrange DAP memory map

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

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

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

SYStem.Option.DEBUGPORTOptions Options for debug port handling

Format: **SYStem.Option.DEBUGPORTOptions** *<option>*

<option>: **SWITCHTOSWD.**[**TryAll** | **None** | **JtagToSwd** | **LuminaryJtagToSwd** | **DormantToSwd** | **JtagToDormantToSwd**]
SWDTRSTKEEP.[**DEFault** | **LOW** | **HIGH**]

Default: SWITCHTOSWD.TryAll, SWDTRSTKEEP.DEFault.

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

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

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

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

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

SYStem.Option.DAPNOIRCHECK No DAP instruction register check

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

Default: OFF.

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

SYStem.Option.DUALPORT Implicitly use run-time memory access

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

All TRACE32 windows that display memory are updated while the processor is executing code (e.g. **Data.dump**, **Data.List**, **PER.view**, **Var.View**). This setting has no effect if **SYStem.MemAccess** is disabled.

If only selected memory windows should update their content during runtime, leave **SYStem.Option.DUALPORT OFF** and use the access class prefix **E** or the format option **%E** for the specific windows.

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.LOCK Tristate the JTAG port

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 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. Please note: nTRST must have a pull-up resistor on the target, EDBGRQ must have a pull-down resistor.

SYStem.Option.EnTRST Control TAP reset

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

Default: ON.

To set the debug interface in a defined state the TAP is reset by driving the TRST pin low and additionally holding TMS low for five 5 TCKs. By setting the EnTRST option to OFF only the TMS method is used. The reason for introducing this command was that in some target systems several chips were connected to the TRST line, which must not be reset together with the debug TAP.

SYStem.Option.INTDIS Disable all interrupts

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

Default: OFF.

If this option is ON, all interrupts on the core are disabled.

SYStem.Option.MUHP High-priority memory access

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

Default: OFF.

Format: **SYStem.Option.OVERLAY** [**ON** | **OFF** | **WithOVS**]

Default: OFF.

ON Activates the overlay extension and extends the address scheme of the debugger with a 16 bit virtual overlay ID. Addresses therefore have the format *<overlay_id>*:*<address>*. This enables the debugger to handle overlaid program memory.

OFF Disables support for code overlays.

WithOVS Like option **ON**, but also enables support for software breakpoints. This means that TRACE32 writes software breakpoint opcodes to both, the *execution area* (for active overlays) and the *storage area*. This way, it is possible to set breakpoints into inactive overlays. Upon activation of the overlay, the target's runtime mechanisms copies the breakpoint opcodes to the execution area. For using this option, the storage area must be readable and writable for the debugger.

Example:

```
SYStem.Option.OVERLAY ON
Data.List 0x2:0x11c4                    ; Data.List <overlay_id>:<address>
```

SYStem.Option.PWRDWN Allow power-down mode

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

Default: OFF.

If this option is OFF, the debugger forces the chip to keep clock and keep power on OMAPxxxx devices.

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

Default: OFF.

This option forces the debugger to use the Target Server from Texas Instruments for all activities. It avoids accelerated procedures and activates a more powerful error handling. This option should be used if the debugger shows any unstable behavior.

SYStem.Option.TURBO

Use DMA for write accesses

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

Default: OFF.

If TURBO is enabled the debugger uses a DMA channel of the DSP to write data to the DSP memory area. This option doubles the download rate. In case the user application uses the same DMA resource, the transfer will not work. Therefore we recommend to use that option for downloading data after reset at the beginning of a debug session, only.

SYStem.RESetOut

Reset the DSP

Format: **SYStem.RESetOut**

This command resets the DSP.

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

Default: OFF.

When this option is enabled, the TI's cTools trace decoder software is used instead of LAUTERBACH's trace decoder software. We recommend not to activate this option since the LAUTERBACH trace decoder software is optimized for TRACE32 software architecture. However, TI' CToolsDecoder might be helpful for error diagnostics.

