C2000 Debugger

C2000 Debugger

Version 18-Nov-2020

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Basics - Training"** (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.

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:	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.

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.

If a DEBUG INTERFACE (LA-7701) is used the trigger system has the following restrictions:

- After starting the application there is a delay until the trigger system is working. The delay depends on the host system and the JTAG frequency. It will be typically between 25 and 100 µs.
- If a terminal window is open the response time of the trigger system is undefined. It is recommended not to use the trigger system and terminal window at the same time.

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).

- **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
C28x	2 or 10 (with ERAD module)	2 single addresses	2 (only with ERAD module)	1 (only with ERAD module)

Memory Classes

The following DSP specific memory classes are available.

Memory Class	Description
P	Program Memory
D	Data Memory
VM	Virtual Memory (memory on the debug system)
E	Emulation Memory, Pseudo Dualport Access to Memory.

To access a memory class, write the class in front of the address. Prepending an E as attribute to the memory class will make memory accesses possible, even when the target CPU is running. See **SYStem.MemAccess** and **SYStem.CpuAccess** for more information.

Examples:

```
Data.dump D:0--0xff
Data.Dump ED:0x8000
Data.List EP:main
```

SYStem.CONFIG.state — Display target configuration

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

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

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

<tab>	Opens the **SYStem.CONFIG.state** window on the specified tab. For tab descriptions, see below.
DebugPort (default)	The **DebugPort** tab informs the debugger about the debug connector type and the communication protocol it shall use. For descriptions of the commands on the **DebugPort** tab, see **DebugPort**.
Jtag	The **Jtag** tab informs the debugger about the position of the Test Access Ports (TAP) in the JTAG chain which the debugger needs to talk to in order to access the debug and trace facilities on the chip. For descriptions of the commands on the **Jtag** tab, see **Jtag**.
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**.

DAP	The **DAP** tab informs the debugger about an ARM CoreSight Debug Access Port (DAP) and about how to control the DAP to access chip-internal memory busses (AHB, APB, AXI) or chip-internal JTAG interfaces. For descriptions of the commands on the **DAP** tab, see **DAP**.
COmponents	The **COmponents** tab informs the debugger (a) about the existence and interconnection of on-chip CoreSight debug and trace modules and (b) informs the debugger on which memory bus and at which base address the debugger can find the control registers of the modules. For descriptions of the commands on the **COmponents** tab, see **COmponents**.

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

<parameter>: **(DebugPort)**
CJTAGFLAGS *<flags>*
CJTAGTCA *<value>*
CONNECTOR [**MIPI34** | **MIPI20T**]
CORE *<core>* *<chip>*
CoreNumber *<number>*
DEBUGPORT [**DebugCable0** | **DebugCableA** | **DebugCableB**]
DEBUGPORTTYPE [**JTAG** | **SWD** | **CJTAG** | **CJTAGSWD**]
NIDNTTRSTTORST [**ON** | **OFF**]

<parameter>: (DebugPort cont.)
NIDNTPSRISINGEDGE [**ON** | **OFF**]
NIDNTRSTPOLARITY [**High** | **Low**]
PortSHaRing [**ON** | **OFF** | **Auto**]
Slave [**ON** | **OFF**]
SWDP [**ON** | **OFF**]
SWDPIDLEHIGH [**ON** | **OFF**]
SWDPTargetSel *<value>*
DAP2SWDPTargetSel *<value>*
TriState [**ON** | **OFF**]

<parameter>: **(JTAG)**
CHIPDRLENGTH *<bits>*
CHIPDRPATTERN [**Standard** | **Alternate** *<pattern>*]
CHIPDRPOST *<bits>*
CHIPDRPRE *<bits>*
CHIPIRLENGTH *<bits>*
CHIPIRPATTERN [**Standard** | **Alternate** *<pattern>*]
CHIPIRPOST*<bits>*
CHIPIRPRE *<bits>*

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

<parameter>: (JTAG cont.)

DRPOST *<bits>*
DRPRE *<bits>*
ETBDRPOST *<bits>*
ETBDRPRE *<bits>*
ETBIRPOST *<bits>*
ETBIRPRE *<bits>*
IRPOST*<bits>*
IRPRE *<bits>*

<parameter>: (JTAG cont.)

NEXTDRPOST *<bits>*
NEXTDRPRE *<bits>*
NEXTIRPOST*<bits>*
NEXTIRPRE *<bits>*
RTPDRPOST *<bits>*
RTPDRPRE *<bits>*
RTPIRPOST *<bits>*
RTPIRPRE *<bits>*

<parameter>: (JTAG cont.)

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

<parameter>: **(Multitap)**

CFGCONNECT *<code>*
DAP2TAP *<tap>*
DAPTAP *<tap>*
DEBUGTAP *<tap>*
ETBTAP *<tap>*
MULTITAP [**NONE** | **IcepickA** | **IcepickB** | **IcepickC** | **IcepickD** | **IcepickBB** | **IcepickBC** | **IcepickCC** | **IcepickDD** | **STCLTAP1** | **STCLTAP2** | **STCLTAP3** |
MSMTAP *<irlength> <irvalue> <drlength> <drvalue>*
JtagSEQuence *<sub_cmd>*]
NJCR *<tap>*
RTPTAP *<tap>*
SLAVETAP *<tap>*

<parameter>: **(DAP)**

AHBACCESSPORT *<port>*
APBACCESSPORT *<port>*
AXIACCESSPORT *<port>*
COREJTAGPORT *<port>*
DAP2AHBACCESSPORT *<port>*
DAP2APBACCESSPORT *<port>*
DAP2AXIACCESSPORT *<port>*
DAP2COREJTAGPORT *<port>*

