





# C7000 Debugger and Trace

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .....	
ICD In-Circuit Debugger .....	
Processor Architecture Manuals .....	
TI DSPs .....	
C7000 Debugger and Trace .....	1
History .....	5
Introduction .....	6
Brief Overview of Documents for New Users	6
Demo and Start-up Scripts	6
Converter from GEL to PRACTICE .....	7
Warning .....	8
DSP specific Implementations .....	9
Trigger	9
Breakpoints	9
Software Breakpoints	9
On-chip Breakpoints for Instructions	9
On-chip Breakpoints for Data	9
Access Classes	11
DSP specific SYStem Commands .....	13
SYStem.CONFIG.state	Display target configuration 13
SYStem.CONFIG	Configure debugger according to target topology 14
<parameters> describing the “DebugPort”	20
<parameters> describing the “JTAG” scan chain and signal behavior	23
<parameters> describing a system level TAP “MultiTap”	27
<parameters> configuring a CoreSight Debug Access Port “AP”	29
<parameters> describing debug and trace “Components”	35
<parameters> which are “Deprecated”	44
SYStem.CPU	Select the used CPU 48
SYStem.ENTERPostMortem	Place core into post-mortem state 48
SYStem.JtagClock	Define JTAG frequency 49
SYStem.LOCK	Tristate the JTAG port 50
SYStem.MemAccess	Select run-time memory access method 51

SYStem.MemFORCEREADY	Unblock memory access in post-mortem state	51
SYStem.Mode	Establish the communication with the target	52
SYStem.Option.Address32	Define address format display	53
SYStem.Option.AHBHPROT	Select AHB-AP HPROT bits	54
SYStem.Option.AXIACEEnable	ACE enable flag of the AXI-AP	54
SYStem.Option.AXICACHEFLAGS	Configure AXI-AP cache bits	54
SYStem.Option.AXIHPROT	Select AXI-AP HPROT bits	55
SYStem.Option.BigEndian	Enable big endian mode	55
SYStem.Option.DAPNOIRCHECK	No DAP instruction register check	55
SYStem.Option.DAPREMAP	Rearrange DAP memory map	56
SYStem.Option.DEBUGPORTOptions	Options for debug port handling	56
SYStem.Option.DUALPORT	Implicitly use run-time memory access	57
SYStem.Option.EnReset	Allow the debugger to drive nRESET (nSRST)	57
SYStem.Option.EnTRST	Control TAP reset	58
SYStem.Option.ICFLUSH	Invalidate instruction cache before go and step	58
SYStem.Option.IMASKASM	Disable interrupts while single stepping	58
SYStem.Option.INTDIS	Disable all interrupts	59
SYStem.Option.DAPDBGPWRUPREQ	Force debug power in DAP	60
SYStem.Option.DAPSYSPWRUPREQ	Force system power in DAP	60
SYStem.Option.ExecutionMode	Sets the CPU execution mode	61
SYStem.Option.IMASKHLL	Disable interrupts while HLL single stepping	61
SYStem.Option.MACHINESPACES	Address extension for guest OSeS	62
SYStem.Option.PWRDWN	Allow power-down mode	62
SYStem.RESetOut	Reset target without reset of debug port	63
<b>CPU specific BenchMarkCounter Commands</b>		<b>64</b>
BMC.<counter>.ATOB	Advise counter to count within AB-range	64
<b>CPU specific SETUP Command</b>		<b>65</b>
SETUP.DIS	Disassembler configuration	65
<b>TrOnchip Commands</b>		<b>66</b>
TrOnchip.state	Display on-chip trigger window	66
TrOnchip.CONVert	Adjust range breakpoint in on-chip resource	66
TrOnchip.RESet	Set on-chip trigger to default state	66
TrOnchip.VarCONVert	Adjust complex breakpoint in on-chip resource	67
<b>Tracing</b>		<b>68</b>
Controlling the Trace Capture		68
Trace Breakpoints		68
<b>Command reference: TRC</b>		<b>69</b>
TRC	Trace control (TRC)	69
TRC.CLEAR	Clear trace settings	69
TRC.CLOCK	Set core clock frequency for timing measurements	70
TRC.DataTrace	Define broadcast of load/store address tracing	71
TRC.OFF	Switch TRC off	71

TRC.ON	Switch TRC on	72
TRC.PCTrace	Enable program counter trace	72
TRC.RESet	Reset TRC settings	72
TRC.STALL	Stall processor to prevent FIFO overflow	72
TRC.StreamBuffer	Enable stream buffer trace	73
TRC.SyncPeriod	Set synchronization frequency	73
TRC.TlmeMode	Set timestamp configuration	73
TRC.TimeStampCLOCK	Specify frequency of the global timestamp	74
TRC.Trace	Enable TRC trace export	74
TRC.TraceID	Change the default ID for a TRC trace source	74
TRC.TracePriority	Define priority of TRC messages	74
TRC.state	Display TRC setup	75
<b>Target Adaption .....</b>		<b>76</b>
Probe Cables		76
Interface Standards JTAG, Serial Wire Debug, cJTAG		76
Connector Type and Pinout		76
Debug Cable		76
CombiProbe		76
Preprocessor		77
<b>FAQ .....</b>		<b>77</b>

## History

---

- 29-Mar-2024 New command [SYStem.Option.MACHINESPACES](#).
- 18-Mar-2024 New command [SYStem.Option.ICFLUSH](#).
- 28-Apr-2023 New command [SETUP.DIS](#).
- 07-Apr-2023 Removed command **SYStem.Option.PWRCHECK**.

# 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 C7000 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/c7000/` 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:**

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.

## 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
C71x	4	up to 4 single address	1 single address or range as bit mask	—

# Access Classes

This section describes the available ARM access classes and provides background information on how to create valid access class combinations in order to avoid syntax errors.

For background information, refer to “Access Classes” in TRACE32 Glossary, page 7 (glossary.pdf).

## Description of the Individual Access Classes

Access Class	Description
A	Absolute addressing (physical address)
AHB, AHB2	See DAP.
APB, APB2	See DAP.
AXI, AXI2	See DAP.
ECR	Access to C7000 extended configuration registers
D	Data Memory
DAP, DAP2, AHB, AHB2, APB, APB2, AXI, AXI2	<p>Memory access via bus masters, so named Memory Access Ports (MEM-AP), provided by a Debug Access Port (DAP).</p> <p>Which bus master (MEM-AP) is used by which access class (e.g. AHB) is defined by assigning a MEM-AP number to the access class:</p> <pre>SYStem.CONFIG DEBUGACCESSPORT &lt;mem_ap#&gt; -&gt; "DAP" SYStem.CONFIG AHBACCESSPORT &lt;mem_ap#&gt; -&gt; "AHB" SYStem.CONFIG APBACCESSPORT &lt;mem_ap#&gt; -&gt; "APB" SYStem.CONFIG AXIACCESSPORT &lt;mem_ap#&gt; -&gt; "AXI"</pre> <p>You should assign the memory access port connected to an AHB (AHB MEM-AP) to “AHB” access class, APB MEM-AP to “APB” access class and AXI MEM-AP to “AXI” access class. “DAP” should get the memory access port where the debug register can be found which typically is an APB MEM-AP.</p> <p>There is a second set of access classes (DAP2, AHB2, APB2, AXI2) and configuration commands (e.g. SYStem.CONFIG DAP2AHBACCESSPORT &lt;mem_ap#&gt;) available in case there are two DAPs which needs to be controlled by the debugger.</p>
E	Run-time memory access (see <a href="#">SYStem.MemAccess</a> )
G	Guest Mode