SYStem.Option.CtoolsNoSync CToolsNoSync

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

Default: OFF.

Some first generation chips with C55x cTools trace functionality do not generate synchronization sequences. In that very untypical case, this option has to be set to ON. Normally, this option is not required!

CPU specific BenchMarkCounter Commands

The benchmark counters can be read at run-time.

For information about *architecture-independent* **BMC** commands, refer to **"BMC"** (general_ref_b.pdf).

For information about *architecture-specific* **BMC** command(s), see command description(s) below.

BMC.<counter>.ATOB — Advise counter to count within AB-range

Format: **BMC.***<counter>***.ATOB** [**ON** | **OFF**]

Advise the counter to count the specified event only in AB-range. Alpha and Beta markers are used to specify the AB-range.

Example to measure the time used by the function sieve:

```
BMC.<counter> ClockCylces              ; <counter> counts clock cycles

BMC.CLOCK 450.Mhz                      ; core is running at 450.MHz

Break.Set sieve /Alpha                 ; set a marker Alpha to the entry
                                       ; of the function sieve

Break.Set V.END(sieve)-1 /Beta         ; set a marker Beta to the exit
                                       ; of the function sieve

BMC.<counter>.ATOB ON                  ; advise <counter> to count only
                                       ; in AB-range
```

Format: **BMC.***<counter>***.EVENT** *<event>*

Assign event to counter.

```
BMC.<counter>.EVENT ClockCycles       ; <counter> counts clock cycles

BMC.<counter> ClockCycles             ; equivalent
```

The following *<events>* are available:

OFF	Disabled
ICMISS	Instruction cache misses: Counts instructions cache misses, in relation to total instruction access.
INST	Instructions
PINST	Parallel instructions
INT	Interrupts
PNULL	Pipe protection NULL
FNULL	Instruction fetch NULL
DNULL	Data fetch NULL
DCMISS	Data cache misses: Counts data cache misses, in relation to total data access.
DCACOLL	Data cache arbitration collisions
IDLE	Idle clock cycles
D	Delta breakpoints
E	Echo breakpoints
CLOCKCYCLES	Clock cycles
TIME	TIME is measured by counting CLOCK. The translation to TIME is done by using the CPU frequency. For this reason, the CPU frequency has to be entered with the command **BMC.CLOCK**.

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.CONVert — Adjust range breakpoint in on-chip resource

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

The on-chip breakpoints can only cover specific ranges. If a range cannot be programmed into the breakpoint, it will automatically be converted into a single address breakpoint when this option is active. This is the default. Otherwise an error message is generated.

```
TrOnchip.CONVert ON
Break.Set 0x1000--0x17ff /Write       ; sets breakpoint at range
Break.Set 0x1001--0x17ff /Write       ; 1000--17ff sets single breakpoint
…                                     ; at address 1001

TrOnchip.CONVert OFF                  ; sets breakpoint at range
Break.Set 0x1000--0x17ff /Write       ; 1000--17ff
Break.Set 0x1001--0x17ff /Write       ; gives an error message
```

The **TrOnchip** command provides low-level access to the on-chip debug register.

TrOnchip.ATOB — Activate on-chip breakpoints in AB-range

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

Activate the on-chip breakpoints between the program breakpoints Alpha and Beta only.

TrOnchip.BMCTR — Configure the benchmark counter

Format: **TrOnchip.BMCTR0** | **BMCTR1** *<bmctr>*

<bmctr>:
OFF
CMISS
INST
PINST
INT
PipeNULL
FetchNULL
DataNULL
IDLE
Delta
Echo
CLOCK
TIME
Init
ATOB

Benchmark Counter - short BMCTR - collect Information about the throughput of the target processor. They count for certain events, like interrupts, cache misses or cpu cycles. This information may be helpful in finding bottlenecks and tuning the application.