<parameter>: (DAP cont.)
DAP2DEBUGACCESSPORT *<port>*
DAP2JTAGPORT *<port>*
DAP2AHBACCESSPORT *<port>*
DEBUGACCESSPORT *<port>*
JTAGACCESSPORT *<port>*
MEMORYACCESSPORT *<port>*

<parameter>: **(COmponents)**
ADTF.Base *<address>*
ADTF.RESET
ADTF.Type [**NONE** | **ADTF** | **ADTF2** | **GEM**]
AET.Base *<address>*
AET.RESET
BMC.Base *<address>*
BMC.RESET
CMI.Base *<address>*
CMI.RESET

<parameter>: (COmponents cont.)
CMI.TraceID *<id>*
COREDEBUG.Base *<address>*
COREDEBUG.RESET
CTI.Base *<address>*
CTI.Config [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
CTI.RESET
DRM.Base *<address>*
DRM.RESET

<parameter>: (COmponents cont.)
DTM.RESET
DTM.Type [**None** | **Generic**]
DWT.Base *<address>*
DWT.RESET
EPM.Base *<address>*
EPM.RESET
ETB2AXI.Base *<address>*
ETB2AXI.RESET

<parameter>: (COmponents cont.)
ETB.ATBSource *<source>*
ETB.Base *<address>*
ETB.NoFlush [**ON** | **OFF**]
ETB.RESET
ETB.Size *<size>*
ETF.ATBSource *<source>*
ETF.Base *<address>*
ETF.RESET
ETM.Base *<address>*

<parameter>: (COmponents cont.)

ETM.RESET
ETR.ATBSource *<source>*
ETR.Base *<address>*
ETR.RESET
FUNNEL.ATBSource *<sourcelist>*
FUNNEL.Base *<address>*
FUNNEL.Name *<string>*
FUNNEL.PROGrammable [**ON** | **OFF**]

<parameter>: (Components cont.)

FUNNEL.RESET
HSM.Base *<address>*
HSM.RESET
HTM.Base *<address>*
HTM.RESET
HTM.Type [**CoreSight** | **WPT**]
ICE.Base *<address>*
ICE.RESET

<parameter>: (Components cont.)

ITM.Base *<address>*
ITM.RESET
L2CACHE.Base *<address>*
L2CACHE.RESET
L2CACHE.Type [**NONE** | **Generic** | **L210** | **L220** | **L2C-310** | **AURORA** | **AURORA2**]
OCP.Base *<address>*
OCP.RESET
OCP.TraceID *<id>*

<parameter>: (Components cont.)

OCP.Type *<type>*
PMI.Base *<address>*
PMI.RESET
PMI.TraceID *<id>*
RTP.Base *<address>*
RTP.PerBase *<address>*
RTP.RamBase *<address>*
RTP.RESET

<parameter>: (Components cont.)

SC.Base *<address>*
SC.RESET
SC.TraceID *<id>*
STM.Base *<address>*
STM.Mode [**NONE** | **XTIv2** | **SDTI** | **STP** | **STP64** | **STPv2**]
STM.RESET
STM.Type [**None** | **GenericARM** | **SDTI** | **TI**]
TPIU.ATBSource *<source>*
TPIU.Base *<address>*
TPIU.RESET
TPIU.Type [**CoreSight** | **Generic**]

<*parameter*>: **BMCBASE** *<address>*
(Deprecated) **BYPASS** *<seq>*
COREBASE *<address>*
CTIBASE *<address>*
CTICONFIG [**NONE** | **ARMV1** | **ARMPostInit** | **OMAP3** | **TMS570** | **CortexV1** | **QV1**]
DEBUGBASE *<address>*
DTMCONFIG [**ON** | **OFF**]

<*parameter*>: **DTMETBFUNNELPORT** *<port>*
(Deprecated cont.) **DTMFUNNEL2PORT** *<port>*
DTMFUNNELPORT *<port>*
DTMTPIUFUNNELPORT *<port>*
DWTBASE *<address>*
ETB2AXIBASE *<address>*
ETBBASE *<address>*

<*parameter*>: **ETBFUNNELBASE** *<address>*
(Deprecated cont.) **ETFBASE** *<address>*
ETMBASE *<address>*
ETMETBFUNNELPORT *<port>*
ETMFUNNEL2PORT *<port>*
ETMFUNNELPORT *<port>*
ETMTPIUFUNNELPORT *<port>*
FILLDRZERO [**ON** | **OFF**]

<*parameter*>: **FUNNEL2BASE** *<address>*
(Deprecated cont.) **FUNNELBASE** *<address>*
HSMBASE *<address>*
HTMBASE *<address>*
HTMETBFUNNELPORT *<port>*
HTMFUNNEL2PORT *<port>*
HTMFUNNELPORT *<port>*
HTMTPIUFUNNELPORT *<port>*

<*parameter*>: **ITMBASE** *<address>*
(Deprecated cont.) **ITMETBFUNNELPORT** *<port>*
ITMFUNNEL2PORT *<port>*
ITMFUNNELPORT *<port>*
ITMTPIUFUNNELPORT *<port>*
PERBASE *<address>*
RAMBASE *<address>*
RTPBASE *<address>*

<parameter>: (Deprecated cont.)	**SDTIBASE** *<address>* **STMBASE** *<address>* **STMETBFUNNELPORT***<port>* **STMFUNNEL2PORT***<port>* **STMFUNNELPORT***<port>* **STMTPIUFUNNELPORT***<port>* **TIADTFBASE** *<address>* **TIDRMBASE** *<address>*
<parameter>: (Deprecated cont.)	**TIEPMBASE** *<address>* **TIICEBASE** *<address>* **TIOCPBASE** *<address>* **TIOCPTYPE** *<type>* **TIPMIBASE** *<address>* **TISCBASE** *<address>* **TISTMBASE** *<address>*
<parameter>: (Deprecated cont.)	**TPIUBASE** *<address>* **TPIUFUNNELBASE** *<address>* **TRACEETBFUNNELPORT** *<port>* **TRACEFUNNELPORT***<port>* **TRACETPIUFUNNELPORT** *<port>* **view**

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 ARM, CevaX, TI DSP and Hexagon debugger. 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**.