Access Class	Description
I	Intermediate address. Available on devices having Virtualization Extension.
N	Non-Secure Mode
P	Program Memory
S	Supervisor Memory (privileged access)
R	Root Mode (Hypervisor)
U	User Memory
VM	Virtual Memory (memory on the debug system)
Z	Secure Mode

SYStem.CONFIG.state

Display target configuration

Format:

SYStem.CONFIG.state [/<tab>]

<tab>:

DebugPort | Jtag | MultiTap | AccessPorts | CComponents

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 <b>SYStem.CONFIG.state</b> window on the specified tab. For tab descriptions, see below.
<b>DebugPort</b> (default)	<p>The <b>DebugPort</b> tab informs the debugger about the debug connector type and the communication protocol it shall use.</p> <p>For descriptions of the commands on the <b>DebugPort</b> tab, see <a href="#">DebugPort</a>.</p>
<b>Jtag</b>	<p>The <b>Jtag</b> 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.</p> <p>For descriptions of the commands on the <b>Jtag</b> tab, see <a href="#">Jtag</a>.</p>
<b>MultiTap</b>	<p>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.</p> <p>For descriptions of the commands on the <b>MultiTap</b> tab, see <a href="#">MultiTap</a>.</p>

<b>AccessPorts</b>	<p>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.</p> <p>For a descriptions of a corresponding commands, refer to <a href="#">AP</a>.</p>
<b>COmponents</b>	<p>The <b>COmponents</b> 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.</p> <p>For descriptions of the commands on the <b>COmponents</b> tab, see <a href="#">COmponents</a>.</p>

SYStem.CONFIG

Configure debugger according to target topology

Format:	<b>SYStem.CONFIG</b> <parameter> <b>SYStem.MultiCore</b> <parameter> (deprecated)
<parameter>: <a href="#">(DebugPort)</a>	<b>CJTAGFLAGS</b> <flags> (C7000 only) <b>CONNECTOR</b> [MIPI34   MIPI20T] (C7000 only) <b>CORE</b> <core> <chip> <b>CoreNumber</b> <number> <b>DEBUGPORT</b> [DebugCable0   DebugCableA   DebugCableB] <b>DEBUGPORTTYPE</b> [JTAG   SWD   CJTAG] <b>Slave</b> [ON   OFF] <b>SWDP</b> [ON   OFF] (C7000 only) <b>SWDPIdleHigh</b> [ON   OFF] <b>SWDPTargetSel</b> <value> <b>TriState</b> [ON   OFF]
<parameter>: <a href="#">(JTAG cont.)</a>	<b>DAPDRPOST</b> <bits> <b>DAPDRPRE</b> <bits> <b>DAPIRPOST</b> <bits> <b>DAPIRPRE</b> <bits>  <b>DRPOST</b> <bits> <b>DRPRE</b> <bits>  <b>ETBDRPOST</b> <bits> (C5000 only) <b>ETBDRPRE</b> <bits> (C5000 only) <b>ETBIRPOST</b> <bits> (C5000 only) <b>ETBIRPRE</b> <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.ACCEnable** [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>:**  
(CComponents)

**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>:**  
(CComponents  
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>  
**TIPMBASE** <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.

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

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

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

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

... **DRPOST** <bits>      Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TDI signal and the TAP you are describing. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below.

Default: 0.

... **DRPRE** <bits>      Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TAP you are describing and the TDO signal. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below.

Default: 0.

... **IRPOST** <bits>      Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between TDI signal and the TAP you are describing. See possible TAP types and example below.

Default: 0.

... **IRPRE** <bits>      Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between the TAP you are describing and the TDO signal. See possible TAP types and example below.

Default: 0.

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

**Slave [ON | OFF]**

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

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

Default: OFF.

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

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.

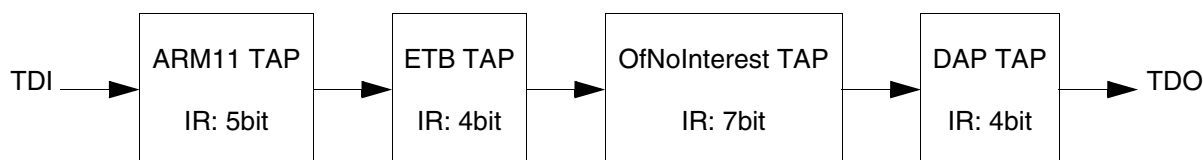
RTP (RAM Trace Port) TAP if the RTP has its own TAP to access its control register.

-> RTPDRPOST, RTPDRPRE, RTPIRPOST, RTPIRPRE.

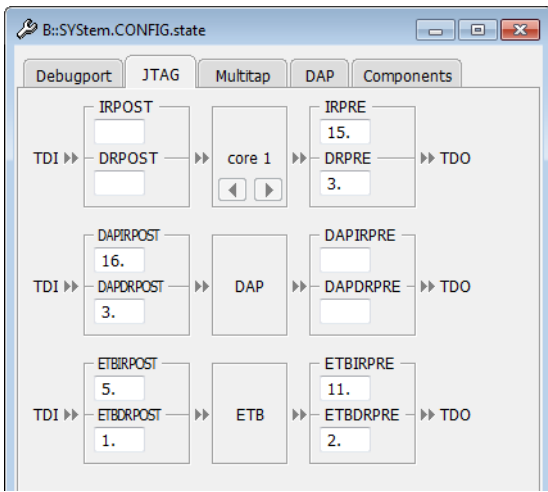
CHIP: Definition of a TAP or TAP sequence in a scan chain that needs a different Instruction Register (IR) and Data Register (DR) pattern than the default BYPASS (1...1) pattern.

-> CHIPDRPOST, CHIPDRPRE, CHIIRPOST, CHIIRPRE.