OFF	Switch off the benchmark counter.
CMISS	Cache MISS counts instruction cache misses.
INST	Count executed INSTructions.
PINST	PINST counts executed Parallel INSTructions and is incremented by 1 for each parallel instruction.
INT	Count INTerrupts which occurred.
PipeNULL	Count cycles which have to be inserted, because of a pipeline conflict, e.g. a resource conflict between instructions.
FetchNULL	Count cycles which have to be inserted, because of an instruction not being available in the instruction pipeline, e.g. if the pipeline is flushed after a conditional branch or a cache miss on the instruction cache.
DataNULL	Count cycles which have to be inserted, because of data not being available (Data NULL), e.g. when reading data from a slow off-chip memory.
IDLE	If the cpu is in IDLE state, this option will count the clock cycles as long as the cpu stays in an IDLE state.
Delta	Count hits of Delta-Marker, if specified.
Echo	Count hits of Echo-Marker, if specified.
CLOCK	Incremented for each cpu clock.
TIME	TIME is measured by counting CLOCK. The translation to TIME is done by using the cpu frequency. For this reason, the cpu frequency has to be entered with the command **TrOnchip.CLOCK**.
Init	Reset the benchmark counter to zero.
ATOB	Activate the benchmark counter to count between the program marker Alpha and Beta only.

NOTE: CMISS, INST, PINS, and INT count the number of occurrences of the corresponding event. PNULL, FNULL, and DNULL add the benchmark counter by one for every NULL cycle, which was inserted.

There are two separate benchmark counters available. If only the first one is selected, it is laid out as a 40-bit counter. If both counters are active, both of them are 16 bit counters only.
Most measurements are only useful in relation to others, e.g. number of events in a given time frame. In this scenario one counter may measure the elapsed time and the second counts for the interrupts which occurred during that time frame.
In this case, both counters are used as 16 bit counters. Therefore they may overrun, shortly. For this reason, the benchmark counter can only be applied to short program sections or single functions.

To count parallel instructions, in relation to the total instructions, use the following commands:

```
TrOnchip.RESet                ; Reset the TrOnchip settings

TrOnchip.view                 ; Display the TrOnchip window

TrOnchip.BMCTR0 INST          ; Set the first (BMCTR0) benchmark counter
                              ; to count INSTructions

TrOnchip.BMCTR1 PINT          ; Set the second (BMCTR1) benchmark counter
                              ; to count parallel instructions
```

BMCTR0 is incremented by 1 for every instruction. If a parallel instruction gets to be executed, both BMCTR0/INST and BMCTR1/PINST are incremented by 1.

In order to count the number of calls of a user-defined function func13, specify as follows:

```
TrOnchip.RESet                                ; Reset the TrOnchip settings

TrOnchip.view                                 ; Display the TrOnchip window

Break.Set func13 /Program /Delta              ; Set the program marker Delta
/Onchip                                       ; to the entry of func13

TrOnchip.BMCTR0 Delta                         ; Set the first (BMCTR0)
                                              ; benchmark counter to count
                                              ; hits on the program marker
                                              ; Delta
```

The following sequence will count on write access, with value of 1, to a variable:

```
; Reset the TrOnchip settings
TrOnchip.RESet

; Set the data write marker on a write 0x01 to the variable flags[3]
Var.Break.Set flags[3] /Write /Echo /Onchip /DATA.Word 0x01

; Set the first (BMCTR0) benchmark counter to count hits on the
; marker Echo
TrOnchip.BMCTR1 Echo
```

In order to measure the time, for one pass through the function sieve:

```
TrOnchip.RESet              ; Reset the TrOnchip settings

TrOnchip.view               ; Display the TrOnchip window

TrOnchip.BMCTR0 TIME        ; Set the first (BMCTR0) benchmark counter
                            ; to measure the time

TrOnchip.CLOCK 12.MHz       ; Set the frequency of the CPU

Go sieve                    ; Go to the function sieve

TrOnchip.BMCTR0 Init        ; Initialize the benchmark counter

Go.Return                   ; Go to the last instruction of the function
                            ; sieve
```