Syntax Remarks

The commands are not case sensitive. Capital letters show how the command can be shortened.
Example: “SYStem.CONFIG.DWT.Base 0x1000” -> “SYS.CONFIG.DWT.B 0x1000”

The dots after “SYStem.CONFIG” can alternatively be a blank.
Example: “SYStem.CONFIG.DWT.Base 0x1000” or “SYStem.CONFIG DWT Base 0x1000”.

CJTAGFLAGS *<flags>*

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

Default: 0

CJTAGTCA *<value>*

Selects the TCA (TAP Controller Address) to address a device in a cJTAG Star-2 configuration. The Star-2 configuration requires a unique TCA for each device on the debug port.

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 uTrace 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 like ARM11MPCore, CortexA5MPCore, CortexA9MPCore and Scorpion 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** | **CJTAGSWD**]

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:

NIDNTPSRISINGEDGE
[**ON** | **OFF**]

Send data on rising edge for NIDnT PS switching.

NIDnT specifies how to switch, for example, the microSD card interface to a debug interface by sending in a special bit sequence via two pins of the microSD card.

TRACE32 will send the bits of the sequence incident to the falling edge of the clock, because TRACE32 expects that the target samples the bits on the rising edge of the clock.

Some targets will sample the bits on the falling edge of the clock instead. To support such targets, you can configure TRACE32 to send bits on the rising edge of the clock by using
SYStem.CONFIG NIDNTPSRISINGEDGE ON

NOTE: Only enable this option right before you send the NIDnT switching bit sequence.
Make sure to DISABLE this option, before you try to connect to the target system with for example **SYStem.Up**.

NIDNTRSTPOLARITY
[**High** | **Low**]

Usually TRACE32 requires that the system reset line of a target system is low active and has a pull-up on the target system.

When connecting via NIDnT to a target system, the reset line might be a high-active signal.
To configure TRACE32 to use a high-active reset signal, use
SYStem.CONFIG NIDNTRSTPOLARITY High

This option must be used together with
SYStem.CONFIG NIDNTTRSTTORST ON
because you also have to use the TRST signal of an ARM debug cable as reset signal for NIDnT in this case.

NIDNTTRSTTORST
[**ON** | **OFF**]

Usually TRACE32 requires that the system reset line of a target system is low active and has a pull-up on the target system.
This is how the system reset line is usually implemented on regular ARM-based targets.

When connecting via NIDnT (e.g. a microSD card slot) to the target system, the reset line might not include a pull-up on the target system.
To circumvent problems, TRACE32 allows to drive the target reset line via the TRST signal of an ARM debug cable.

Enable this option if you want to use the TRST signal of an ARM debug cable as reset signal for a NIDnT.

PortSHaRing [**ON** | **OFF** | **Auto**]

Configure if the debug port is shared with another tool, e.g. an ETAS ETK.

OFF: Default. Communicate with the target without sending requests.

ON: Request for access to the debug port and wait until the access is granted before communicating with the target.

Auto: Automatically detect a connected tool on next **SYStem.Mode Up**, **SYStem.Mode Attach** or **SYStem.Mode Go**. If a tool is detected switch to mode **ON** else switch to mode **OFF**.

The current setting can be obtained by the **PORTSHARING()** function, immediate detection can be performed using **SYStem.DETECT PortSHaRing**.

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).

DAP2SWDPTargetSel *<value>*

Device address of the second CoreSight DAP (DAP2) in case of a multidrop serial wire debug port (SWD).

Default: none set (any address accepted).

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

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

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

Default: OFF.

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

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

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

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

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

CHIPDRLENGTH *<bits>*

Number of Data Register (DR) bits which needs to get a certain BYPASS pattern.

CHIPDRPATTERN [**Standard** | **Alternate** *<pattern>*]

Data Register (DR) pattern which shall be used for BYPASS instead of the standard (1...1) pattern.

CHIPIRLENGTH *<bits>*

Number of Instruction Register (IR) bits which needs to get a certain BYPASS pattern.

CHIPIRPATTERN [**Standard** | **Alternate** *<pattern>*]

Instruction Register (IR) pattern which shall be used for BYPASS instead of the standard pattern.

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.

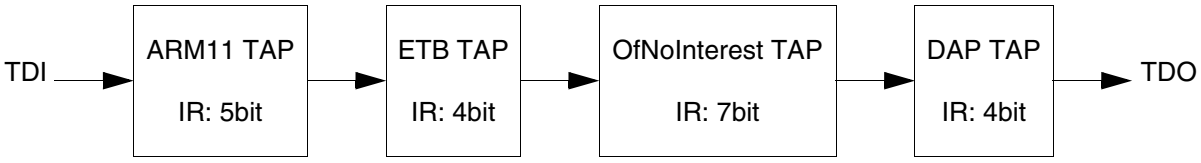
ETB (Embedded Trace Buffer) TAP if the ETB has its own TAP to access its control register (typical with ARM11 cores).
-> ETBDRPOST, ETBDRPRE, ETBIRPOST, ETBIRPRE.

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.