## 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.
```

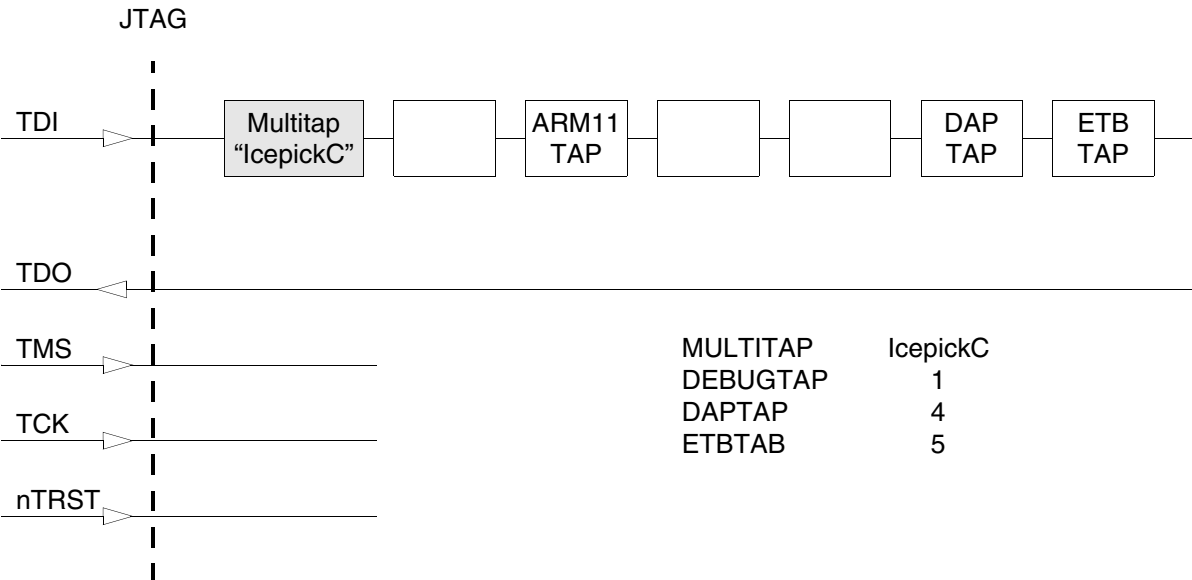


<parameters> describing a system level TAP “MultiTap”

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.

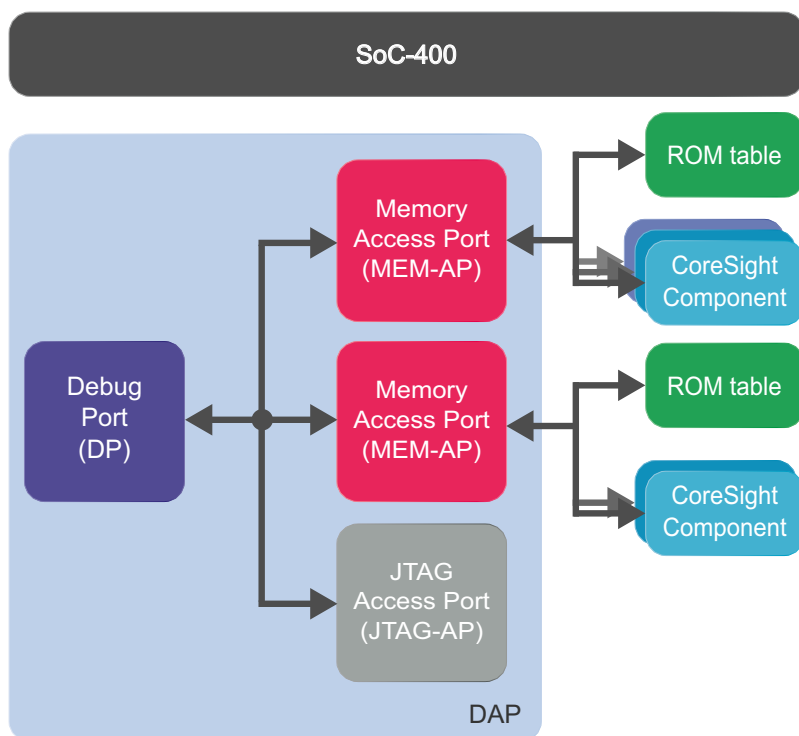
Used if MULTITAP=IcepickXY (two Icepicks).

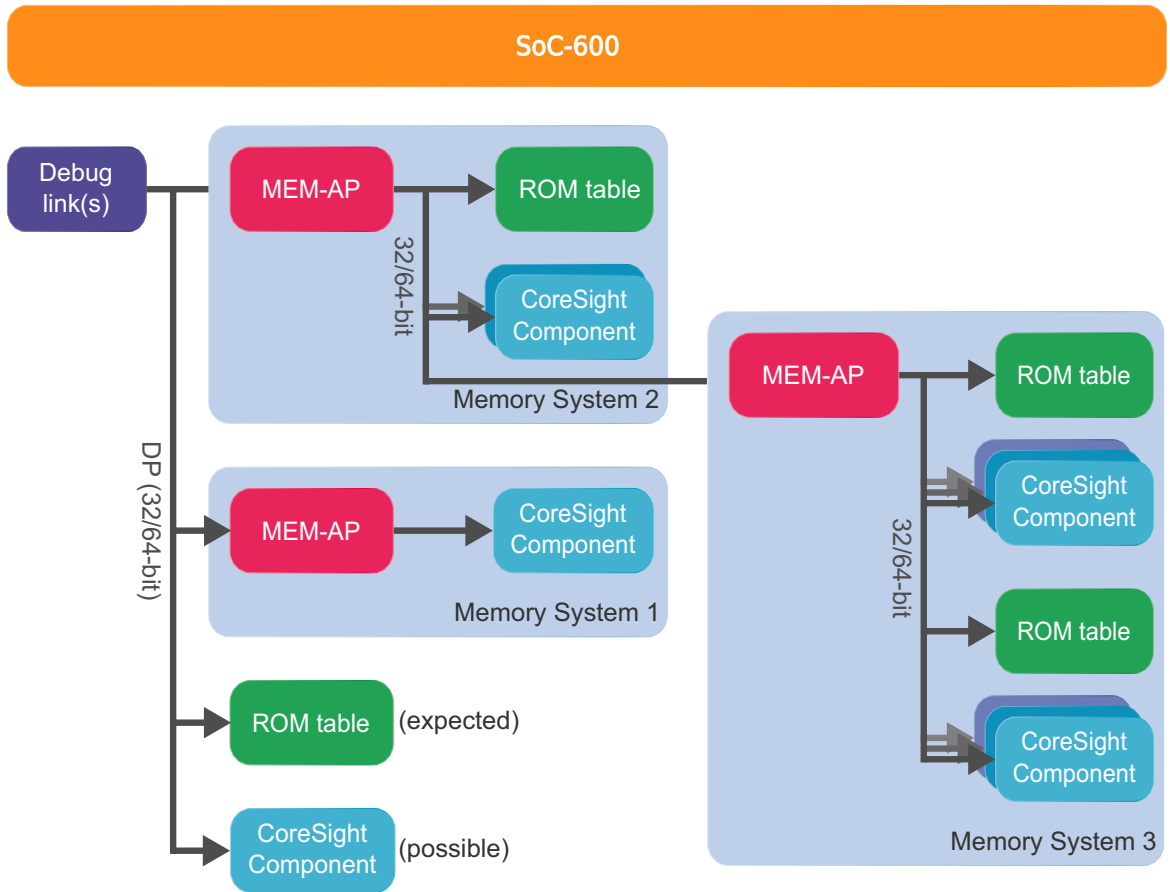
## <parameters> configuring a CoreSight Debug Access Port “AP”

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

1. Different memory busses (AHB, APB, AXI). This is especially important if the on-chip debug register needs to be accessed this way. You can access the memory buses by using certain access classes with the debugger commands: “AHB:”, “APB:”, “AXI:”, “DAP”, “E:”. The interface to these buses is called Memory Access Port (MEM-AP).
2. Other, chip-internal JTAG interfaces. This is especially important if the core you intend to debug is connected to such an internal JTAG interface. The module controlling these JTAG interfaces is called JTAG Access Port (JTAG-AP). Each JTAG-AP can control up to 8 internal JTAG interfaces. A port number between 0 and 7 denotes the JTAG interfaces to be addressed.
3. A transactor name for virtual connections to AMBA bus level transactors can be configured by the property **System.CONFIG.\*Apn.XtorName** <name>. A JTAG or SWD transactor must be configured for virtual connections to use the property “Port” or “Base” (with “DP:” access) in case XtorName remains empty.

### Example 1: SoC-400





**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.ACCEnable [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.AXI-CACHEFLAGS** <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
<b>DeviceSYStem</b>	=0x30: Domain=0x3, Cache=0x0
<b>NonCacheableSYStem</b>	=0x32: Domain=0x3, Cache=0x2
<b>ReadAllocateNonShareable</b>	=0x06: Domain=0x0, Cache=0x6
<b>ReadAllocateInnerShareable</b>	=0x16: Domain=0x1, Cache=0x6
<b>ReadAllocateOuterShareable</b>	=0x26: Domain=0x2, Cache=0x6
<b>WriteAllocateNonShareable</b>	=0x0A: Domain=0x0, Cache=0xA
<b>WriteAllocateInnerShareable</b>	=0x1A: Domain=0x1, Cache=0xA
<b>WriteAllocateOuterShareable</b>	=0x2A: Domain=0x2, Cache=0xA
<b>ReadWriteAllocateNonShareable</b>	=0x0E: Domain=0x0, Cache=0xE
<b>ReadWriteAllocateInnerShareable</b>	=0x1E: Domain=0x1, Cache=0xE
<b>ReadWriteAllocateOuterShareable</b>	=0x2E: Domain=0x2, Cache=0xE

<b>AHBAPn.XtorName</b> <name>	AHB bus transactor name that shall be used for “AHBn:” access class.
<b>APBAPn.XtorName</b> <name>	APB bus transactor name that shall be used for “APBn:” access class.
<b>AXIAPn.XtorName</b> <name>	AXI bus transactor name that shall be used for “AXIn:” access class.
<b>DEBUGAPn.XtorName</b> <name>	APB bus transactor name identifying the bus where the debug register can be found. Used for “DAP:” access class.
<b>MEMORYAPn.XtorName</b> <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 “ <b>SYStem.MemAccess DAP</b> ”.