Activate the benchmark counter to count parallel instructions in between the program marker Alpha and Beta only:

```
TrOnchip.RESet              ; Reset the TrOnchip settings

TrOnchip.view               ; Display the TrOnchip window

Break.Set sieve /Alpha      ; Set the program marker Alpha to the begin
                            ; of function sieve

Break.Set V.END(sieve)-1    ; Set the program marker Beta to the end of
/Beta                       ; function sieve

TrOnchip.BMCTR0 ATOB ON     ; Instruct the benchmark counter to count
                            ; between the program marker Alpha and Beta
                            ; only

TrOnchip.BMCTR0 PINST       ; Instruct the benchmark counter to count
                            ; for parallel instruction between Alpha and
                            ; Beta only
```

Format: **TrOnchip.CLOCK** *<value>*

In order to measure the time, as one option of the benchmark counter, clock has to be set to the frequency of the target cpu.

```
TrOnchip.CLOCK 30.MHz        ; Set the frequency of the cpu
```

TrOnchip.CoefficientAccess

AET trigger optimization

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

Default: OFF.

Supports AET trigger optimization.

TrOnchip.DualAccess

AET trigger optimization

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

Default: ON.

Supports AET trigger optimization.

TrOnchip.PROfile

Display the benchmark data

Format **TrOnchip.PROfile** [*<value>*]

Displays the collected data of the first benchmark counter BMCTR0 in a graphical representation. In order to specify a vertical scaling, use the optional *<value>* parameter.

Format: **TrOnchip.RESet**

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

Tracing

Depending on the chip the C5000 trace can either be directly output on dedicated trace pins or sent into a Coresight trace bus using an **ADTF** component. Another option is to store the trace onchip in trace buffers (e.g. ETB). For further information about Coresight component configuration please refer to **"Setup of the Debugger for a CoreSight System"** (app_arm_coresight.pdf).

For information about trace hardware setup refer to the **"AutoFocus User's Guide"** (autofocus_user.pdf).

Controlling the Trace Capture

On the C5x cores the trace capture setup is controlled by the **AET** command group.

Trace Breakpoints

The following breakpoint examples use AET resources:

```
; Broadcast only the execution of the specified instructions
Break.Set <address> | <range>  /Program /TraceEnable
; Broadcast only the instructions that perform the specified data access
Break.Set <address> | <range> /ReadWrite | /Read | /Write /TraceData
```

Broadcast only the execution of the instruction at address 0x4dd84.

```
Break.Set 0x1234ABCD /Program /TraceEnable
…
Trace.List                                      ; display the result
…
Break.Delete                                    ; delete breakpoint
```

Mechanical Description of the 20-pin Debug Cable

This connector is defined by ARM. Our debuggers “JTAG Debugger for ARM7” (LA-7746) and “JTAG Debugger for ARM9” (LA-7742) and “JTAG Debugger for TMS320” are supplied with this connector:

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

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

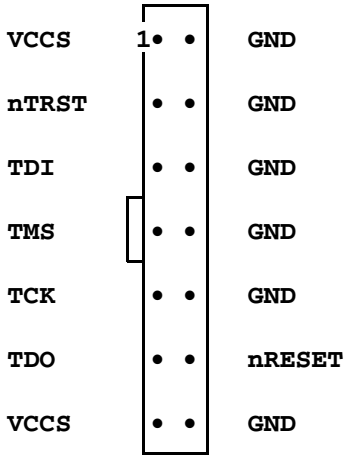
We strongly recommend to use a connector on your target with housing and having a center polarization (e.g. AMP: 2-827745-0). A connection the other way around indeed causes damage to the output driver of the debugger.