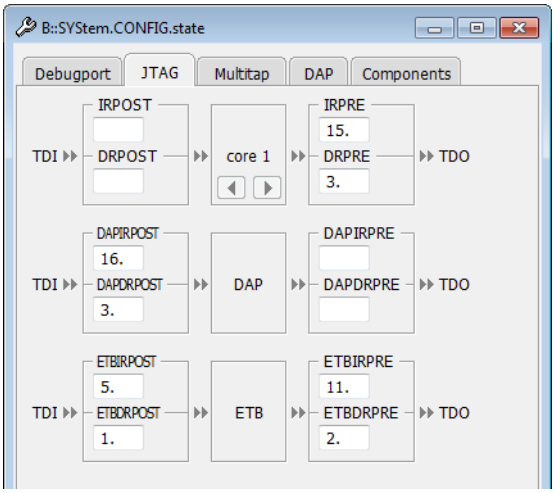
RTP (RAM Trace Port) TAP if the RTP has its own TAP to access its control register.
-> RTPDRPOST, RTPDRPRE, RTPIRPOST, RTPIRPRE.

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:



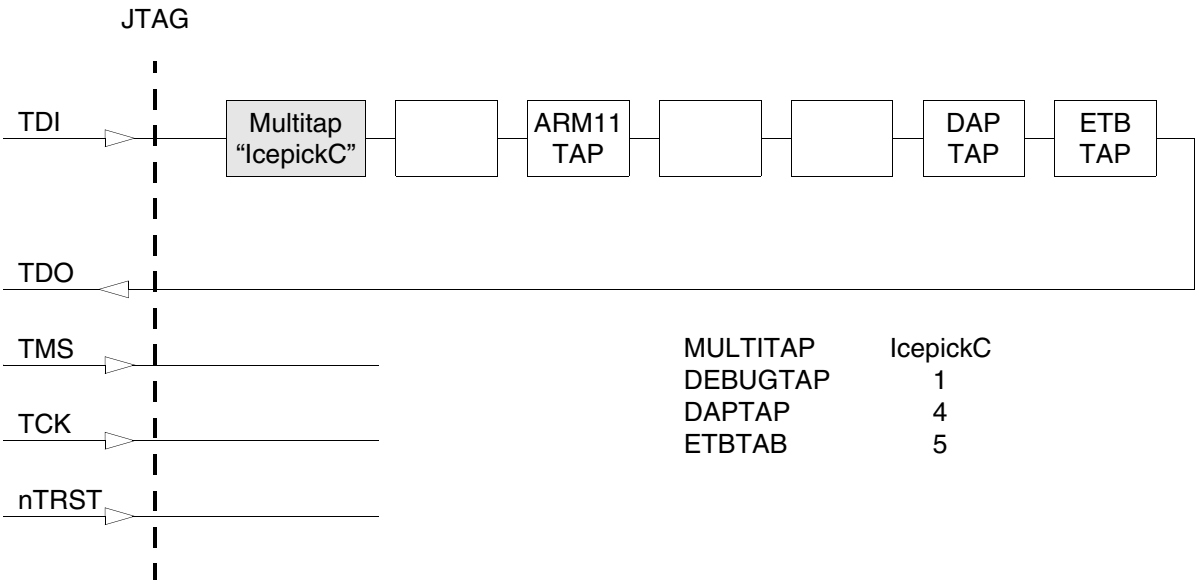

```
SYStem.CONFIG IRPRE 15.
SYStem.CONFIG DRPRE  3.
SYStem.CONFIG DAPIRPOST 16.
SYStem.CONFIG DAPDRPOST  3.
SYStem.CONFIG ETBIRPOST  5.
SYStem.CONFIG ETBDRPOST  1.
SYStem.CONFIG ETBIRPRE  11.
SYStem.CONFIG ETBDRPRE   2.
```



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:



CFGCONNECT *<code>*

The *<code>* is a hexadecimal number which defines the JTAG scan chain configuration. You need the chip documentation to figure out the suitable code. In most cases the chip specific default value can be used for the debug session.

Used if MULTITAP=STCLTAPx.

DAPTAP *<tap>*

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

Used if MULTITAP=Icepickx.

DAP2TAP *<tap>*

Specifies the TAP number which needs to be activated to get a 2nd 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.

Used if MULTITAP=Icepickx. ETB = Embedded Trace Buffer.

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

Selects the type and version of the MULTITAP.

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

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

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

NJCR *<tap>*

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

Used if MULTITAP=Icepickx.

RTPTAP *<tap>*

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

Used if MULTITAP=Icepickx. RTP = RAM Trace Port.

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).