<b>... .RESet</b>	Undo the configuration for this access port. This does not cause a physical reset for the access port on the chip.
<b>... .view</b>	Opens a window showing the current configuration of the access port.



**AHBAPn.Port** <port>  
**AHBACCESSPORT** <port>  
(deprecated)

Access Port Number (0-255) of a SoC-400 system which shall be used for “AHBn:” access class. Default: <port>=0.

**APBAPn.Port** <port>  
**APBACCESSPORT** <port>  
(deprecated)

Access Port Number (0-255) of a SoC-400 system which shall be used for “APBn:” access class. Default: <port>=1.

**AXIAPn.Port** <port>  
**AXIACCESSPORT** <port>  
(deprecated)

Access Port Number (0-255) of a SoC-400 system which shall be used for “AXIn:” access class. Default: port not available.

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

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

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

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

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

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

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

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

## SoC-600 Specific Commands

---

**AHBAPn.Base** <address>

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

**Example:** SYStem.CONFIG.AHBAP1.Base DP:0x80002000

Meaning: The control register block of the AHB access ports starts at address 0x80002000.

**APBAPn.Base** <address>

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

**Example:** SYStem.CONFIG.APBAP1.Base DP:0x80003000

Meaning: The control register block of the APB access ports starts at address 0x80003000.

**AXIAPn.Base** <address>

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

**Example:** SYStem.CONFIG.AXIAP1.Base DP:0x80004000

Meaning: The control register block of the AXI access ports starts at address 0x80004000.

**JTAGAPn.Base** <address>

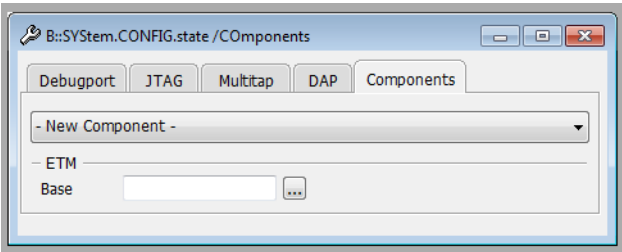
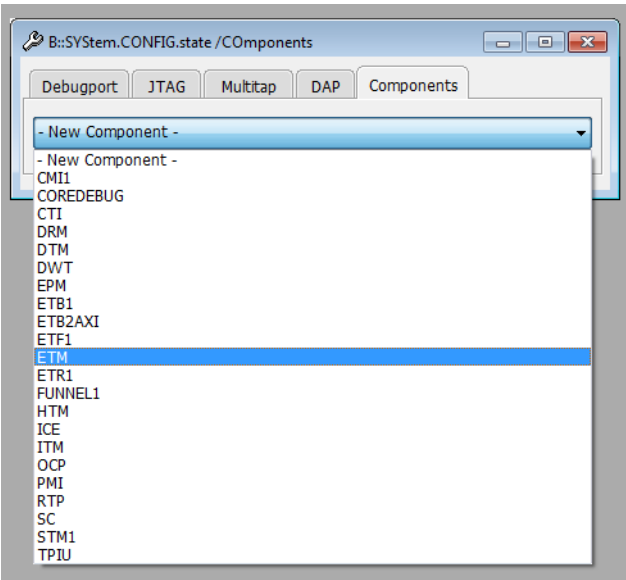
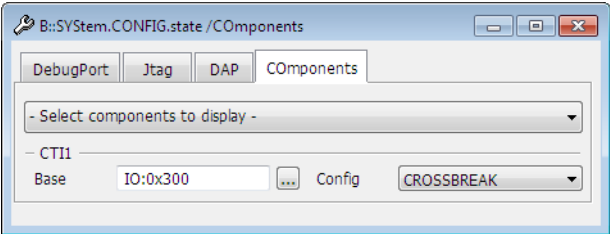
This command informs the debugger about the start address of the register block of the “JTAGAPn:” access port. And this way it notifies the existence of the access port. An access port typically provides a control register block which needs to be accessed by the debugger to read/write from/to the bus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

**Example:** SYStem.CONFIG.JTAGAP1.Base DP:0x80005000

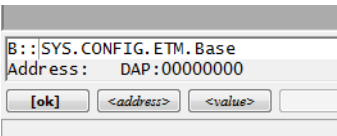
Meaning: The control register block of the JTAG access ports starts at address 0x80005000.

<parameters> describing debug and trace “Components”

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



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.

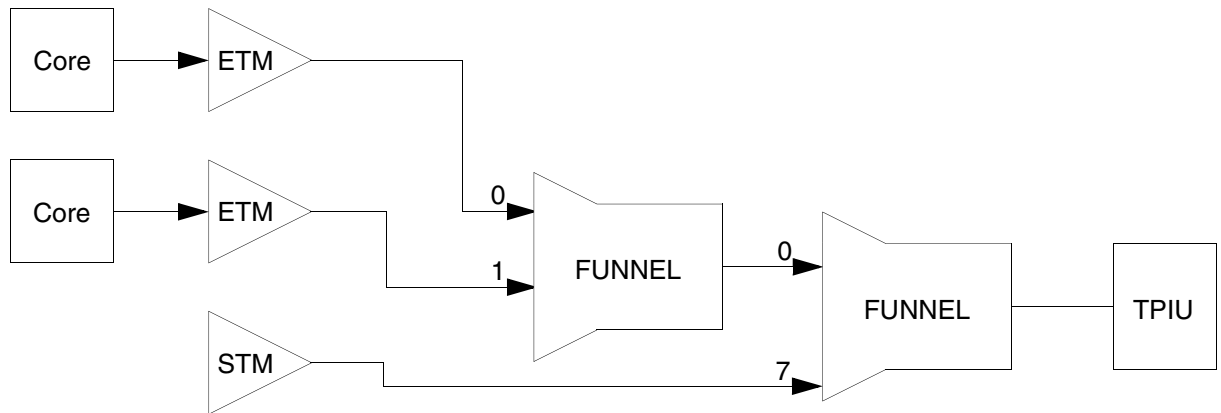


You can have several of the following components: CMI, ETB, ETF, ETR, FUNNEL, STM.

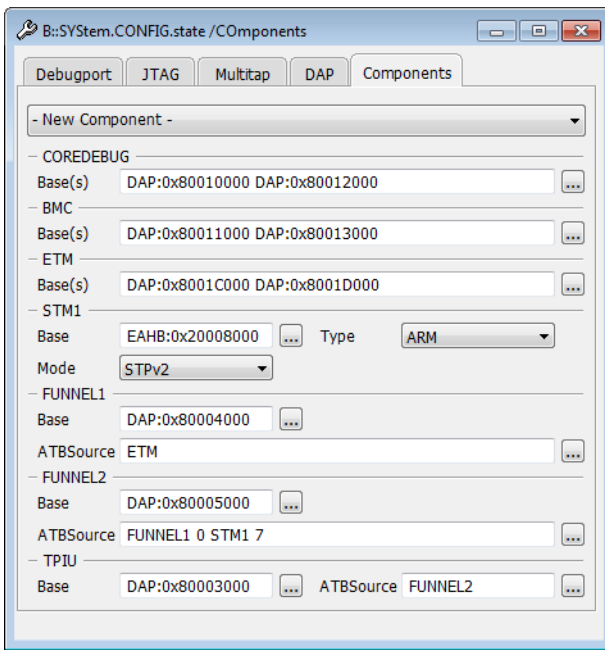
**Example:** FUNNEL1, FUNNEL2, FUNNEL3,...

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
```