- The input and output signals are connected to a supply translating transceiver (74ALVC164245). Therefore the ICD/AICD can work in an voltage range of (1.5 V) 1.8 … 3..3 V (3.6 V). Please note that a 5 V supply environment is not supported! This would cause damage on the ICD/AICD. Please contact us for alternate solutions if you need to work with 5 V.
- VTREF is used as a sense line for the target voltage. It is also used as supply voltage for the supply translating transceiver of the ICD/AICD interface to make an adaptation to the target voltage (1.5 V) 1.8 … 3.3 V (3.6 V).
- nTRST, TDI, TMS, TCK are driven by the supply translating transceiver. 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.
- RTCK is the return test clock signal from the target JTAG port. This signal can be used to synchronize JTAG clock with the processor clock (see **SYStem.JtagClock**).
- TDO is an ICD/AICD input. It is connected to the supply translating transceiver.
- nSRST (=nRESET) 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. A 47 kΩ pull-up resistor is included in the ICD/AICD connector. The debugger will only assert a pulse on nSRST when the SYStem.UP, the SYStem.Mode Go or the SYStem.RESetOUT command is executed. If it is ensured that the DSP is able to enter debug mode every time (no hang-up condition), the nSRST line is optional.
- EDBGRQ is driven by the supply translating transceiver. This line is optional. It allows to halt the program execution by an external trigger signal.
- DBGACK is an ICD/AICD input. It is connected to the supply translating transceiver. A 47 kΩ pull-down resistor is included in the ICD/AICD connector. This line is optional. It allows exact runtime measurement and exact triggering of other devices on a program execution halt.
- N/C (= Vsupply) is not connected in the ICD/AICD. This pin is used by debuggers of other manufacturers for supply voltage input. The ICD/AICD is self-powered.

There is an additional plug in the connector on the debug cable to the debug interface. This signal is tristated if the JTAG connector is tristated by the debugger and it is pulled low otherwise. This signal is normally not required, but can be used to detect the tristate state if more than one debug tools are connected to the same JTAG port.

Mechanical Description of the 14-pin Debug Cable

This connector was defined by ARM. We used that debug cable in former times. An adapter to the 20-pin connector (see above) and vice versa is available (LA-7747: JTAG ARM Converter 14-20).



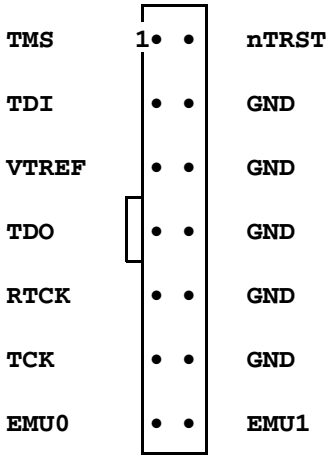
This is a standard 14 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.).

Electrical Description of the 14-pin Debug Cable

- TCK, TMS, TDI and nTRST are driven by a VHC125 driver which is supplied with VCCS. Therefore the ICD/AICD can work in an voltage range of (2.25 V) 2.5…5.0 V (5.5 V). In normal operation mode this driver is enabled, but it can be disabled to give another tool access to the JTAG port. The TMS, TDI and nTRST lines have a 47k pull-up resistor in the ICD connector. 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 and needs standard TTL level.
- VCCS is used as a sense line for the target voltage. It is also used as supply voltage for the output driver 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. A 22k pull-up resistor is included in the ICD connector. The debugger will only assert a pulse on nRESET when the SYStem.Up command is executed. If it is ensured that the DSP is able to enter debug mode every time (no hang-up condition), the nRESET line is optional.

Mechanical Description of the TI Connector

This connector is defined by Texas Instruments. It is typically used on TMS320 designs. Our debuggers are not supplied with this connector, but an adapter is available (LA-7748: JTAG ARM Converter ARM-TI).



This is a standard 14 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.).

FAQ

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

Operation Voltage

Adapter	OrderNo	Voltage Range
JTAG Debugger for C5500 (ICD)	LA-7830	1.8 .. 3.6 V