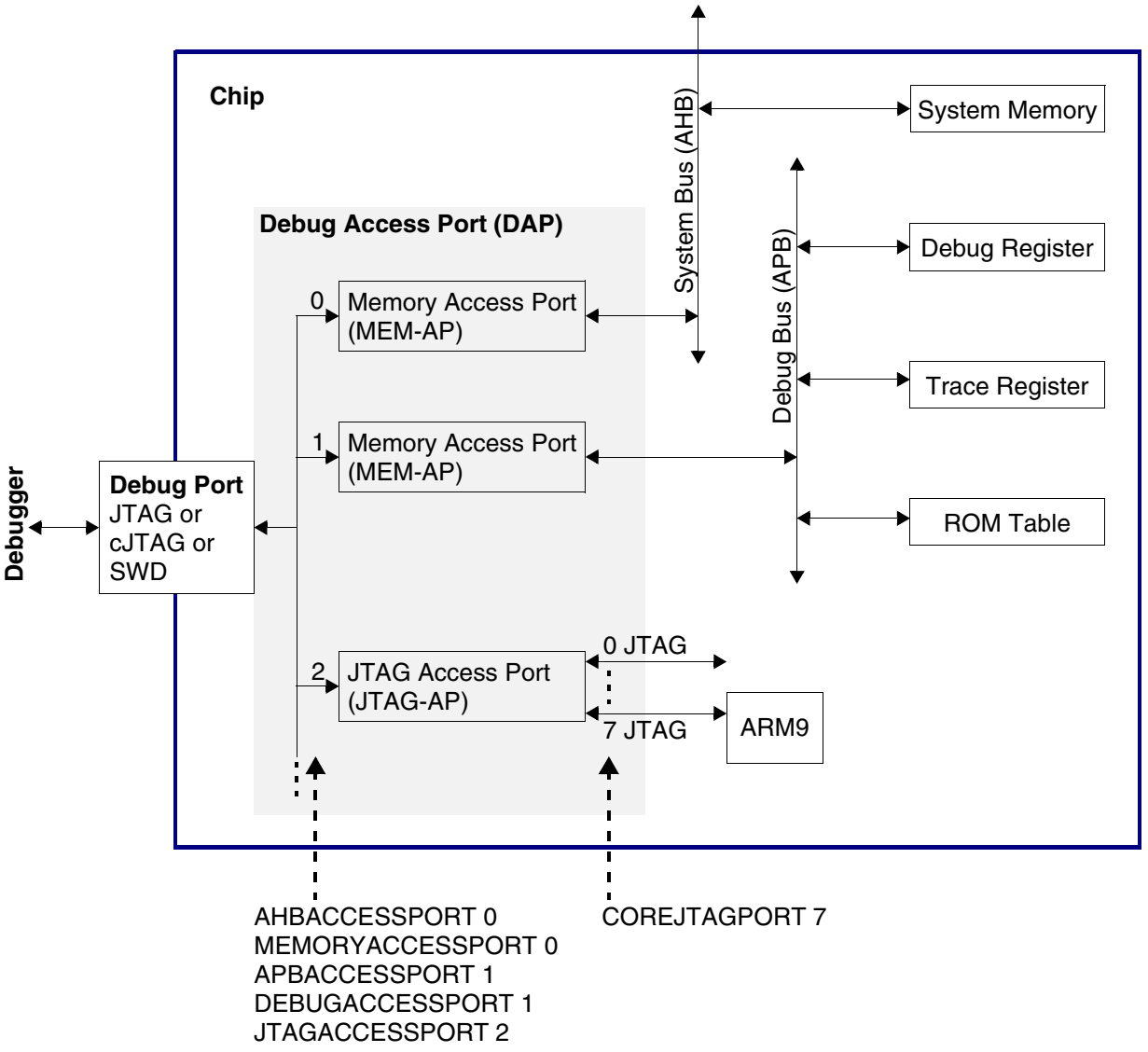
A Debug Access Port (DAP) is a CoreSight module from ARM which provides access via its debugport (JTAG, cJTAG, SWD) to:

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

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

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

Example:



AHBACCESSPORT *<port>* DAP access port number (0-255) which shall be used for “AHB:” access class. Default: *<port>*=0.

APBACCESSPORT *<port>* DAP access port number (0-255) which shall be used for “APB:” access class. Default: *<port>*=1.

AXIACCESSPORT *<port>* DAP access port number (0-255) which shall be used for “AXI:” access class. Default: port not available

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

DAP2AHBACCESSPORT *<port>*
DAP2 access port number (0-255) which shall be used for "AHB2:" access class. Default: *<port>*=0.

DAP2APBACCESSPORT *<port>*
DAP2 access port number (0-255) which shall be used for "APB2:" access class. Default: *<port>*=1.

DAP2AXIACCESSPORT *<port>*
DAP2 access port number (0-255) which shall be used for "AXI2:" access class. Default: port not available

DAP2DEBUGACCESS-PORT *<port>*
DAP2 access port number (0-255) where the debug register can be found (typically on APB). Used for "DAP2:" access class. Default: *<port>*=1.

DAP2COREJTAGPORT *<port>*
JTAG-AP port number (0-7) connected to the core which shall be debugged. The JTAG-AP can be found on another DAP (DAP2).

DAP2JTAGPORT *<port>*
JTAG-AP port number (0-7) for an (other) DAP which is connected to a JTAG-AP.

DAP2MEMORYACCESS-PORT *<port>*
DAP2 access port number where system memory can be accessed even during runtime (typically on AHB). Used for "E:" access class while running, assuming "SYStem.MemoryAccess DAP2". Default: *<port>*=0.

DEBUGACCESSPORT *<port>*
DAP access port number (0-255) where the debug register can be found (typically on APB). Used for "DAP:" access class. Default: *<port>*=1.

JTAGACCESSPORT *<port>*
DAP access port number (0-255) of the JTAG Access Port.

MEMORYACCESSPORT *<port>*
DAP access port number where system memory can be accessed even during runtime (typically on AHB). Used for "E:" access class while running, assuming "SYStem.MemoryAccess DAP". Default: *<port>*=0.

AHBNAME *<name>*
AHB bus transactor name that shall be used for "AHB:" access class.

APBNAME *<name>*
APB bus transactor name that shall be used for "APB:" access class.

AXINAME *<name>*
AXI bus transactor name that shall be used for "AXI:" access class.

DAP2AHBNAME *<name>*
AHB bus transactor name that shall be used for "AHB2:" access class.

DAP2APBNAME *<name>*
APB bus transactor name that shall be used for “APB2:” access class.

DAP2AXINAME *<name>*
AXI bus transactor name that shall be used for “AXI2:” access class.

DAP2DEBUGBUSNAME *<name>*
APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP2:” access class.

DAP2MEMORYBUSNAME *<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.MemoryAccess DAP2”.