... **.ATBSrc** <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 "... .ATBSrc <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.ATBSrc 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, COREDEBUG, 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>

Inform 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 DBGRRQ, 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** [**NotAvailable** | **TRGETM** | **FULLTIDRM** | **NOTSET** | **FULLSTOP** | **FULLCTI**]

Specifies the which method is used to implement the Stack mode of the on-chip trace.

**NotAvailable**: stack mode is not available for this on-chip trace.

**TRGETM**: the trigger delay counter of the onchip-trace is used. It starts by a trigger signal that must be provided by a trace source. Usually those events are routed through one or more CTIs to the on-chip trace.

**FULLTIDRM**: trigger mechanism for TI devices.

**NOTSET**: the method is derived by other GUIs or hardware. detection.

**FULLSTOP**: on-chip trace stack mode by implementation.

**FULLCTI**: on-chip trace provides a trigger signal that is routed back to on-chip trace over a CTI.

**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 | XTlv2 | 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 | XTlv2 | 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).

## <parameters> which are “Deprecated”

---

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

... **BASE** <address>

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

**Example:** SYStem.CONFIG ETMBASE APB:0x8011c000

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

In an SMP (Symmetric MultiProcessing) debug session you can enter for the components 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>

Inform 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>

Inform 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 DBGRRQ, 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)

**BMCBASE** <address>

**BYPASS** <seq>

**COREBASE** <address>

**CTIBASE** <address>

**DEBUGBASE** <address>

**DTMCONFIG** [ON | OFF]

**DTMETBFUNNELPORT** <port>

**DTMFUNNEL2PORT** <port>

**DTMFUNNELPORT** <port>

**DTMTPIUFUNNELPORT** <port>

**DWTBASE** <address>

**ETB2AXIBASE** <address>

**ETBBASE** <address>

**ETBFUNNELBASE** <address>

**ETFBASE** <address>

**ETMBASE** <address>

**ETMETBFUNNELPORT** <port>

**ETMFUNNEL2PORT** <port>

**ETMFUNNELPORT** <port>

**ETMTPIUFUNNELPORT** <port>

**FILLDRZERO** [ON | OFF]

**FUNNEL2BASE** <address>

**FUNNELBASE** <address>

**HSMBASE** <address>

<parameter>:  
(New)

**BMC.Base** <address>

**CHIPIRPRE** <bits>

**CHIPIRLENGTH** <bits>

**CHIPIRPATTERN.Alternate** <pattern>

**COREDEBUG.Base** <address>

**CTI.Base** <address>

**COREDEBUG.Base** <address>

**DTM.Type.Generic**

**FUNNEL4.ATBSource DTM** <port> (1)

**FUNNEL2.ATBSource DTM** <port> (1)

**FUNNEL1.ATBSource DTM** <port> (1)

**FUNNEL3.ATBSource DTM** <port> (1)

**DWT.Base** <address>

**ETB2AXI.Base** <address>

**ETB1.Base** <address>

**FUNNEL4.Base** <address>

**ETF1.Base** <address>

**ETM.Base** <address>

**FUNNEL4.ATBSource ETM** <port> (1)

**FUNNEL2.ATBSource ETM** <port> (1)

**FUNNEL1.ATBSource ETM** <port> (1)

**FUNNEL3.ATBSource ETM** <port> (1)

**CHIPDRPRE** 0

**CHIPDRPOST** 0

**CHIPDRLENGTH** <bits\_of\_complete\_dr\_path>

**CHIPDRPATTERN.Alternate** 0

**FUNNEL2.Base** <address>

**FUNNEL1.Base** <address>

**HSM.Base** <address>

**HTMBASE** <address>  
**HTMETBFUNNELPORT** <port>  
**HTMFUNNEL2PORT** <port>  
**HTMFUNNELPORT** <port>  
**HTMTPIUFUNNELPORT** <port>  
**ITMBASE** <address>  
**ITMETBFUNNELPORT** <port>  
**ITMFUNNEL2PORT** <port>  
**ITMFUNNELPORT** <port>  
**ITMTPIUFUNNELPORT** <port>  
**PERBASE** <address>  
**RAMBASE** <address>  
**RTPBASE** <address>  
**SDTIBASE** <address>  
  
**STMBASE** <address>  
  
**STMETBFUNNELPORT** <port>  
**STMFUNNEL2PORT** <port>  
**STMFUNNELPORT** <port>  
**STMTPIUFUNNELPORT** <port>  
**TIADTFBASE** <address>  
**TIDRMBASE** <address>  
**TIEPMBASE** <address>  
**TIICEBASE** <address>  
**TIOCPBASE** <address>  
**TIOCPTYPE** <type>  
**TIPMIBASE** <address>  
**TISCBASE** <address>  
**TISTMBASE** <address>  
  