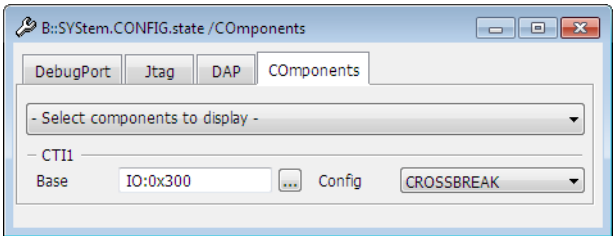
DEBUGBUSNAME *<name>*
APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP:” access class.

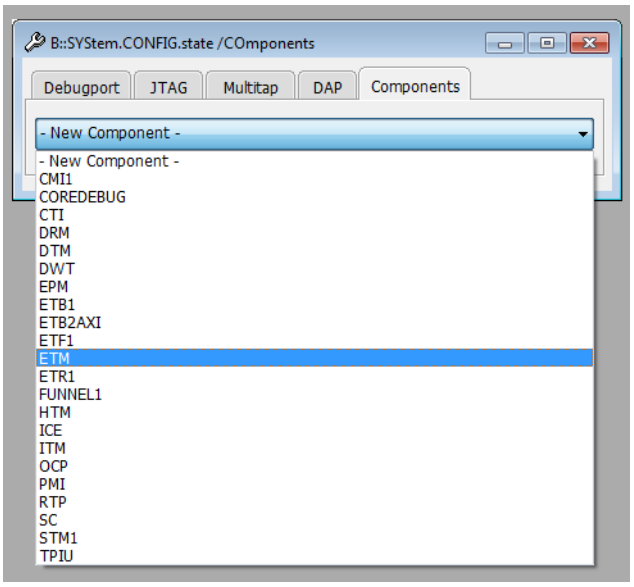
MEMORYBUSNAME *<name>*
AHB bus transactor name identifying the bus where system memory can be accessed even during runtime. Used for “E:” access class while running, assuming “SYStem.MemoryAccess DAP”.

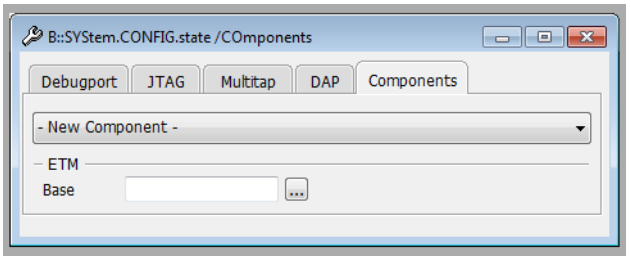
DAPNAME *<name>*
DAP transactor name that shall be used for DAP access ports.

DAP2NAME *<name>*
DAP transactor name that shall be used for DAP access ports of 2nd order.

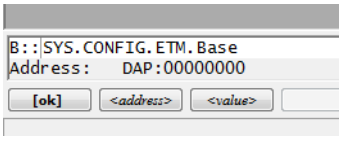
On the **Components** tab in the **SYStem.CONFIG.state** window, you can comfortably add the debug and trace components your chip includes and which you intend to use with the debugger’s help.







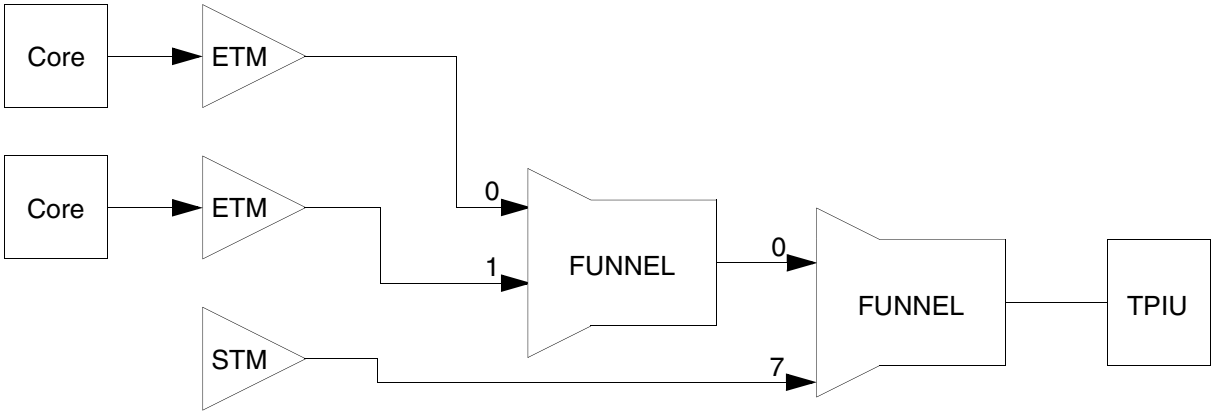
Each configuration can be done by a command in a script file as well. Then you do not need to enter everything again on the next debug session. If you press the button with the three dots you get the corresponding command in the command line where you can view and maybe copy it into a script file.



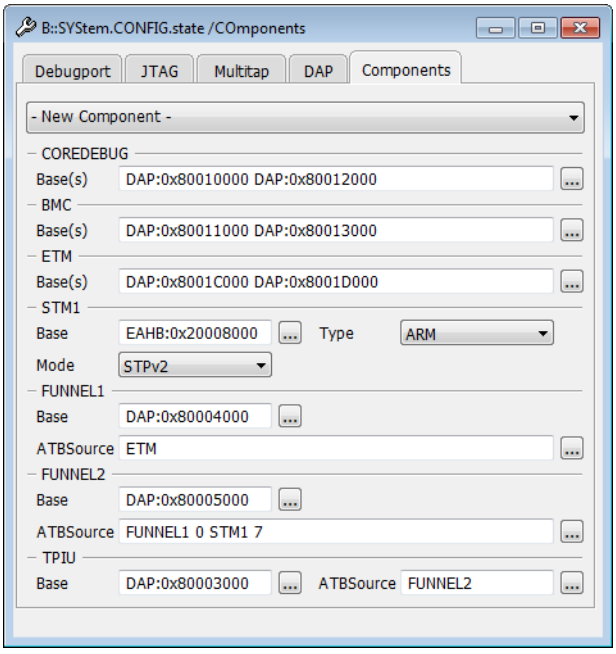
You can have several of the following components: CMI, ETB, ETF, ETR, FUNNEL, STM.
Example: FUNNEL1, FUNNEL2, FUNNEL3,...

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**.

… **.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**.

… **.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.

DTM.Type [**None** | **Generic**]

Informs the debugger that a customer proprietary Data Trace Message (DTM) module is available. This causes the debugger to consider this source when capturing common trace data. Trace data from this module will be recorded and can be accessed later but the unknown DTM module itself will not be controlled by the debugger.

ETB.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.
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.
ETM.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**]	IIn case the funnel can not or may not be programmed by the debugger, this option needs to be OFF. Default is ON.
HTM.Type [**CoreSight** | **WPT**]	Selects the type of the AMBA AHB Trace Macrocell (HTM). CoreSight is the type as described in the ARM CoreSight manuals. WPT is a NXP proprietary trace module.
L2CACHE.Type [**NONE** | **Generic** | **L210** | **L220** | **L2C-310** | **AURORA** | **AURORA2**]	Selects the type of the level2 cache controller. L210, L220, L2C-310 are controller types provided by ARM. AURORAx are Marvell types. The 'Generic' type does not need certain treatment by the debugger.
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.

AET.Base *<address>*
AET.RESET
Advanced Event Triggering unit (AET) - Texas Instruments
Trace source module of a TMS320C5x or TMS320C6x core delivering program and data trace information.

BMC.Base *<address>*
BMC.RESET
Performance Monitor Unit (PMU) - ARM debug module, e.g. on Cortex-A/R
Bench-Mark-Counter (BMC) is the TRACE32 term for the same thing.
The module contains counter which can be programmed to count certain events (e.g. cache hits).

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.

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

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

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

DTM.RESET
DTM.Type [**None** | **Generic**]
Data Trace Module (DTM) - generic, CoreSight compliant trace source module
If specified it will be considered in trace recording and trace data can be accessed afterwards.
DTM module itself will not be controlled by the debugger.

DWT.Base *<address>*
DWT.RESET
Data Watchpoint and Trace unit (DWT) - ARM debug module on Cortex-M cores
Normally fix address at 0xE0001000 (default).

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

ETB2AXI.Base *<address>*
ETB2AXI.RESET
ETB to AXI module
Similar to an ETR.

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

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

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

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

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

HSM.Base *<address>*
HSM.RESET
Hardware Security Module (HSM) - Infineon

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

ICE.Base *<address>*
ICE.RESET
ICE-Crusher (ICE) - Texas Instruments

ITM.Base *<address>*
ITM.RESET
Instrumentation Trace Macrocell (ITM) - ARM CoreSight module
Trace source delivering system trace information e.g. sent by software in printf() style.

L2CACHE.Base *<address>*
L2CACHE.RESET
L2CACHE.Type [**NONE** | **Generic** | **L210** | **L220** | **L2C-310** | **AURORA** | **AURORA2**]
Level 2 Cache Controller
The debugger might need to handle the controller to ensure cache coherency for debugger operation.

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.

RTP.Base *<address>*
RTP.PerBase *<address>*
RTP.RamBase *<address>*
RTP.RESET
RAM Trace Port (RTP) - Texas Instruments
Trace source delivering trace data about memory interface usage.

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**.

BYPASS *<seq>*

With this option it is possible to change the JTAG bypass instruction pattern for other TAPs. It works in a multi-TAP JTAG chain for the IRPOST pattern, only, and is limited to 64 bit. The specified pattern (hexadecimal) will be shifted least significant bit first. If no BYPASS option is used, the default value is “1” for all bits.

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.

DTMCONFIG [**ON** | **OFF**]	Informs the debugger that a customer proprietary Data Trace Message (DTM) module is available. This causes the debugger to consider this source when capturing common trace data. Trace data from this module will be recorded and can be accessed later but the unknown DTM module itself will not be controlled by the debugger.
FILLDRZERO [**ON** | **OFF**]	This changes the bypass data pattern for other TAPs in a multi-TAP JTAG chain. It changes the pattern from all “1” to all “0”. This is a workaround for a certain chip problem. It is available on the ARM9 debugger, only.
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.

Deprecated and New Commands

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

SYStem.CONFIG *<parameter>*

<parameter>: (Deprecated)	*<parameter>*: (New)
BMCBASE *<address>*	**BMC.Base** *<address>*
BYPASS *<seq>*	**CHIPIRPRE** *<bits>* **CHIPIRLENGTH** *<bits>* **CHIPIRPATTERN.Alternate** *<pattern>*
COREBASE *<address>*	**COREDEBUG.Base** *<address>*
CTIBASE *<address>*	**CTI.Base** *<address>*
CTICONFIG *<type>*	**CTI.Config** *<type>*
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.CPU — Select the used CPU

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**C280X** | **C2808** | …

Default selection: C280X.

Selects the processor type.

Format: **SYStem.JtagClock** [<*frequency*> | **CRTCK** | **RTCK** | **RTCK** <*frequency*>] | **ARTCK** <*frequency*>]
SYStem.BdmClock (deprecated)

<*frequency*>: **10000.** … **40000000.**

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.state** window. Besides a decimal number like "100000." short forms like "10kHz" or "15MHz" can also be used. The short forms imply a decimal value, although no "." is used.