**TPIUBASE** <address>  
**TPIUFUNNELBASE** <address>  
**TRACEETBFUNNELPORT** <port>

**HTM.Base** <address>  
**FUNNEL4.ATBSource** HTM <port> (1)  
**FUNNEL2.ATBSource** HTM <port> (1)  
**FUNNEL1.ATBSource** HTM <port> (1)  
**FUNNEL3.ATBSource** HTM <port> (1)  
**ITM.Base** <address>  
**FUNNEL4.ATBSource** ITM <port> (1)  
**FUNNEL2.ATBSource** ITM <port> (1)  
**FUNNEL1.ATBSource** ITM <port> (1)  
**FUNNEL3.ATBSource** ITM <port> (1)  
**RTP.PerBase** <address>  
**RTP.RamBase** <address>  
**RTP.Base** <address>  
**STM1.Base** <address>  
**STM1.Mode** SDTI  
**STM1.Type** SDTI  
  
**STM1.Base** <address>  
**STM1.Mode** STPV2  
**STM1.Type** ARM  
  
**FUNNEL4.ATBSource** STM1 <port> (1)  
**FUNNEL2.ATBSource** STM1 <port> (1)  
**FUNNEL1.ATBSource** STM1 <port> (1)  
**FUNNEL3.ATBSource** STM1 <port> (1)  
**ADTF.Base** <address>  
**DRM.Base** <address>  
**EPM.Base** <address>  
**ICE.Base** <address>  
**OCP.Base** <address>  
**OCP.Type** <type>  
**PMI.Base** <address>  
**SC.Base** <address>  
**STM1.Base** <address>  
**STM1.Mode** STP  
**STM1.Type** TI  
  
**TPIU.Base** <address>  
**FUNNEL3.Base** <address>  
**FUNNEL4.ATBSource** ADTF <port> (1)

**TRACEFUNNELPORT** <port>  
**TRACETPIUFUNNELPORT** <port>  
**view**

**FUNNEL1.ATBSource ADTF** <port> (1)  
**FUNNEL3.ATBSource ADTF** <port> (1)  
**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>:

**C7X, C71X ...**

Default selection: C7X.

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

**SYStem.ENTERPostMortem**

Place core into post-mortem state

Format:

**SYStem.ENTERPostMortem**

The post-mortem state can be used for hang recovery, when the CPU is hanging in “running” state and no CPU registers can be read. Executing this command will force the CPU into a halted state where all CPU and ECR registers can be read.

Warning: Performing “Go” in post-mortem state results in undefined behavior.



Format:	<b>SYStem.JtagClock</b> [ <i>&lt;frequency&gt;</i> ]   <b>CRTCK</b>   <b>RTCK</b>   <b>RTCK</b> [ <i>&lt;frequency&gt;</i> ]   <b>ARTCK</b> [ <i>&lt;frequency&gt;</i> ] <b>SYStem.BdmClock</b> (deprecated)
<i>&lt;frequency&gt;</i> :	<b>10000. ... 40000000.</b>

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 "10 kHz" or "15 MHz" can also be used. The short forms imply a decimal value, although no "." is used.

**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 (Compensation 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.

**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 (**Compensation 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).

## SYStem.LOCK

Tristate the JTAG port

Format:	<b>SYStem.LOCK [ON   OFF]</b>
---------	-------------------------------

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:	<b>SYStem.MemAccess</b> <mode>
<mode>:	<b>Enable</b> <b>Denied</b> <b>StopAndGo</b>

Default: Denied.

Allows to select a method for memory access while the CPU is running. If **SYStem.MemAccess** is not **Denied**, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program. For more information, see [SYStem.CpuBreak](#) and [SYStem.CpuSpot](#).

<b>Denied</b>	No memory access is possible while the CPU is executing the program.
<b>Enable CPU</b> (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.
<b>StopAndGo</b>	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.

**SYStem.MemFORCEREADY**

Unblock memory access in post-mortem state

Format:	<b>SYStem.MemFORCEREADY</b>
---------	-----------------------------

If the CPU has been placed in post-mortem state with [SYStem.ENTERPostMortem](#) this command allows to unblock memory accesses which are currently blocked by the memory system. If a subsequent memory access leads to a blocking state, the command has to be used again.

Format:	<b>SYStem.Mode</b> <mode>
	<b>SYStem.Attach</b> (alias for <b>SYStem.Mode Attach</b> ) <b>SYStem.Down</b> (alias for <b>SYStem.Mode Down</b> ) <b>SYStem.Up</b> (alias for <b>SYStem.Mode Up</b> )
<mode>:	<b>Down</b> <b>NoDebug</b> <b>Prepare</b> <b>Go</b> <b>Attach</b> <b>StandBy</b> <b>Up</b>

Default: Down.

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

<b>Down</b>	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
<b>NoDebug</b>	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated.
<b>Prepare</b>	<p>The debugger initializes the debug port (JTAG, SWD, cJTAG) and CoreSight DAP interface, but does not connect to the CPU.</p> <p>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.</p> <p>Typical use cases:</p> <ul style="list-style-type: none"><li>• The debugger accesses (physical) memory and bypasses the CPU if a mapping exists. Memory might require initialization before it can be accessed.</li><li>• 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.</li><li>• 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.</li></ul>
<b>Go</b>	<p>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.</p> <p>Program execution can, for example, be stopped by the <b>Break</b> command.</p>

<b>Attach</b>	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 <b>Break</b> command.
<b>StandBy</b>	<p>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.</p> <p>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.</p> <p>When a CPU power-down is detected, the debugger switches automatically back to the <b>StandBy</b> mode. This allows debugging of a power cycle because debug registers will be restored on power-up.</p> <p><b>NOTE:</b> 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.</p>
<b>Up</b>	<p>Resets the target via the reset line, initializes the debug port (JTAG, SWD, cJTAG), stops the CPU, and enters debug mode.</p> <p>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.</p>

## SYStem.Option.Address32

## Define address format display

Format: **SYStem.Option.Address32 [ON | OFF | AUTO | NARROW]**

Default: AUTO.

Selects the number of displayed address digits in various windows, e.g. **List.auto** or **Data.dump**.

<b>ON</b>	Display all addresses as 32-bit values. 64-bit addresses are truncated.
<b>OFF</b>	Display all addresses as 64-bit values.
<b>AUTO</b>	Number of displayed digits depends on address size.
<b>NARROW</b>	32-bit display with extendible address field.

Format:

**SYSystem.Option.AHBHPROT** <value> (deprecated)  
**Use SYSystem.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.

**SYSystem.Option.AXIACEEnable**

ACE enable flag of the AXI-AP

Format:

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

**SYSystem.Option.AXICACHEFLAGS**

Configure AXI-AP cache bits

Format:

**SYSystem.Option.AXICACHEFLAGS** <value> (deprecated)  
**Use SYSystem.CONFIG.AXIAPn.CacheFlags instead.**

Default: DeviceSYSystem (=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.

Format:

**SYSystem.Option.AXIHPROT** <value> (deprecated)  
Use **SYSystem.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.

**SYSystem.Option.BigEndian**

Enable big endian mode

Format:

**SYSystem.Option.BigEndian** [ON | OFF]

Switches endianness of memory access to Big Endian

- ON

Switches endianness to Big Endian
- OFF (default)

Switches endianness to Little Endian

**SYSystem.Option.DAPNOIRCHECK**

No DAP instruction register check

Format:

**SYSystem.Option.DAPNOIRCHECK** [ON | OFF]

Default: OFF.

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

Format:

SYSystem.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.

SYSystem.Option.DEBUGPORTOptions

Options for debug port handling

Format:

SYSystem.Option.DEBUGPORTOptions <option>

<option>:

SWICTHTOSWD.[TryAll | None | JtagToSwd | LuminaryJtagToSwd | DormantToSwd | JtagToDormantToSwd]  
SWDTRSTKEEP.[DEfault | LOW | HIGH]

Default: SWICTHTOSWD.TryAll, SWDTRSTKEEP.DEfault.

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

**SWICTHTOSWD** 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.



<b>DormantToSwd</b>	Switching procedure which is required if the device starts up in dormant state. The device has a dormant state but does not support JTAG.
<b>JtagToDormantToSwd</b>	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.

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

SYStem.Option.DUALPORT

Implicitly use run-time memory access

Format:	<b>SYStem.Option.DUALPORT</b> [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:	<b>SYStem.Option.EnReset</b> [ON   OFF]
---------	---

Default: ON.

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

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

**SYStem.Option.EnTRST**

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.ICFLUSH**

Invalidate instruction cache before go and step

[build 167635 - DVD 09/2024]

Format:

**SYStem.Option.ICFLUSH [ON | OFF]**

Default: OFF.

Invalidates the instruction cache before starting the target program (Step or Go).  
If this option is disabled, the debugger will not automatically invalidate the instruction cache before any debug command.

**SYStem.Option.IMASKASM**

Disable interrupts while single stepping

Format:

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

Default: OFF.

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

Format:	<b>SYStem.Option.INTDIS [ON   OFF]</b>
---------	--

Default: OFF.

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

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 <b>not</b> released at the end of the debug session, and the control bit is set to 0.
OFF	Only for test purposes: Debug power is <b>not</b> requested and <b>not</b> 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

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

**SYStem.Option.ExecutionMode**

Sets the CPU execution mode

Format:	<b>SYStem.Option.ExecutionMode</b> [StopMode   Real-Time]
---------	---

Default: StopMode.

The option controls the behavior of code execution and debug communication during debug events.

<b>StopMode</b>	The CPU can be halted at any point and only resumes execution after a <b>Go</b> command.
<b>Real-Time</b>	The CPU is not halted during real-time interrupts. If the CPU is in halted state and a real-time interrupt occurs, the CPU may leave the halted state to service the interrupt. The CPU then continues execution until all real-time interrupts have completed.

**SYStem.Option.IMASKHLL**

Disable interrupts while HLL single stepping

Format:	<b>SYStem.Option.IMASKHLL</b> [ON   OFF]
---------	--

Default: OFF.

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

SYStem.Option.MACHINESPACES

Address extension for guest OSes

Format:	SYStem.Option.MACHINESPACES [ON   OFF]
---------	--

Default: OFF

Enables the TRACE32 support for debugging virtualized systems. Virtualized systems are systems running under the control of a hypervisor.

After loading a Hypervisor Awareness, TRACE32 is able to access the context of each guest machine. Both currently active and currently inactive guest machines can be debugged.

If **SYStem.Option.MACHINESPACES** is set to **ON**:

- Addresses are extended with an identifier called **machine ID**. The machine ID clearly specifies to which host or guest machine the address belongs.  
The host machine always uses machine ID 0. Guests have a machine ID larger than 0. TRACE32 currently supports machine IDs up to 30.
- The debugger address translation (**MMU** and **TRANSlation** command groups) can be individually configured for each virtual machine.
- Individual symbol sets can be loaded for each virtual machine.

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.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**.

# CPU specific BenchMarkCounter Commands

The benchmark counters can be read at run-time.

For information about the *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> ClockCycles           ; <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:

SETUP.DIS [*<fields>*] [*<bar>*] [*<constants>*]

*<constants>*:

[LowerCase | UpperCase] [*<other\_constants>*]

Sets **default values** for configuring the disassembler output of **newly opened windows**.

The command does **not affect existing windows** containing disassembler output.

<i>&lt;fields&gt;</i> , <i>&lt;bar&gt;</i> , <i>&lt;other_constants&gt;</i>	For a description of the generic arguments, see <a href="#">SETUP.DIS</a> in <a href="#">general_ref_s.pdf</a> .
LowerCase	Default. Sets the mnemonics to lower case.
UpperCase	Sets the mnemonics to upper case. This format is closer to the TI disassembler.

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

TrOnchip.RESet

Set on-chip trigger to default state

Format:

TrOnchip.RESet

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

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

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

# Tracing

The processor trace for C7000 is based on the ARM Coresight architecture. The trace can either be stored in trace buffers on the chip (e.g. ETB) or sent out through an external trace port (TPIU). For further information about Coresight component configuration please refer to **“Setup of the Debugger for a CoreSight System”** (app\_arm\_coresight.pdf).

## Controlling the Trace Capture

On the C7x core the trace capture is controlled by the **TRC** and **AET** command groups. The **TRC** commands control trace export and stream selection, whereas the **AET** commands control triggers and events.