CTCK With this option higher JTAG speeds can be reached.
The TDO signal will be sampled by a signal which derives from TCK, but which is timely compensated regarding the debugger-internal driver propagation delays (Compensation by TCK). This feature can be used with a debug cable versions 3b or newer. If it is selected, although the debug cable is not suitable, a fix JTAG clock will be selected instead (minimum of 10 MHz and selected clock).

```
SYStem.JtagClock CTCK 10MHz
```

CRTCK With this option higher JTAG speeds can be reached.
The TDO 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 is always output with the selected, fixed frequency.

```
SYStem.JtagClock CRTCK
```

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 cannot become higher than the physical interface can manage.

```
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.

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 TI DSP 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.

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

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.

Enable
CPU (deprecated) Used to activate the memory access while the CPU is running on the TRACE32 Instruction Set Simulator and on debuggers which do not have a fixed name for the memory access method.

Denied (default) Real-time memory access during program execution to target is disabled.

SYStem.Mode Establish the communication with the target

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

<*mode*>: **Attach**
Down
Go
NoDebug
Up

Attach The connection to the DSP is established without resetting the DSP.

Down (default) Disables the debugger. The state of the DSP remains unchanged. The JTAG port is tristated.

Go Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or external trigger.

NoDebug Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.

StandBy You need to be in DOWN state when switching to this mode. It resets and starts the program when power is detected. Halt the program execution and set all the breakpoints and trace conditions you need, then re-start the program. Now you can even debug a power cycle, because debug register (breakpoints and trace control) will be restored on power up. This mode is available on ARM7, ARM9 and ARM11 family.

Up Resets the DSP and establishes the connection. After the execution of this command the DSP is stopped and all register are set to the default level.

SYStem.Option AHBHPROT — Select AHB-AP HPROT bits

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

Default: 0

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

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

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

Default: OFF.

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.

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

<value>:
DEVICENONSHAREABLE
DEVICEINNERSHAREABLE
DEVICEOUTERSHAREABLE
DeviceSYStem
NonCacheableNonShareable
NonCacheableInnerShareable
NonCacheableOuterShareable
NonCacheableSYStem
WriteThroughNonShareable
WriteThroughInnerShareable
WriteBackOuterShareable
WRITETHROUGHSYSTEM

Default: 0

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

DEVICENONSHAREABLE	CSW.CACHE = 0x0, CSW.DOMAIN = 0x0
DEVICEINNERSHAREABLE	CSW.CACHE = 0x1, CSW.DOMAIN = 0x1
DEVICEOUTERSHAREABLE	CSW.CACHE = 0x1, CSW.DOMAIN = 0x2
DeviceSYStem	CSW.CACHE = 0x1, CSW.DOMAIN = 0x3
NonCacheableNonShareable	CSW.CACHE = 0x2, CSW.DOMAIN = 0x0
NonCacheableInnerShareable	CSW.CACHE = 0x3, CSW.DOMAIN = 0x1
NonCacheableOuterShareable	CSW.CACHE = 0x3, CSW.DOMAIN = 0x2
NonCacheableSYStem	CSW.CACHE = 0x3, CSW.DOMAIN = 0x3
WriteThroughNonShareable	CSW.CACHE = 0x6, CSW.DOMAIN = 0x0
WriteThroughInnerShareable	CSW.CACHE = 0xA, CSW.DOMAIN = 0x1
WriteBackOuterShareable	CSW.CACHE = 0xE, CSW.DOMAIN = 0x2
WRITETHROUGHSYSTEM	CSW.CACHE = 0xE, CSW.DOMAIN = 0x3

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

Default: 0

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

SYStem.Option DAPNOIRCHECK No DAP instruction register check

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

Default: OFF.

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

SYStem.Option DAPREMAP Rearrange DAP memory map

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

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

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

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

Default: SWITCHTOSWD.TryAll, SWDTRSTKEEP.DEFault.

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

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

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

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

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

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

Default: OFF.

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

SYStem.Option DAPDBGPWRUPREQ Force debug power in DAP

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

Default: ON.

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

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

Use case:

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

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

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 IMASKHLL Disable interrupts while HLL single stepping

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

Default: OFF.

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

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

This option can be used to force 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.RESetOut Reset target without reset of debug port

Format: **SYStem.RESetOut**

If possible (nRESET is open collector), this command asserts the nRESET line on the debug connector. This will reset the target including the CPU but not the debug port. The function only works when the system is in **SYStem.Mode.Up**.

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

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

Format: **TrOnchip.RESet**

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

ERAD — Embedded real-time analysis and diagnostic module

ERAD.OFF — Turn ERAD features off

Format: **ERAD.OFF**

Some TMS320F28xx devices include an “Embedded Real-time Analysis and Diagnostic” module. It provides 8 additional hardware breakpoints as well as benchmark counters (**“BMC”** (general_ref_b.pdf)).

If an application running on the device needs to access the ERAD module, the ERAD has to be turned off in TRACE32.

ERAD.ON — Turn ERAD features on

Format: **ERAD.ON**

Turns on the ERAD module features on if they were turned off before (see **ERAD.OFF**).

Usually the ERAD is automatically turned on if available and does not need to be turned on manually.

Mechanical Description of the 20-pin Debug Cable

This connector is defined by ARM. LAUTERBACH's debugger 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
VTREF	1	2	VSUPPLY(not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
SRST-	15	16	GND
DBGRQ	17	18	GND
DBGACK	19	20	GND

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 handover 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 47k 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 47k 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.

FAQ

Please refer to our Frequently Asked Questions page on the Lauterbach website.

Operation Voltage

Adapter	OrderNo	Voltage Range
JTAG Debugger for C2000 DSP (ICD)	LA-7847	1.8 .. 3.6 V