## Trace Breakpoints

The following breakpoints use TRC and 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
```

...

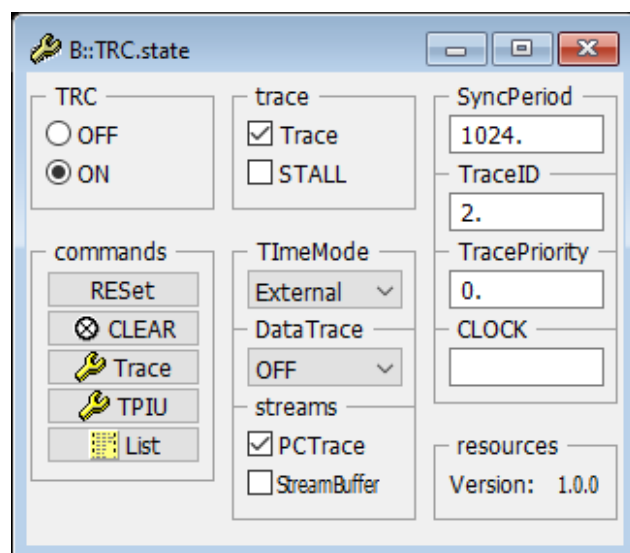
```
Break.Delete                                ; delete breakpoint
```

## TRC

## Trace control (TRC)

The **TRC** command group implements the trace specific functions for the C7000 debugger.

For configuration, use the TRACE32 command line, a PRACTICE script (\*.cmm), or the **TRC.state** window.



The following TRACE32 commands are available to configure the C7x trace.

## TRC.CLEAR

## Clear trace settings

Format: **TRC.CLEAR**

Switches the TRC ON, clears the trace and clears all settings.

Format: **TRC.CLOCK** *<frequency>*  
(alias for **<trace>.CLOCK**)

Tells the debugger the core clock frequency of the traced DSP core.

- If the timing information is based on external timestamps with **TRC.TimeMode** set to **External**, this setting is used to calculate the elapsed clock cycles from the elapsed time in seconds.
- If the timing information is based on asynchronous internal timestamps (**TRC.TimeMode AsyncTimeStamp**), this setting is used to calculate the elapsed clock cycles from the elapsed time in seconds.
- For timing modes which combine timestamps with cycle count information, this setting is not required.

Format:

TRC.DataTrace <def>

<def>:

OFF  
ReadAddress  
WriteAddress  
Address  
ReadData  
WriteData  
Data  
Read  
Write  
Only  
ON

Defines which data access information is included in the trace.

TRC.DataTrace	Trace Data Values of Read Accesses	Trace Data Values of Write Accesses	Trace <b>Addresses</b> of Read Accesses	Trace <b>Addresses</b> of Write Accesses
OFF				
ReadAddress			■	
WriteAddress				■
Address			■	■
ReadData	■			
WriteData		■		
Data	■	■		
Read	■		■	
Write		■		■
ON	■	■	■	■

TRC.OFF

Switch TRC off

Format:

TRC.OFF

Disables TRC functionality.

There is no need to disable the TRC functionality. In case of general problems with a new processor or if a TRC power-down should be enforced it can be reasonable.

TRC.ON

Switch TRC on

Format:TRC.ON

Enables TRC functionality.

TRC.PCTrace

Enable program counter trace

Format:TRC.PCTrace [ON | OFF]

ON	The TRC exports the program counter trace stream (default).
OFF	The TRC does not export the program counter trace stream.

TRC.RESet

Reset TRC settings

Format:TRC.RESet

Reset the TRC settings and clear the TRC (see also [TRC.Clear](#))

TRC.STALL

Stall processor to prevent FIFO overflow

Format:TRC.STALL [ON | OFF]

Allows the TRC to stall the processor to prevent an output FIFO overflow. If enabled, the trace will be no longer real time.



Format:

TRC.StreamBuffer [ON | OFF]

- ON
- The TRC exports the stream buffer trace stream (default).
- OFF
- The TRC does not export the stream buffer trace stream.

Format:

TRC.SyncPeriod [<period>]

Sets the period in bytes for synchronisation packets in the trace stream. The default frequency is 1024 bytes.

Format:

TRC.TimeMode <mode>

<mode>:

OFF | External | AsyncTimeStamps

Defines which timing information is included in the trace stream.

External	The timestamps are generated in the debug hardware (default)
OFF	No timestamp information is included in the trace
AsyncTimeStamps	The timestamps are generated by the TRC unit

Format:

TRC.TimeStampClock <frequency>

If the trace infrastructure of the SoC provides a global timestamp and if the clock of the global timestamp is different from the core clock, TRACE32 needs to know the frequency of this timestamp.

Format:

TRC.Trace [ON | OFF]

ON	The TRC exports trace information (default).
OFF	The TRC is not exporting trace information.

Format:

TRC.TraceID <id>

By default TRACE32 automatically assigns a trace source ID to all cores with a CoreSight trace source. The command **TRC.TraceID** allows to assign an ID to a trace source, overriding the defaults.

Format:

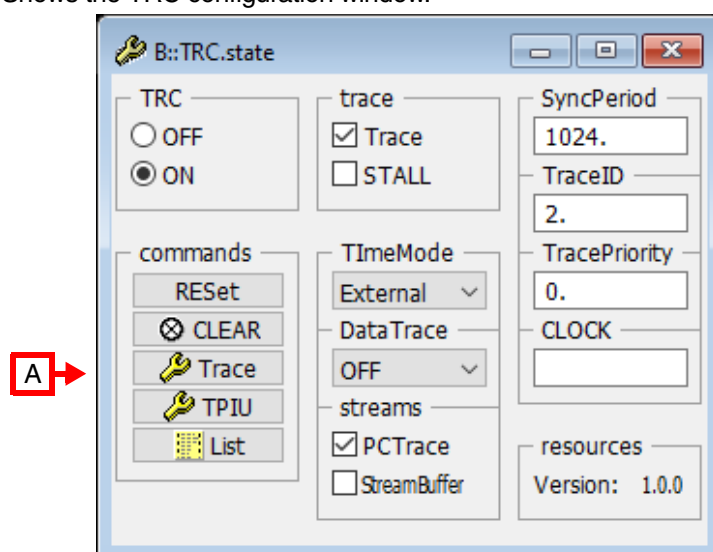
TRC.TracePriority <priority>

The CoreSight Trace Funnel combines 2 to 8 ATB input ports to a single ATB output. An arbiter determines the priority of the ATB input port. A priority value of 0 defines the highest priority and 7 the lowest.

The command **TRC.TracePriority** allows to change the default priority of an ATB input port.

Format: **TRC.state**

Shows the TRC configuration window.



**A** For descriptions of the commands in the **TRC.state** window, please refer to the **TRC.\*** commands in this chapter. **Example:** For information about **ON**, see [TRC.ON](#).

#### Exceptions:

- The **Trace** button opens the main trace control window ([Trace.state](#)).
- The **TPIU** button opens the [TPIU.state](#) window.
- The **List** button opens the main trace list window ([Trace.List](#)).

# Target Adaption

---

## Probe Cables

---

For debugging two kind of probe cable can be used to connect the debugger to the target: “Debug Cable” and “CombiProbe”

For off-chip program and data trace an additional trace probe cable “Preprocessor” is needed.

## Interface Standards JTAG, Serial Wire Debug, cJTAG

---

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

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

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

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

## Connector Type and Pinout

---

### Debug Cable

---

Adaptation for ARM Debug Cable: See <https://www.lauterbach.com/adarmdbg.html>.

For details on logical functionality, physical connector, alternative connectors, electrical characteristics, timing behavior and printing circuit design hints refer to “**Arm Debug and Trace Interface Specification**” (app\_arm\_target\_interface.pdf).

### CombiProbe

---

Adaptation for ARM CombiProbe: See <https://www.lauterbach.com/adarmcombi.html>.

The CombiProbe will always be delivered with 10-pin, 20-pin, 34-pin connectors. The CombiProbe can not detect which one is used. If you use the trace of the CombiProbe you need to inform about the used connector because the trace signals can be at different locations: **SYStem.CONFIG CONNECTOR [MIPI34 | MIPI20T]**.

If you use more than one CombiProbe cable (twin cable is no standard delivery) you need to specify which one you want to use by **SYStem.CONFIG DEBUGPORT [DebugCableA | DebugCableB]**. The CombiProbe can detect the location of the cable if only one is connected.

## Preprocessor

---

Adaptation for ARM ETM Preprocessor Mictor: See <https://www.lauterbach.com/adetmmictor.html>.

Adaptation for ARM ETM Preprocessor MIPI-60: See <https://www.lauterbach.com/adetmmipi60.html>

## FAQ

---

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